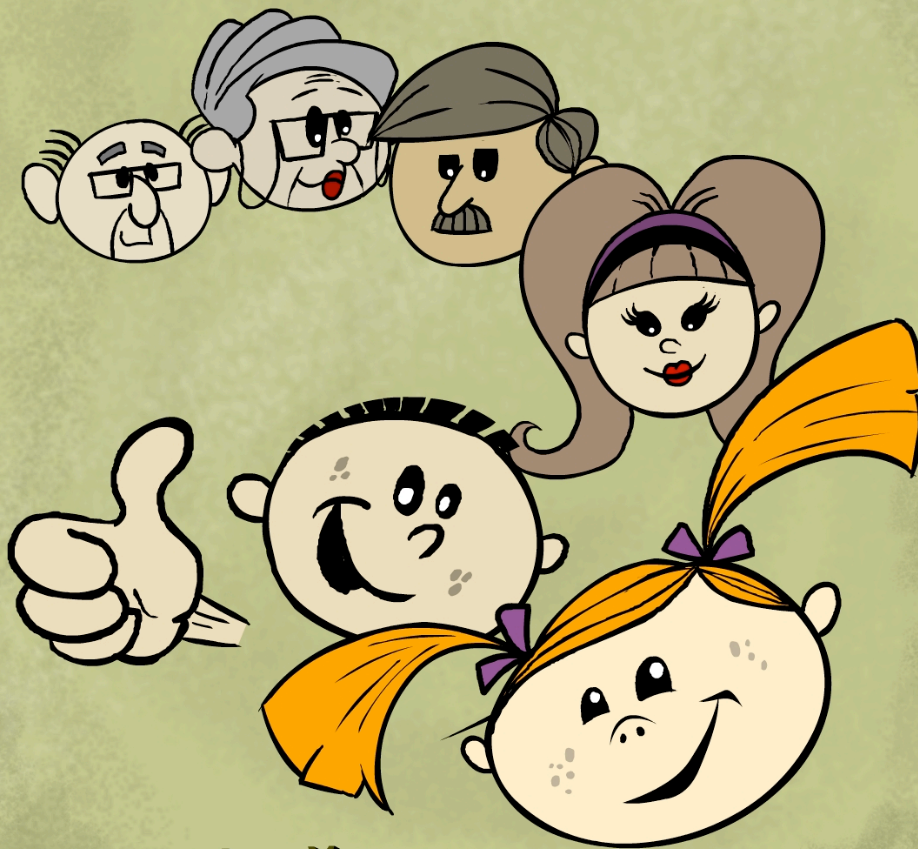
Программирование на JAVA для детей, родителей, бабушек и дедушек



Яков Файн

Java Programming for Kids, Parents and Grandparents

by Yakov Fain

Copyright © 2011 Yakov Fain

All rights reserved. No part of this book may be reproduced, in any form or by any, without permission in writing from the publisher.

Cover design and illustrations: Yuri Fain

Adult technical editor: Yuri Goncharov

Kid technical editor: David Fain

May 2004: First Electronic Edition (English)

June 2005: Second Electronic Edition (French)
Programmation Java pour les enfants, les parentset les grands-parents

October 2011: Third Electronic Edition (Russian)
Программирование на Java для детей, родителей, дедушек и бабушек

The information in this book is distributed without warranty. Neither the author nor the publisher shall have any liability to any person or entitle to any liability, loss or damage to be caused directly or indirectly by instructions contained in this book or by the computer software or hardware products described herein.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the United States and other countries.

Windows 7 and Windows XP are trademarks of Microsoft Corporation.

All other product names and company names are the property of their respective owners.

ISBN: 0-9718439-5-3

Предисловие к русскому изданию

Здравствуйте дети, родители, а также родители родителей. Спасибо, что взяли в руки (хоть и виртуально) мою книжку. Написал я ее в 2004 году по-английски, ибо живу я в США и первым читателем этой книги должен был стать мой младший сын Дейв, для которого английский – основной язык общения. Книжка эта была выложена бесплатно в электронном виде. Рисунки к ней нарисовал мой старший сын Юрий, который тогда еще учился в колледже на мультипликатора. Сейчас он уже работает, а сайт его http://yurifain.com. Специально для русского издания Юрий нарисовал новую обложку. Нравится?

Книжка пользовалась успехом во всем англоязычном мире, а через год-другой ее перевели и на французский язык. Прошло много лет, и я стал записывать аудио подкасты на русском языке на всякие житейские темы. Подкасты – это mp3 файлы, которые можно слушать на любом аудио плеере или прямо на компе.

Подкасты мои выкладываются в интернете по адресу http://americhka.us и слушают их тысячи русскоговорящих людей по всему миру. И вот, однажды, я спросил, не найдется-ли среди моих подслушателей 3-4 человека, которые и английский хорошо знают, и на языке Java программировать умеют, и готовы в жесткие сроки бесплатно перевести по паре глав на русский язык. На следующий день мне написали пять человек, сказав, что они и знают, и умеют, и готовы. Вот имена этих добрых людей:

Александр Коноплев
Денис Лунев
Константин Медведенко
Юрий Ополев
Александр Тетерин

Спасибо вам, ребята, за то, что слово сдержали, а главное, выполнили работу в срок! Ибо есть много мальчиков с моторчиками, которые быстро загораются, берутся за работу, а потом, также быстро их интерес пропадает. К счастью, к нашим переводчикам это не относится. Когда перевод был закончен, я его перечитал и слегка отредактировал. Затем один сибиряк по имени Юрий Мякотин снова перечитал и отредактировал текст, так что, если вы заметите сибирский акцент, то знайте, кого винить ☺.

В конце каждой главы есть практические упражнения и ссылки материалы для дополнительного чтения, правда на английском языке. Если вы серьезно относитесь к изучению программирования, то я вам советую и английский свой подтянуть. В мире программистов английский – это основной язык общения. Я

знаю, что и в России есть много хороших программистов и авторов пишущих по русски. Но зачем себя ограничивать, когда есть столько полезной технической литературы на английском?

На самом деле, задача была сложнее, чем перевод и легкое редактирование. За семь лет, прошедших после выхода английской книжки, язык программирования Java несколько изменился, и нужно было внести соответствующие корректировки в разных местах, что и было сделано. Поэтому, пользуясь издательским языком, можно смело сказать, что это издание исправленное и дополненное.

Исправленное-то оно исправленное, но, все-же я вполне допускаю, что несколько опечаток могли проскочить незамеченными. Поэтому, напишите мне лично (yakovfain@gmail.com), если что заметите. А теперь, усаживайтесь поудобнее, попкорн слева, ноут справа и, как говорят у нас в Америке, инджойте шоу под названием Учимся Программировать на языке Java!

Предисловие

Однажды, мой сын Дэйв-пароход появился у меня в офисе, держа мой учебник по Java для взрослых. Он попросил меня научить его программированию, чтобы создавать компьютерные игры. На тот момент я уже написал пару книг по языку Java и провел обучение программированию на компьютерах в нескольких группах, но это были взрослые! В результатах поиска на Amazon (крупнейший американский интернет-магазин) не было ничего, кроме книжек «для чайников», но Дэйв не был «чайником»!

После того, как я провел несколько часов поиска в Google, мне удалось найти, либо несколько не самых удачных попыток создать курсы Java для детей, либо несколько книг, написанных в стиле популярной в Америке детской серии Reader-Rabbit. Угадайте, что я решил сделать? Я решил написать книгу по программированию для детей. С целью понимания детского образа мыслей, я попросил Дэйва стать моим первым учеником-ребенком.

Так появилась эта книга, которая подойдет следующим группам людей:

- детям в возрасте от 11 до 18 лет;
- школьным преподавателям информатики;
- родителям, желающим обучить программированию своих детей;
- абсолютным новичкам в программировании (возраст значения не имеет).

Несмотря на то, что при объяснении программирования я использую простой язык, обещаю уважительное отношение к моим читателям. Я не планирую писать, что-либо похожее на «Дорогие друзья! Вы собираетесь начать новое и удивительное путешествие…». Да, конечно! Просто возьмемся за дело.

Первые главы книги завершатся небольшой программой-игрой, которая сопровождается подробными инструкциями о том, как сделать ее рабочей. Также мы создадим калькулятор, который выглядит и работает аналогично калькулятору вашего компьютера. Во второй части книги мы вместе создадим программы для игры в крестики-нолики и пинг-понг.

Вам потребуется привыкнуть к языку профессиональных программистов. Все важные слова будут напечатаны *вот таким шрифтом*.

Элементы языка Java и программ также будут выделены, например, `String`.

Эта книга не охватывает все элементы языка Java. В противном случае, это сделало бы ее слишком толстой и скучной. Однако в конце каждой главы помещен раздел с материалами для дополнительного чтения, который содержит ссылки на англоязычные веб-сайты с более подробными сведениями о рассматриваемой теме.

Кроме того, в конце каждой главы вы найдете задания для самостоятельного выполнения. Каждый читатель должен выполнить задания, которые содержатся в разделе *Практические упражнения*. Если эти задания покажутся вам слишком легкими, то попробуйте выполнить задания из раздела *Практические упражнения повышенной сложности*. В самом деле, если вы решили читать эту книгу, то вы наверняка способный человек и должны попытаться выполнить все задания.

Чтобы получить максимум из этой книги, прочитайте ее от начала до конца. Не следует двигаться дальше, пока вы не поймете содержание текущей главы. Подростки, родители, дедушки и бабушки должны справиться с этой книгой, не прибегая к посторонней помощи, однако маленькие дети должны читать эту книгу вместе со взрослыми.

Благодарности

Благодарю всех архитекторов и разработчиков, безвозмездно работающих над программой Eclipse, которая является одной из лучших из доступных сред интегрированной разработки программ.

Особая благодарность водителям междугородних автобусов компании New Jersey Transit за плавное вождение — половина этой книги была написана по пути на работу на автобусе № 139.

Благодарю жену Наташу за успешное управление бизнесом под названием семья.

Особая благодарность Юрию Гончарову, эксперту в области программирования на Java из Торонто, Канада. Он выполнил редактирование книги, проверил каждый пример кода и предоставил ценный отзыв, который позволил улучшить эту книгу.

Содержание

ПРЕДИСЛОВИЕ К РУССКОМУ ИЗДАНИЮ III

ПРЕДИСЛОВИЕ V

БЛАГОДАРНОСТИ VII

СОДЕРЖАНИЕ VIII

ГЛАВА 1. ПЕРВАЯ ПРОГРАММА 13

Установка Среды Java 14

Три основных шага в программировании 18
Шаг 1 – ввод текста программы 18
Шаг 2 – компиляция программы 20
Шаг 3 – запуск программы 21

Материалы для дополнительного чтения 22

ГЛАВА 2. ПЕРЕХОД К ECLIPSE IDE 23

Установка Eclipse IDE 23

Приступаем к работе с Eclipse 28

Создание программ в Eclipse IDE 32

Запуск HelloWorld в Eclipse 35

Как работает программа HelloWorld 36

Материалы для дополнительного чтения 38

Практические упражнения 38

Практические упражнения для умников и умниц 39

ГЛАВА 3. ДОМАШНЕЕ ЖИВОТНОЕ И РЫБА НА ЯЗЫКЕ JAVA 40

Классы и объекты 40

Типы Данных 42

Создаём Домашнее Животное 46

Наследование – Рыбка Тоже Домашнее Животное 52

Переопределение методов 56

Дополнительное чтение 57

Практические упражнения ... 57

Практические упражнения для умников и умниц ... 58

ГЛАВА 4. ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА JAVA ... 59

Комментарии в программе ... 59

Принятие решений с помощью оператора if ... 60

Логические операторы ... 62

Во втором случае *логическое не* применяется к результату вычисления выражения в скобках. ... 63

Условный оператор ... 63

Использование else if ... 64

Оператор `switch` и принятие решений ... 65

Как долго живут переменные? ... 66

Специальные методы: конструкторы ... 67

Ключевое слово `this` ... 68

Массивы ... 69

Повторение действий с помощью циклов ... 71

Материалы для дополнительного чтения ... 74

Практические упражнения ... 74

Практические упражнения для умников и умниц ... 75

ГЛАВА 5. ДЕЛАЕМ ГРАФИЧЕСКИЙ КАЛЬКУЛЯТОР ... 76

AWT и Swing ... 76

Пакеты и ключевое слово import ... 77

Основные элементы Swing ... 78

Схемы Размещения ... 81

FlowLayout - построчное расположение ... 82

GridLayout - табличное расположение ... 82

BorderLayout - размещение по областям ... 84

Комбинирование схем размещения ... 85

BoxLayout - расположение по горизонтали или вертикали ... 88

GridBag Layout - более гибкое табличное расположение ... 89

CardLayout – колода карт **90**
Можно ли создавать окна, не используя схемы? 91

Компоненты окна **91**

Материалы для дополнительного чтения **95**

Практические упражнения **95**

Практические упражнения для умников и умниц **96**

ГЛАВА 6. СОБЫТИЯ ОКНА 97

Интерфейсы **98**

Слушатель по имени ActionListener **100**

Регистрация компонентов с ActionListener **101**
Из-за кого событие-то? 102

Приведение типов - casting **102**

Как передавать данные между классами **105**

Доделываем калькулятор **106**

Некоторые другие слушатели событий **112**

Как использовать адаптеры **114**

Материалы для дополнительного чтения **115**

Практические упражнения **115**

Практические упражнения для умников и умниц **115**

ГЛАВА 7. АППЛЕТ КРЕСТИКИ-НОЛИКИ 116

Изучаем HTML за 15 минут **117**

Апплеты и AWT **120**

Как писать апплеты **121**

Пишем игру Крестики-нолики **123**

Стратегия **124**

Текст программы **124**

Материалы для дополнительного чтения **134**

Практические упражнения **135**

Практические упражнения для умников и умниц **136**

ГЛАВА 8. ИСКЛЮЧЕНИЯ – ОШИБКИ В ПРОГРАММАХ 137

Чтение трассировки стека ... 138

Генеалогическое дерево исключений ... 139

Блок try/catch ... 141

Ключевое слово throws ... 144

Ключевое слово finally ... 145

Ключевое слово throw ... 147

Создание своих исключений ... 148

Материалы для дополнительного чтения ... 150

Практические упражнения ... 151

Практические упражнения для умников и умниц ... 151

ГЛАВА 9. СОХРАНЕНИЕ СЧЁТА ИГРЫ ... 152

Байтовые потоки ... 153

Буферизированные потоки ... 155

Аргументы командной строки ... 157

Чтение текстовых файлов ... 160

Класс File ... 164

Материалы для дополнительного чтения ... 166

Практические упражнения ... 166

Практические упражнения для умников и умниц ... 167

ГЛАВА 10. РАЗНЫЕ ПОЛЕЗНЫЕ ШТУЧКИ ... 169

Работа с датами и временем ... 169

Перегрузка методов ... 171

Чтение данных с клавиатуры ... 174

Тебе пакет ... 176

Уровни доступа ... 179

`public class Car {` ... 181

Возвращаемся к массивам ... 183

Класс ArrayList ... 186

Материалы для дополнительного чтения ... 190

Практические упражнения ... 190

Практические упражнения для умников и умниц ... 191

ГЛАВА 11. ВОЗВРАЩАЕМСЯ К ГРАФИКЕ. ПИНГ-ПОНГ ... 192

Стратегия ... 192

Код ... 193

Основы многопоточности ... 201

Заканчиваем игру Пинг-Понг ... 206

Материалы для дополнительного чтения ... 216

Практические задания ... 216

Практические упражнения для умников и умниц ... 217

ПРИЛОЖЕНИЕ А. JAVA АРХИВЫ - JARS ... 218

Материалы для дополнительного чтения ... 219

ПРИЛОЖЕНИЕ Б. СОВЕТЫ ДЛЯ РАБОТЫ В ECLIPSE ... 220

Отладчик Eclipse ... 221

ПРИЛОЖЕНИЕ В. КАК ОПУБЛИКОВАТЬ ВЕБ-СТРАНИЦУ ... 225

Материалы для дополнительного чтения ... 229

Практические упражнения ... 229

ИНДЕКС ... 230

Глава 1. Первая программа

Люди говорят друг с другом, используя для этого различные языки. Точно также они пишут компьютерные программы, такие как игры, калькуляторы, текстовые редакторы, используя для этого различные языки программирования. Без программ ваш компьютер будет бесполезен, а его экран всегда будет черным. Компоненты компьютера называют ***аппаратным обеспечением***, а программы — ***программным обеспечением***. Самыми популярными компьютерными языками являются C# и Java. Чем язык Java отличается от множества других языков?

Во-первых, одна и та же программа Java может быть *запущена* (работать) без каких-либо изменений на различных компьютерах, например PC, Apple или других платформах. Фактически программы, написанные на Java даже не знают, на каком компьютере они выполняются, так как они выполняются внутри специальной программной оболочки, которая называется виртуальная машина JVM (Java Virtual Machine).

Если, например, программе Java требуется напечатать какие-то сообщения, она просит сделать это виртуальную машину JVM, которая знает, как нужно взаимодействовать с вашим принтером.

Второе, Java позволяет создавать программные элементы (*классы*), которые представляют объекты из реального мира. Например, можно создать класс Java с именем `Car` (автомобиль) и задать свойства этого класса, такие как двери, колеса, подобно тем, какие есть у настоящих автомобилей. После этого, основываясь на этом классе, можно создать другой класс, например, `Ford`, который будет иметь все свойства

класса `Car` плюс те свойства, которые есть только у автомобилей марки Ford.

Третье, язык Java обладает огромным количеством дополнительных и бесплатных примочек (программных библиотек), написанных тысячами программистов со всего мира, и это делает среду программирования Java намного более мощной по сравнению с другими языками.

Четвертое, язык Java поставляется бесплатно! Вы можете найти в Интернете все необходимое для создания программ на Java, не заплатив ни копейки за это!

Установка Среды Java

Чтобы начать программирование на Java, необходимо загрузить специальное программное обеспечение (ПО) с веб-сайта компании Oracle. Язык Java был создан компанией Sun Microsystems. В 2009 году компания Oracle приобрела компанию Sun Microsystems. Полное наименование этого ПО — Java Development Kit (JDK). На момент написания книги последнюю версию этого ПО (JDK 7) можно загрузить на этом веб-сайте:

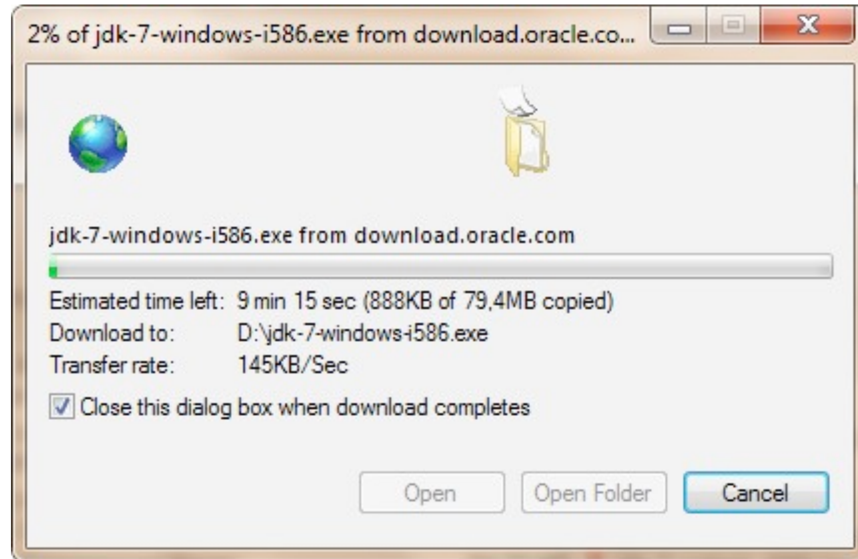
http://www.oracle.com/technetwork/java/javase/downloads/index.html

Выберите в таблице выпуск Java SE 7 или более новый. Нажмите кнопку **Download** под заголовком JDK в строке таблице с этим выпуском. Примите условия лицензионного соглашения. Выберите пакет, соответствующий операционной системе, которая установлена на вашем компьютере. Если на вашем компьютере установлена операционная система Windows, то это будет Windows x86 (32-разрядная версия) или Windows x64 (64-разрядная версия).

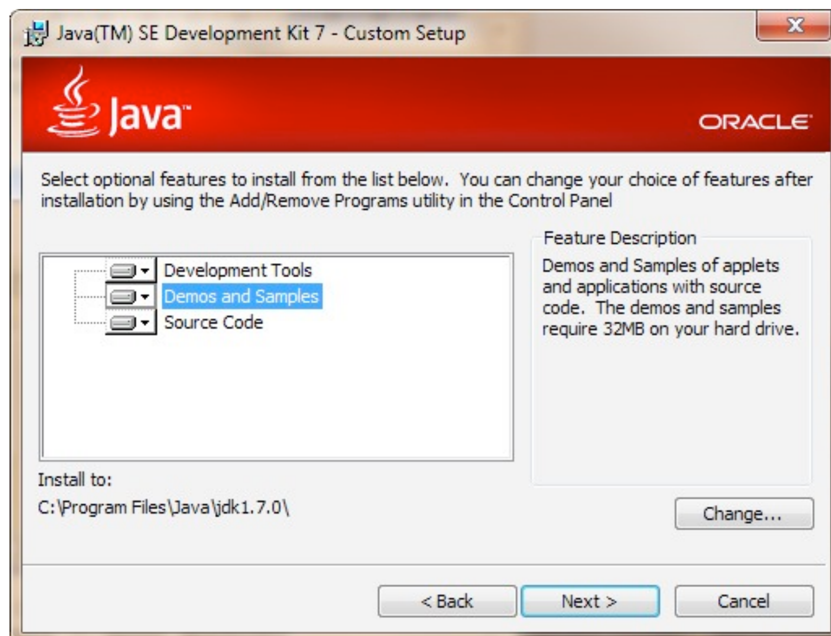
Если на вашем компьютере установлена система Linux или Solaris, выберите соответствующий пакет. До недавнего времени компьютеры Apple поставлялись с уже предустановленным пакетом JDK. Если у вас компьютер Apple, запустите программу Терминал, введите там слово Java и нажмите клавишу Enter. Если вы увидите сообщение Command not found, значит на вашем Маке Java не установлена, и ее нужно загрузить отсюда:

http://support.apple.com/downloads/#Java

Щелкните по ссылке с выбранным пакетом. Начнется загрузка файла. На компьютерах с операционной системой Windows это может выглядеть так:



После окончания загрузки запустите процесс установки — дважды щелкните кнопкой мыши по загруженному файлу.

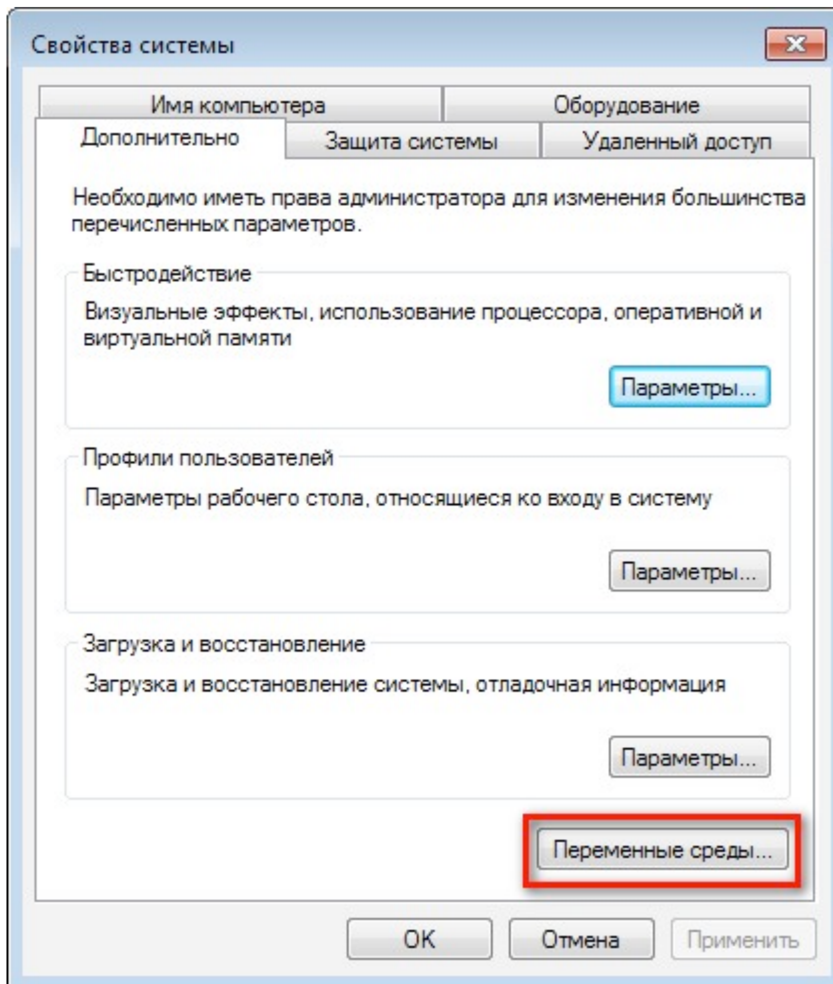


Произойдет запуск мастера установки. Например, на компьютере под управлением Windows будет создан следующий каталог:

`C:\Program Files\Java\jdk1.7.0`, где `C:` это имя жесткого диска.

Если недостаточно места на диске C:, то выберите другой диск. Следуйте указаниям мастера установки — просто нажимайте кнопки *Next*, *Install* и *Finish* в появляющихся на экране окнах. В течение нескольких минут установка Java на компьютер будет завершена.

На следующем этапе установки необходимо задать две *системные переменные*. Например, в Windows нажмите кнопку *Пуск*, далее щелкните правой кнопкой мыши по пункту *Мой компьютер* и выберите пункт *Свойства*.

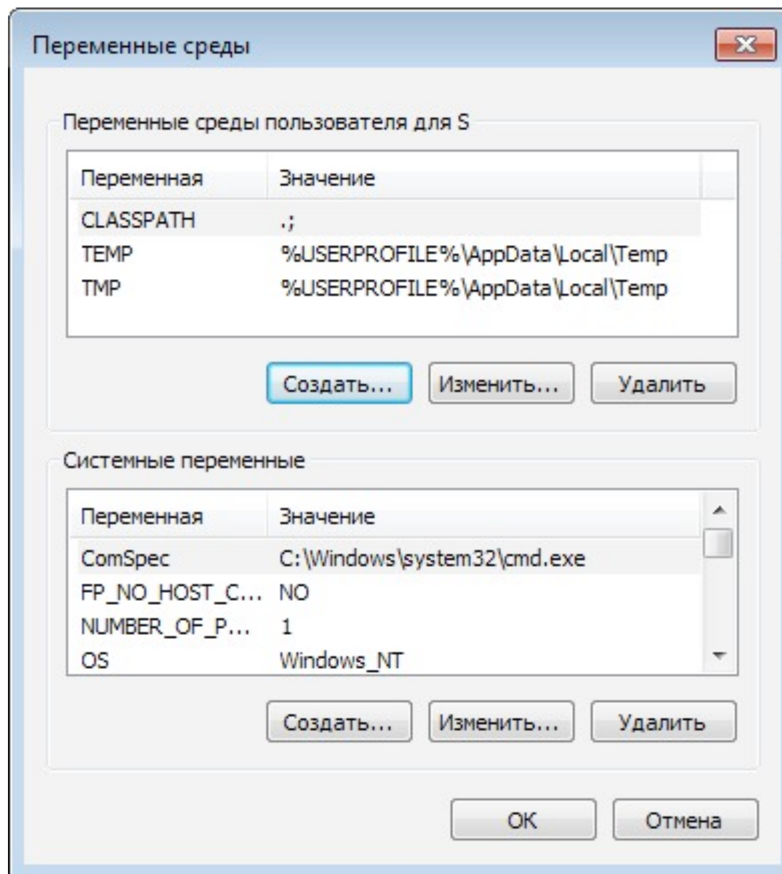


Далее перейдите по ссылке *Дополнительные параметры системы*, которая расположена в левой верхней части панели

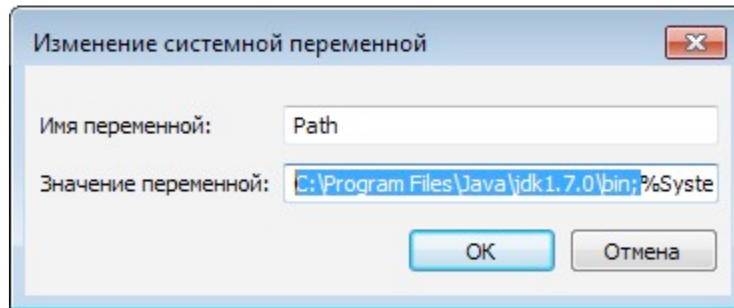
слева. Перейдете на вкладку *Дополнительно* и нажмите на кнопку *Переменные среды*...

На следующей странице приведено изображение окна с настройками для Windows 7.

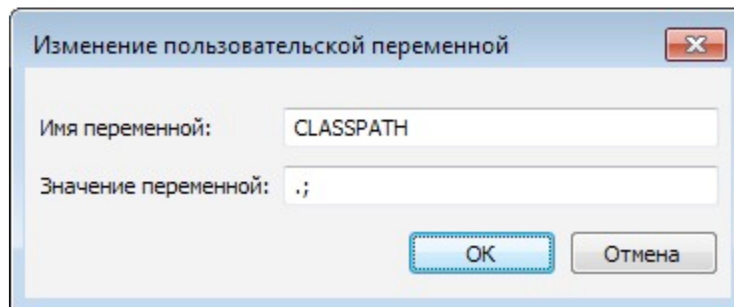
Так выглядит окно, которое содержит все системные переменные, которые есть в системе:



Нажмите нижнюю кнопку *Создать* и объявите переменную `Path`, которая поможет Windows (или Unix) найти пакет JDK на вашем компьютере. Тщательно проверьте путь к каталогу, в который была произведена установка Java. Если переменная `Path` уже существует, просто добавьте путь к каталогу Java и точку с запятой в самое начало поля *Значение переменной*:



Также объявите переменную CLASSPATH. В качестве ее значения укажите точку, за которой следует точка с запятой (.;). Данная системная переменная поможет Java найти ваши программы. Точка означает, что Java начнет поиск ваших программ в текущем каталоге. Точка с запятой всего лишь играет роль разделителя, за которыми могут идти другие каталоги.



Ну вот, установка пакета JDK завершена!

Три основных шага в программировании

Чтобы создать работающую программу на Java необходимо пройти через три следующих шага.

- ✓ *Написать* программу на Java и сохранить ее на диск.
- ✓ *Выполнить компиляцию* программы, чтобы перевести ее с языка Java в специальный байт-код, который понимает виртуальная машина JVM.
- ✓ *Запустить* программу.

Шаг 1 – ввод текста программы

Для того чтобы написать программу на Java, вы можете использовать любой текстовый редактор, например, «Блокнот».

Первое, нужно будет напечатать программу и сохранить ее в текстовый файл, имя которого будет оканчиваться расширением .java. Например, если нужно написать программу с названием `HelloWorld`, напечатайте ее текст (этот текст мы называем *исходный код*) в программе «Блокнот» и сохраните его в файл с именем `HelloWorld.java`. Внимание, не используйте пробелы в именах Java-файлов.

Ниже приведена программа, которая выводит на экран слова *Hello World*:

```
public class HelloWorld {

      public static void main(String[] args) {

            System.out.println("Hello World");
      }
}
```

Я объясню работу этой программы немного позднее в этой главе, но в данный момент просто поверьте мне — эта программа будет печатать слова *Hello World* на третьем шаге.

Шаг 2 – компиляция программы

Теперь необходимо откомпилировать программу. Вы будете использовать компилятор `javac`, который входит в состав пакета JDK.

Допустим, вы сохранили программу в каталоге с именем `C:\practice`. Нажмите кнопку *Пуск*, введите в строке поиска команду `cmd` и нажмите клавишу ВВОД. В результате откроется черное командное окно.

Чтобы убедиться в том, что вы правильно установили значения системных переменных `PATH` и `CLASSPATH`, введите команду `set`. Найдите и проверьте эти значения в результатах вывода этой команды.

Измените текущий каталог на `C:\practice` и откомпилируйте программу:

```
cd \practice

javac HelloWorld.java
```

Вам необязательно называть каталог именем *practice* – назовите его так, как вам нравится.

Программа `javac` это компилятор языка Java. Вы не увидите какого-либо подтверждения, что ваша программа `HelloWorld` была успешно откомпилирована. Это как раз тот случай, когда отсутствие новостей это хорошая новость!

Введите команду `dir`. Результатом ее выполнения является вывод на экран всех файлов, которые существуют в каталоге. Вы должны увидеть, что появился новый файл с именем `HelloWorld.class`. Это является доказательством того, что ваша программа была успешно

откомпилирована. Ваш исходный файл HelloWorld.java также будет там. Вы можете изменить его позже так, чтобы на экран выводились слова *Привет, мама* или что-нибудь еще.

Если в программе есть синтаксические ошибки, скажем, вы забыли напечатать последнюю фигурную скобку, компилятор Java выведет сообщение об ошибке. В этом случае вам необходимо исправить ошибку и откомпилировать программу еще раз. Если есть несколько ошибок, то может потребоваться повторение этих действий несколько раз, пока не будет создан файл HelloWorld.class.

Шаг 3 – запуск программы

Ну а теперь, запустим программу. В том же командном окне введите следующую команду:

```
java HelloWorld
```

Обратили ли вы внимание, что в этот раз вы использовали программу java, а не javac? Данная программа входит в среду выполнения *JRE* (Java Run-time Environment) и она запускает *JVM,* в которую загружает вашу программу HelloWorld.

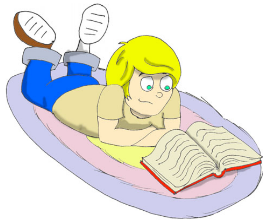
Помните, что язык Java чувствителен к регистру, это значит, что если вы назвали программу HelloWorld с заглавной буквой H и с заглавной буквой W, не пытайтесь запустить программу helloworld или helloWorld – JVM будет жаловаться, что, мол, не нахожу файл.

Теперь можно поразвлечься — попытайтесь догадаться, как можно изменить эту программу. В следующей главе я буду объяснять, как работает эта программа. Однако попробуйте предположить, как можно изменить ее, чтобы сказать «Привет!» своему домашнему животному, другу или напечатать свой адрес.

Пройдите через все три шага, чтобы увидеть, работает ли моя программа после ваших изменений ☺.

В следующей главе я покажу, как печатать, компилировать и запускать ваши программы в более привлекательном месте, нежели текстовый редактор и черное командное окно.

Материалы для дополнительного чтения



Создание первого приложения:
http://download.oracle.com/javase/tutorial/getStarted/cupojava/win32.html

Инструкции по установке Java для Windows:

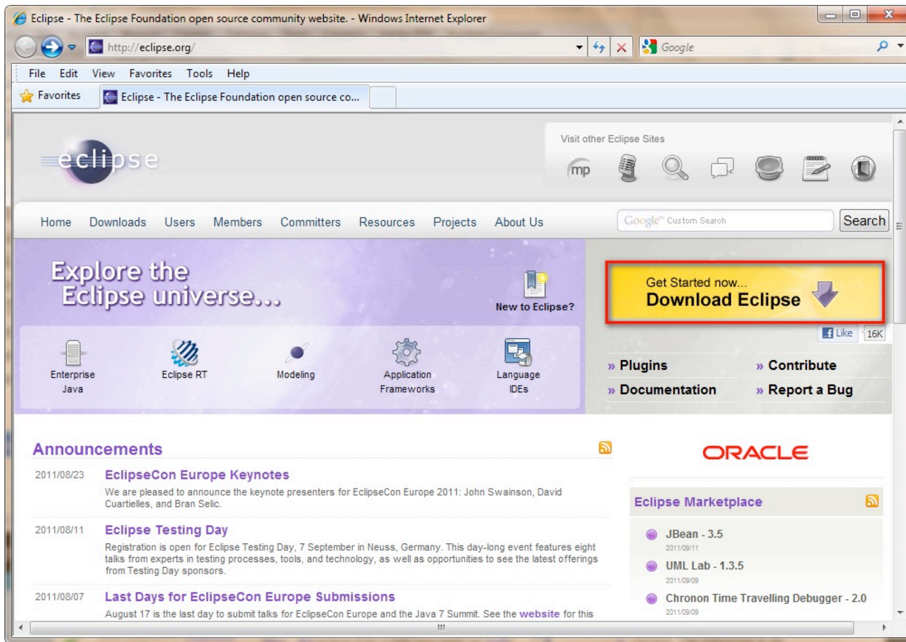
http://download.oracle.com/javase/7/docs/webnotes/install/windows/jdk-installation-windows.html

Глава 2. Переход к Eclipse IDE

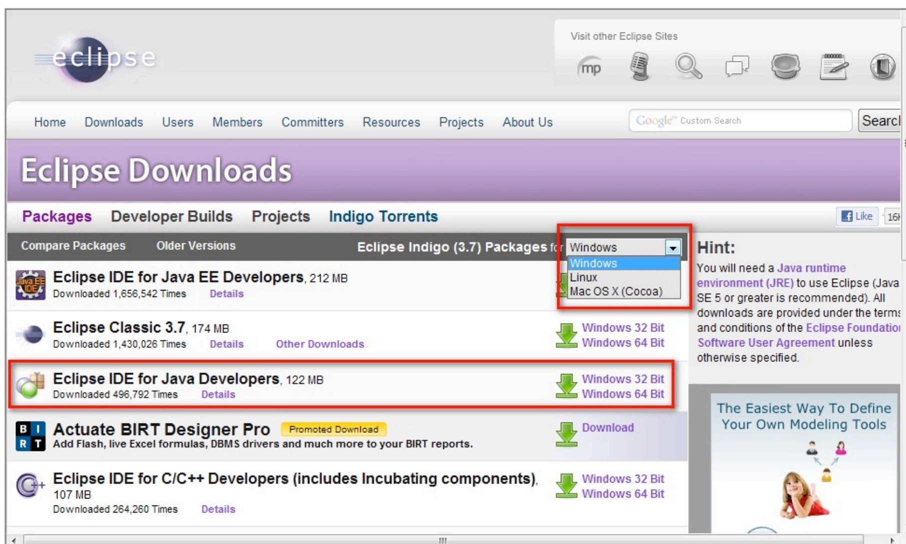
Программисты обычно работают с инструментом, который называется *интегрированная среда разработки IDE* (Integrated Development Environment). Программы можно писать, компилировать и запускать прямо в этой среде. В IDE также есть такая штука, как *Справка*, содержащая все элементы языка, которая упрощает поиск и исправление ошибок в программах. Некоторые программы IDE имеют высокую стоимость, однако есть превосходная бесплатная IDE под названием Eclipse. Ее можно загрузить с веб-сайта www.eclipse.org. В этой главе я помогу вам загрузить и установить Eclipse IDE на ваш компьютер и создать в этой среде проект с названием `Hello World`. После этого все наши программы мы будем создавать в этой среде. Устраивайтесь поудобнее в Eclipse — это превосходный инструмент, который используется множеством профессиональных программистов на Java.

Установка Eclipse IDE

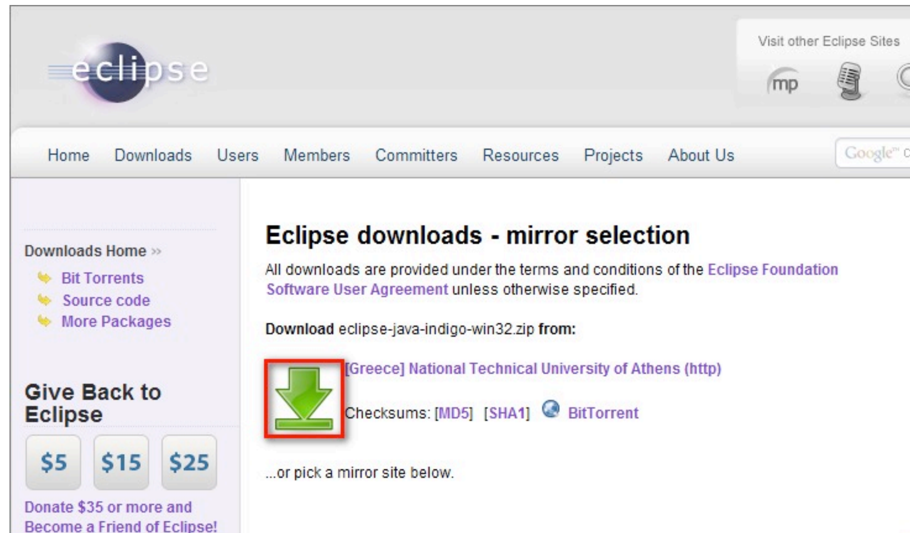
Откройте веб-сайт www.eclipse.org и нажмите в верхней правой части страницы кнопку *Download Eclipse* («Загрузка Eclipse»).



После перехода на страницу *Eclipse Downloads*, выберите в выпадающем списке версию вашей операционной системы. Далее выберите выпуск *Eclipse IDE for Java Developers*. Чтобы загрузить его, перейдите по ссылке в правой части, которая соответствует версии вашей системы. Если вы не уверены какая версия Windows у вас установлена, выбирайте 32-х разрядную версию.



На открывшейся странице, для того чтобы начать загрузку с предложенного системой сервера, щелкните по кнопке со стрелкой.



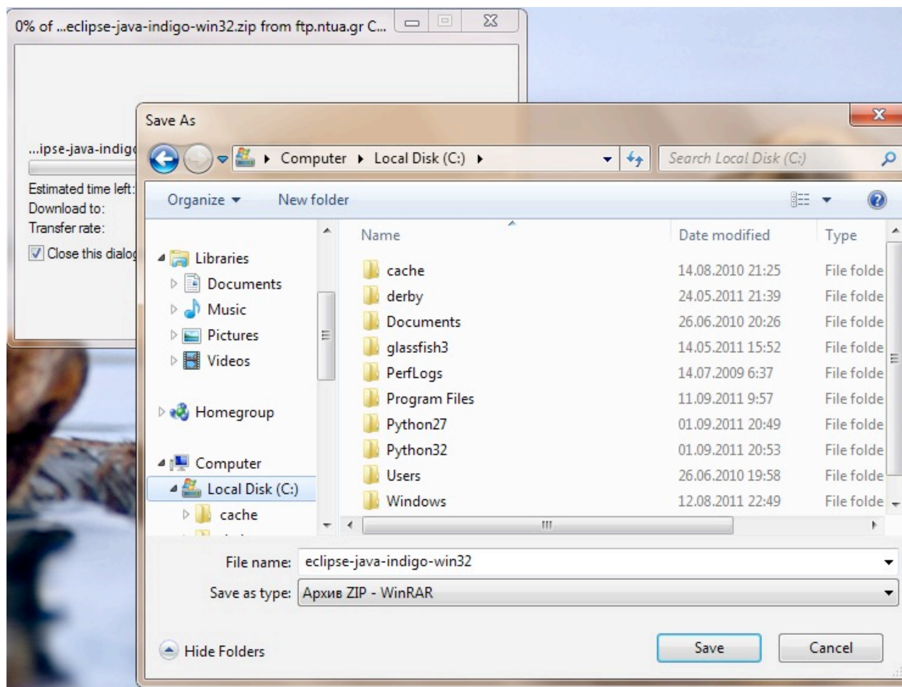
Либо, в нижней части этой страницы, вы можете выбрать альтернативный источник загрузки из ближайшей к вам страны или города. Это рекомендуется сделать в случае, если загрузка из предложенного источника невозможна или выполняется слишком медленно (так же зависит от скорости вашего подключения).

Please choose a mirror close to you

Europe

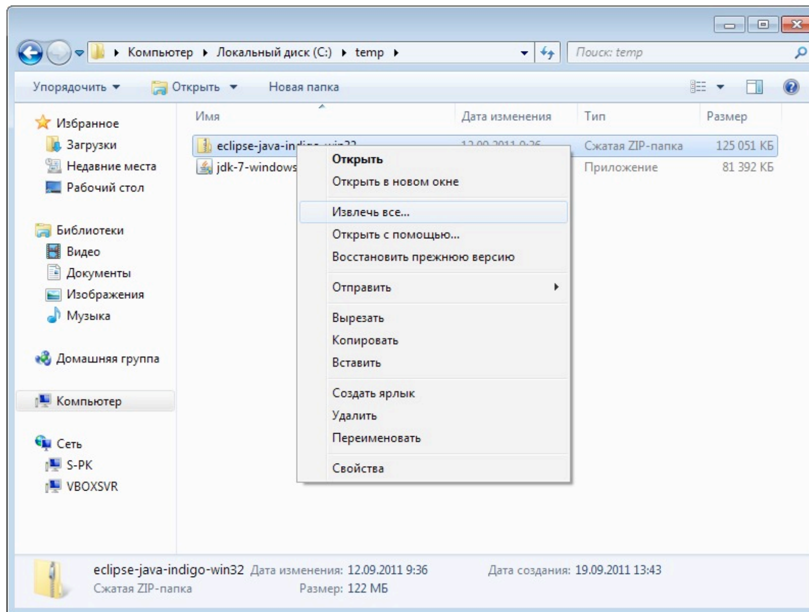
- [Greece] National Technical University of Athens (ftp)
- [Italy] GARR/CILEA (http)
- [Sweden] ING-net Umea University (http)
- [Italy] GARR (http)
- [France] Ialto (http)
- [Portugal] Universidade do Porto - Faculdade de Engenharia (http)
- [Czech Republic] UPC Ceska republika, a.s. (http)
- [Romania] Romanian Education Network (http)
- [Romania] National Agency ARNIEC - RoEduNet (http)
- [Germany] University of Applied Sciences Esslingen (http)
- [Switzerland] SWITCHmirror (http)

Сохраните загружаемый файл на диск в любой папке.



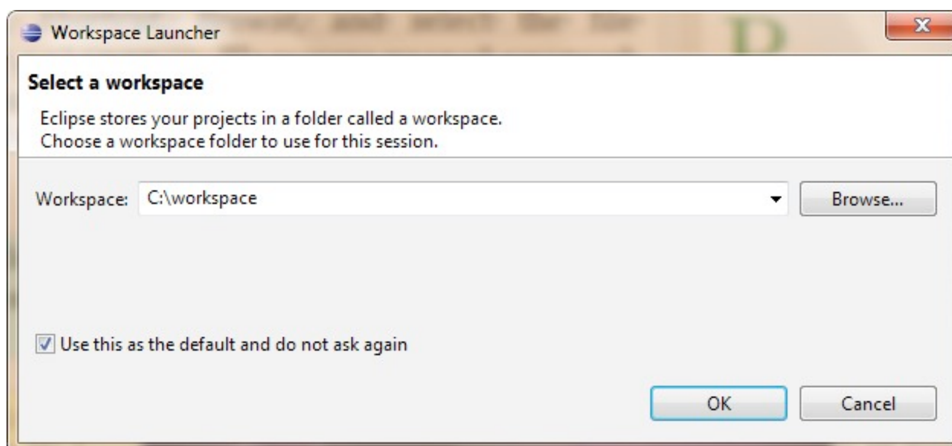
Теперь нужно просто распаковать этот файл на диск `C:`. Для этого щелкните правой кнопкой мыши по архиву. Выберите пункт *Извлечь все*…

Файлы, которые имеют расширение ZIP — это архивы. Внутри себя они могут содержать другие файлы. *Распаковать* файл означает *извлечь* содержимое архива на диск.



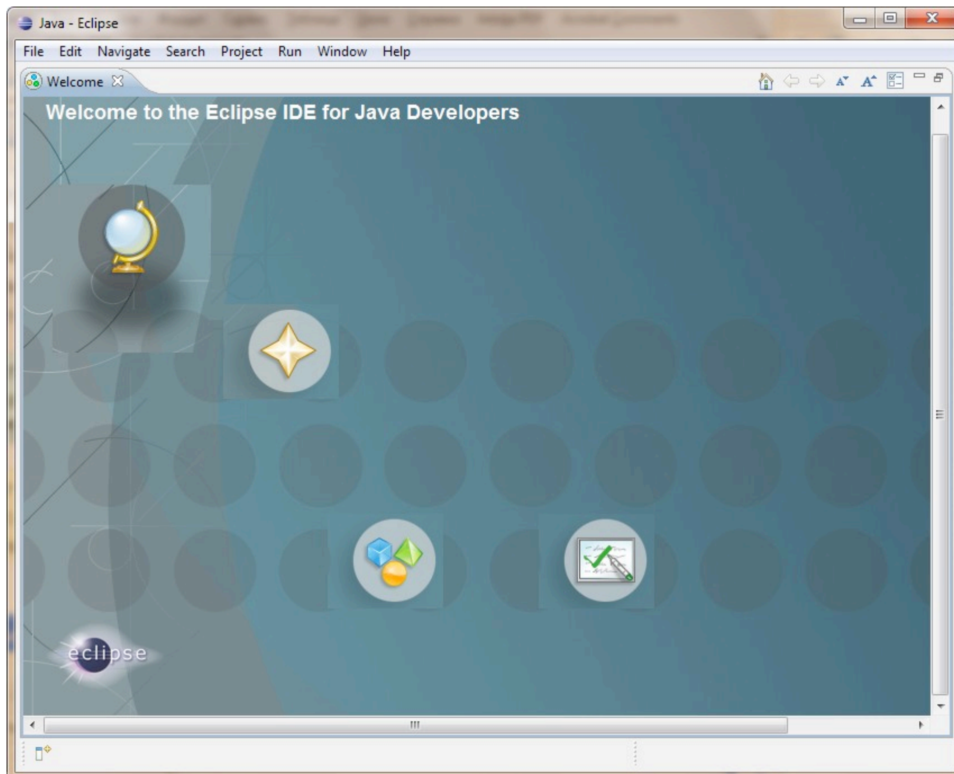
Установка Eclipse выполнена! Для удобства создайте *ярлык* для Eclipse. Щелкните правой кнопкой по рабочему столу. В появившемся меню последовательно выберите пункты *Создать* > *Ярлык* > *Обзор* и выберите в папке `C:\eclipse` файл `eclipse.exe`. Чтобы запустить программу, дважды щелкните по синему значку *Eclipse*.

При первом запуске Eclipse будет открыто окно с просьбой указать расположение рабочей области (workspace) — каталога, в котором будут храниться файлы ваших программ.



В поле *Workspace* («Рабочая область») укажите удобное для вас место на диске, например `C:\workspace`. Если вы не хотите, чтобы Eclipse

каждый раз при запуске просила вас указать рабочую область, установите флажок *Use this as the default and do not ask again* («Использовать это значение по умолчанию в дальнейшем»). Нажмите кнопку *OK*. После запуска Eclipse открывается начальная страница *Welcome*. Внешний вид этого окна незначительно меняется с каждой сборкой Eclipse.



Теперь перейдите в так называемую среду *Workbench* («Рабочая среда»), которая является рабочим местом для ваших проектов на Java. Для этого просто закройте начальную страницу, нажав на крестик после слова Welcome.

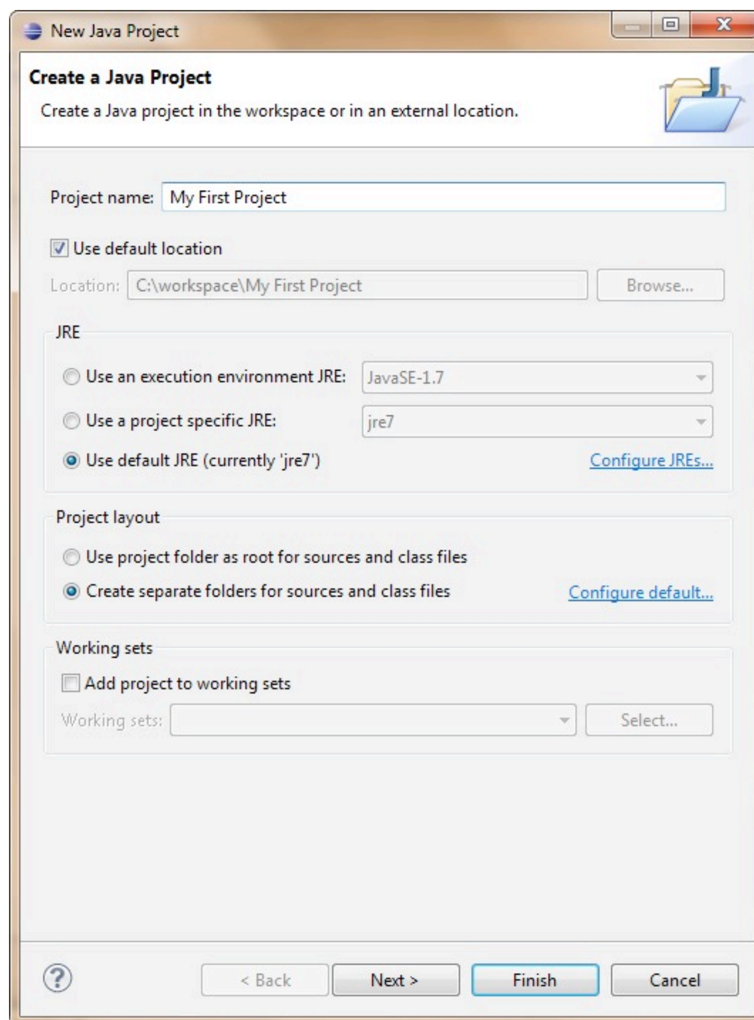
Приступаем к работе с Eclipse

В данном разделе я покажу вам, как можно быстро создать и запустить программы на Java в Eclipse. Вы также можете найти неплохой учебник прямо в Eclipse. Для этого последовательно перейдите меню *Help* («Справка»), *Help Contents* («Оглавление справки»), далее перейдите в

раздел *Java Development User Guide* («Руководство пользователя по разработке на Java»).

Чтобы начать работать над программой, необходимо создать новый проект. Простой проект, как наш *My First Project*, будет содержать всего один файл — `HelloWorld.java`. Позже мы создадим более сложные проекты, которые будут содержать несколько файлов.

Чтобы создать новый проект в Eclipse, выберите следующие пункты меню *File* («Файл»), *New* («Создать»), *Java Project* («Проект Java»). В результате откроется окно *New Java Project* («Создание проекта Java»). Теперь необходимо ввести имя нового проекта, например, *My First Project*.

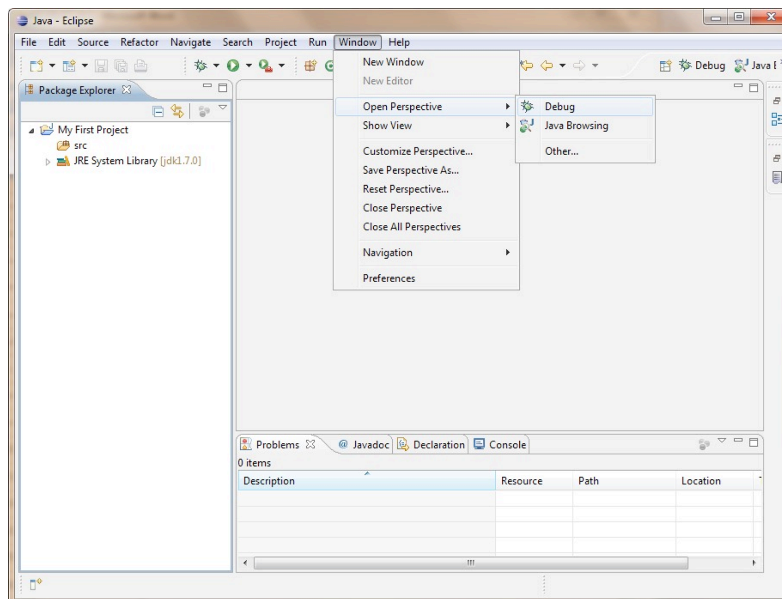


Обратите внимание на поле *Location* («Расположение»). Оно указывает вам расположение на диске, в котором будет размещен этот проект. По умолчанию будет предложено сохранить проект в каталоге с именем

данного проекта. Этот каталог будет находиться в рабочей области, которая была вами задана при запуске Eclipse. Позже вы создадите отдельные проекты для калькулятора, игры в крестики-нолики и других программ. К концу этой книги в рабочей области будет находиться несколько проектов.

Рабочая область Eclipse содержит несколько панелей, определенную компоновку которых называют проекцией (perspective).

Чтобы открыть проекцию, нужно выбрать в меню пункты *Window* («Окно»), *Open Perspective* («Открыть проекцию») и далее выбрать нужную проекцию.

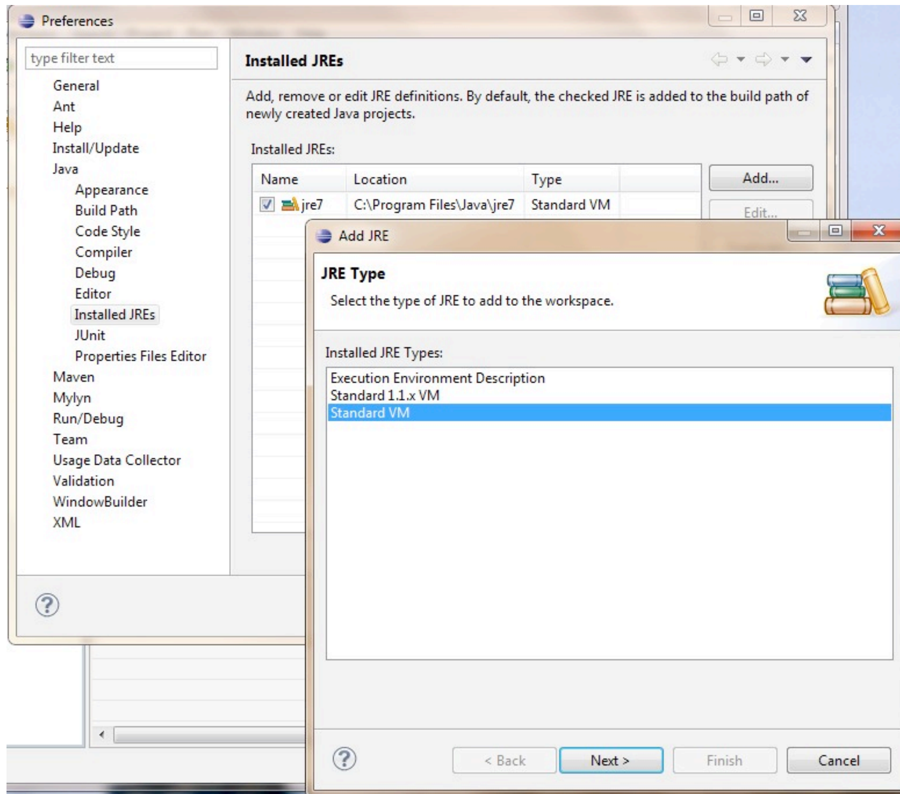


Каждая проекция содержит набор панелей, которые используются при решении различных задач. Например, при разработке программ на Java мы будем использовать проекцию Java, а при отладке программ — проекцию Debug («Отладка»).

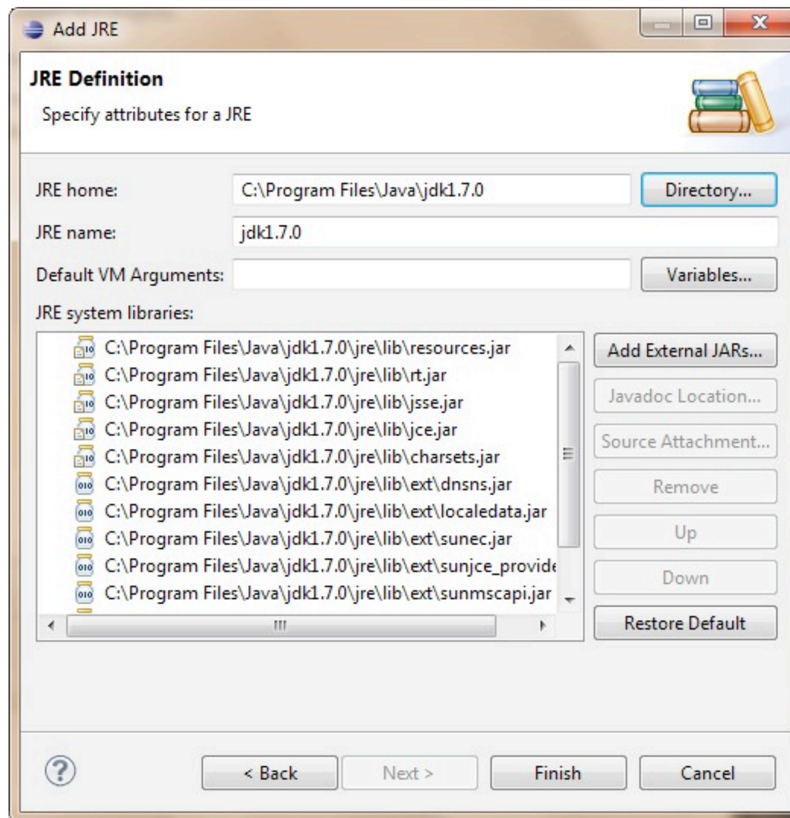
Если щелкнуть по значку треугольника рядом с именем проекта *My First Project*, при раскрытии будет показан элемент `JRE System Library (jdk 1.7.0)`. Этот элемент является частью проекта. Если по какой-либо причине у вас нет этого элемента, добавьте его. Для этого выберите пункты меню *Windows* («Окно»), *Preferences* («Параметры»).

В открывшемся окне перейдите в раздел *Java, Editor* («Редактор»), *Installed JREs* («Установленные JRE»). Нажмите кнопку *Add* («Добавить»).

В открывшемся окне выберите элемент *Standard VM* («Стандартная виртуальная машина»). Нажмите кнопку *Next* («Далее»).



В открывшемся окне нажмите кнопку *Directory* и укажите директорию, в которой вы установили Java, например, `C:\Program Files\Java\jdk1.7.0\`.



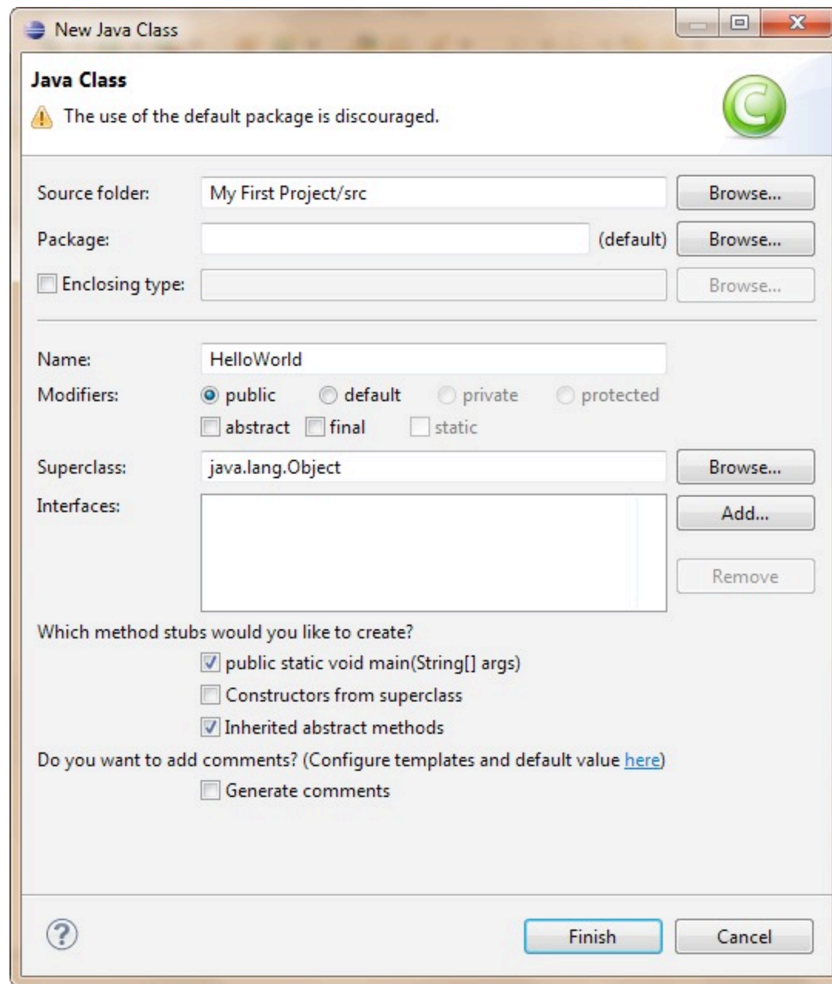
Нажмите кнопку *Finish*. В результате должна добавиться новая JRE.

Создание программ в Eclipse IDE

Теперь повторим создание программы `HelloWorld` из главы 1 в Eclipse. Программы Java являются *классами*, которые представляют объекты из реальной жизни. Дополнительные сведения о классах вы узнаете из следующей главы.

Чтобы создать класс в Eclipse, выберите в меню пункты *File*, *New*, *Class*. В открывшемся окне введите `HelloWorld` в поле *Name* («Имя). Также в разделе *Which methods stubs you would like to create* («Какие заготовки для методов необходимо создать?») установите флажок для метода main():

```
public static void main(String[] args)
```



Нажмите кнопку *Finish* и Eclipse создаст класс `HelloWorld`, который будет содержать пустой *метод* `main()`. Слово *метод* в данном случае означает *действие*. Для того, чтобы класс Java можно было запустить как программу, необходимо, чтобы этот класс имел метод с именем `main()`.

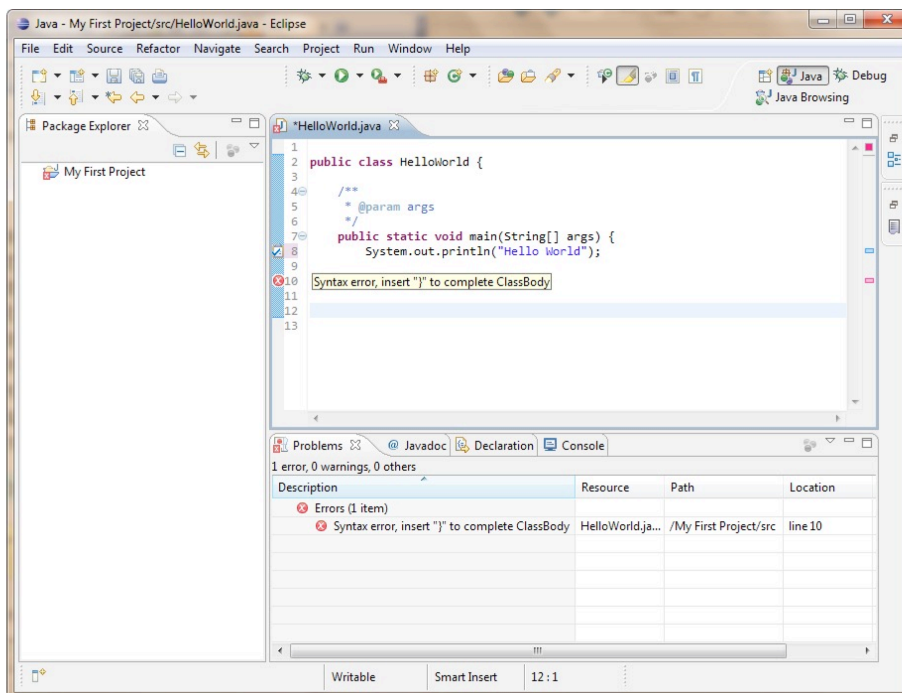
```
public class HelloWorld {

      public static void main(String[] args) {

      }
}
```

Чтобы завершить нашу программу, поместите курсор после открывающей фигурной скобки в строке с именем `main`, нажмите клавишу *Enter* и на новой строке напечатайте следующий текст:

```
System.out.println("Hello World");
```

Чтобы сохранить программу на диск и одновременно откомпилировать ее, нажмите комбинацию клавиш *Ctrl-S*. Если вы не сделали синтаксических ошибок, то никаких сообщений об успешной компиляции не будет, но программа откомпилирована. А давайте специально сделаем ошибку, чтобы посмотреть, что произойдет.

Сотрите последнюю фигурную скобку и снова нажмите комбинацию клавиш *Ctrl-S*. На вкладке *Problems* Eclipse выведет сообщение об ошибке *Syntax error, insert "}" to complete ClassBody* («Синтаксическая ошибка; вставьте "}" для завершения *ClassBody*»). Также рядом со строкой, которая содержит ошибку, появится красная отметка. При наведении на нее указателя мыши, всплывет сообщение с аналогичным описанием ошибки.

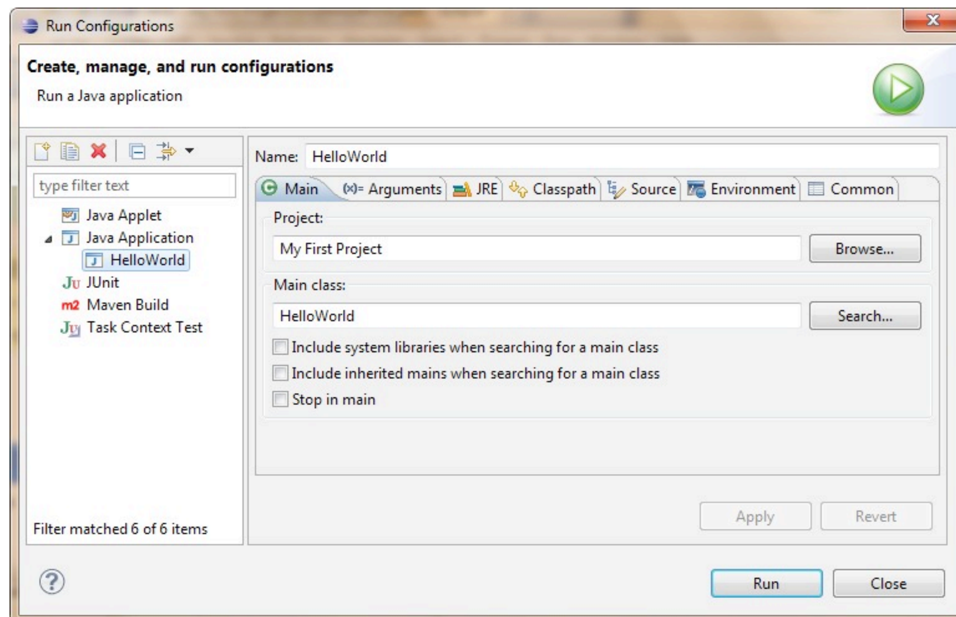


С увеличением размера ваших проектов, они будут содержать несколько файлов, и компилятор может генерировать большее количество ошибок. Вы научитесь быстро находить и исправлять ошибки синтаксиса. Для этого дважды щелкните кнопкой мыши по сообщению об ошибке в панели *Problems*. Верните фигурную скобку на место и снова нажмите *Ctrl-S* — вуаля, сообщение об ошибке исчезло!

Запуск HelloWorld в Eclipse

Наш простой проект состоит из одного класса, что не типично. Вот почему перед запуском проекта в первый раз, необходимо указать Eclipse класс, который является главным в этом проекте.

Выберите в меню *Run* («Запуск»), далее *Run Configurations* («Выполнить…»). В открывшемся окне проверьте, что в левой части выбран пункт *Java Application* («Приложение Java»). На вкладке *Main* в поле *Project* («Проект») введите имя проекта, в поле *Main class* («Главный класс») введите имя главного класса.



Теперь, чтобы запустить программу, нажмите кнопку *Run* («Запуск»). В результате в панели *Console* («Консоль») будут напечатаны слова *Hello World,* точно так же, как это было сделано в главе 1.

Теперь можно запускать проект на выполнение с помощью выбора в меню пунктов *Run* («Запуск»), *Run* («Запуск») или с помощью нажатия клавиш *Ctrl-F11*.

Как работает программа HelloWorld

Давайте начнем разбираться - что же фактически происходит в программе `HelloWorld`?

Класс `HelloWorld` содержит только один метод `main()`, который является точкой входа *приложения* на Java. То, что `main` — это метод, говорят круглые скобки после слова `main`. Методы могут *вызывать* (использовать) другие методы, например, наш метод `main()`, чтобы напечатать на экране текст `Hello World`, вызывает метод `println()`.

Каждый метод начинается со *строки объявления*, которую называют ***сигнатурой метода***:

```
public static void main(String[] args)
```

Сигнатура метода говорит нам о следующем.

- Кто имеет доступ к методу — `public`. Ключевое слово `public` означает, что метод `main()` доступен для любого другого класса Java или самой JVM.
- Как вызывать метод — `static`. Ключевое слово `static` означает, что вам не нужно создавать *экземпляр* (копию) объекта `HelloWorld` в памяти, чтобы использовать этот метод. Более подробно об экземплярах класса мы поговорим в следующей главе.
- Возвращает ли метод какие-то данные? Ключевое слово `void` означает, что метод `main()` не возвращает данных в программу, которая его вызвала. В данном случае это программа Eclipse. Однако, если метод должен сделать какие-то расчеты, он может возвращать полученные результат вызывающему объекту.
- Именем метода является слово перед круглыми скобками - `main`.
- Список аргументов — некие данные, которые могут быть переданы методу — `String[] args`. В методе `main()` слова

> `String[] args` означают, что этот метод может получать *массив* объектов с типом `String`, то есть текстовые данные. Значения, которые передаются методу, называются *аргументами или параметрами*.

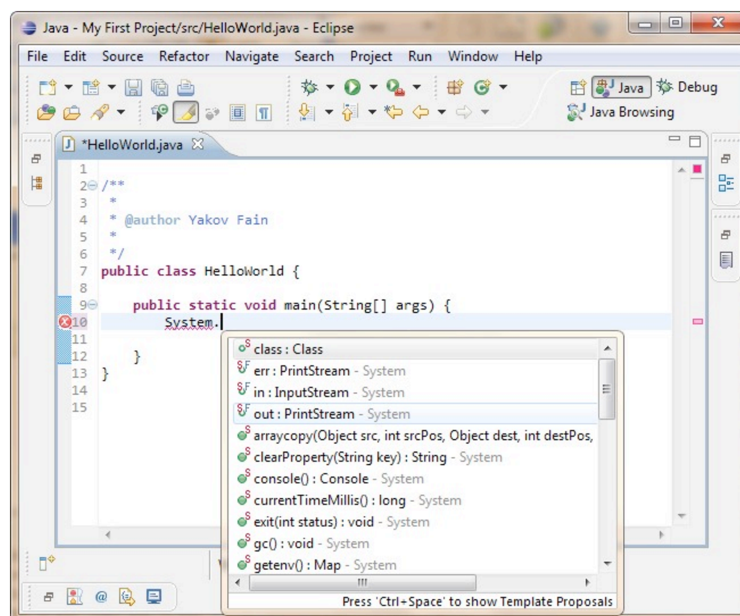
Как я уже говорил, программа может состоять из нескольких классов, но только один из них содержит метод `main()`. Класс Java обычно содержит несколько методов. Например, класс Game может содержать методы `startGame()`, `stopGame()`, `readScore()` и так далее.

Тело нашего метода `main()` содержит только одну строку:

```
System.out.println("Hello World");
```

Каждая команда или вызов метода должен заканчиваться точкой с запятой (;). Метод `println()` знает как выводить данные в системную *консоль* (командную консоль). После имени методов Java всегда идут круглые скобки. Если вы видите метод с пустыми круглыми скобками, это значит, что этот метод не имеет аргументов.

Слова `System.out` означают, что переменная `out` определена внутри класса `System`, который поставляется вместе с Java. Как же вам узнать, что в классе `System` есть что-то с именем `out`? Eclipse поможет вам с этим. После того, как вы напечатаете слово `System` и поставите точку, Eclipse покажет вам все, что есть в этом классе. В любой момент вы можете поместить курсор после точки и нажать комбинацию клавиш *Ctrl-Space*, чтобы вызвать окно справки, показанное ниже.



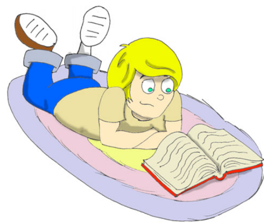
Слова `out.println()` говорят нам, что есть объект, который представлен *переменной* `out` и это «что-то под названием `out`» содержит метод с именем `println()`. Точка, которая находится между классом и именем метода означает, что этот метод был обьявлен внутри этого класса. Скажем, у вас есть класс `PingPongGame`, который содержит метод `saveScore()` — сохраняет счет в игре. Ниже приведен пример, как вы можете *вызвать* этот метод для Дейва, который выиграл три игры:

```
PingPongGame.saveScore("Дейв", 3);
```

Напоминаю, данные в круглых скобках называются *аргументы* или *параметры.* Эти параметры даются методу, чтобы он выполнил над ними определенную обработку, например, сохранил данные на диск. Метод `saveScore()` имеет два аргумента — строка текста «Дейв», и число 3.

Создавать программы в Eclipse намного приятнее, чем в обычном текстовом редакторе, правда? В Приложении Б есть полезные советы, которые позволят вам ускорить процесс программирования на Java в этом превосходном IDE.

Материалы для дополнительного чтения



Eclipse IDE Tutorial by Lars Vogel:

http://www.vogella.de/articles/Eclipse/article.html

Практические упражнения

Измените класс `HelloWorld`, чтобы напечатать свой адрес с помощью нескольких вызовов метода `println()`.

Практические упражнения для умников и умниц

Измените класс `HelloWorld`, чтобы напечатать слово *Hello* , так как показано ниже.

Глава 3. Домашнее Животное и Рыба на Языке Java

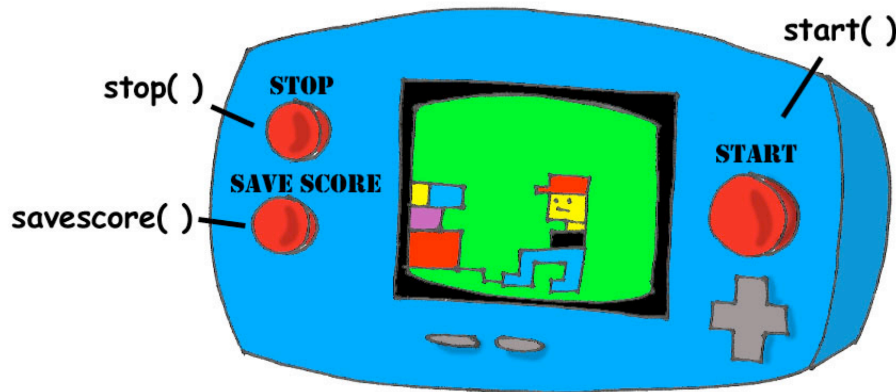
Программы на языке Java состоят из классов, которые представляют объекты реального мира. Люди понимают по-разному, что значит хороший стиль программирования, однако большинство согласно, что нужно стараться программировать в так называемом объектно-ориентированном стиле. Это значит, что хорошие программисты сначала решают, какие объекты включить в программу и какие Java классы будут их представлять, а только потом уже они начинают писать программу.

Классы и объекты

Давайте придумаем и обсудим класс, который будет называться VideoGame (видео игра). Этот класс может иметь несколько методов, из которых будет ясно, что могут делать объекты этого класса: начать игру, остановить её, сохранить (запомнить) счет и так далее.

А ещё этот класс может иметь какие-нибудь атрибуты, например цена, цвет экрана, количество ручек управления, и все такое прочее.

- ✓ Классы Java могут иметь и методы и атрибуты.
- ✓ Методы определяют, что класс может сделать.
- ✓ Атрибуты – это характеристики класса.



Наш класс может выглядеть вот так:

```
class VideoGame {
  int color; // цвет
  String int price; // цена

  void start () {
  }
  void stop () {
  }
  void saveScore(String playerName, int score) {
  }
}
```

Класс VideoGame будет похож на другие классы, которые представляют видео игры – у всех у них есть экраны, правда, разного цвета и размера, все они выполняют похожие действия, и все они стоят денег.

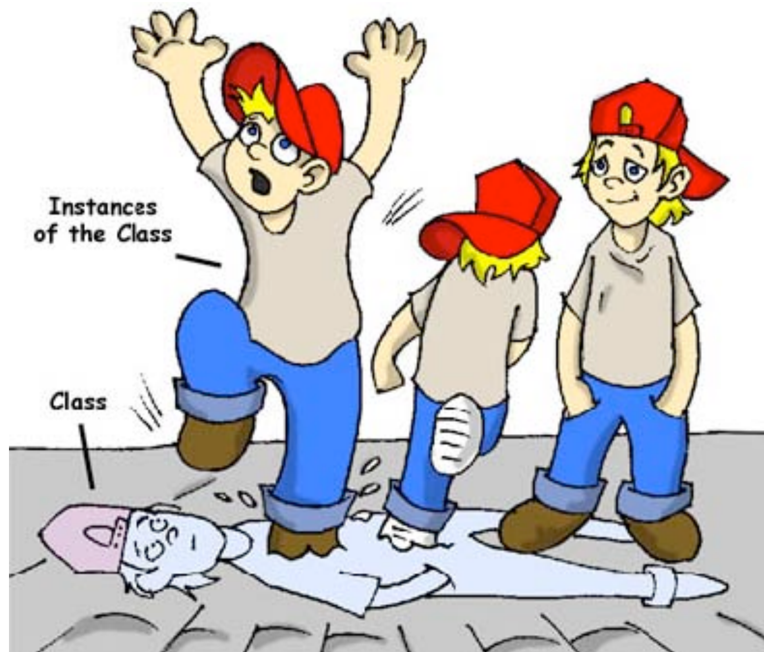
А теперь давайте добавим побольше конкретных деталей и создадим другой класс называемый GameBoyAdvance – когда-то популярная в Америке электронная игра. Этот класс тоже принадлежит семейству видео игр, однако он будет иметь некоторые атрибуты, которые имеет только игра GameBoy Advance, например тип кассеты.

В этом примере класс GameBoyAdvance объявляет два атрибута – cartridgeType and screenWidth и два метода – startGame() и stopGame(). Но эти методы ещё не готовы выполнять какие-либо действия, потому что у них нет никакого программного кода между фигурными скобочками.

```
class GameBoyAdvance {
```

```
    String cartridgeType; // тип кассеты
    int screenWidth; // ширина экрана
    void startGame() {
    }
    void stopGame() {
    }
}
```

Фабричное описание игры GameBoy Advance имеет такое же отношение к уже сделанной игре, как Java-класс к его экземпляру в памяти компьютера. А процесс изготовления настоящих игр на основе такого описания, похож на процесс создания экземпляров объектов GameBoyAdvance на языке Java.



Во многих случаях программа может пользоваться классом только после создания его экземпляра. Производители игр ведь тоже создают тысячи копий по одному и тому же описанию. Не смотря на то, что эти копии представляют тот же самый класс, их атрибуты могут иметь разные значения - какие-то из них голубые, какие-то перламутровые, и так далее. Иными словами, программа может создавать множество экземпляров объектов GameBoyAdvance.

Типы Данных

Помимо слова *класс* вам придется привыкать к ещё одному новому значению слова *объект*.

А фраза “создать экземпляр объекта” – просто значит создать копию объекта в памяти компьютера согласно описанию этого класса.

Переменные представляют собой атрибуты класса, параметры метода или просто могут использоваться для краткосрочного хранения каких-нибудь данных. Переменные сначала должны быть объявлены, и только после этого ими можно пользоваться.

Помните уравнения типа y=x+2? На языке Java вам придется объявить переменные x и y, используя какой-нибудь числовой тип данных, например целое число (integer или int) или число двойной длины (double):

```
int x;
int y;
```

Следующие две строчки кода присваивают значение этим переменным. Если ваша программа присвоит переменной икс значение пять, переменная игрек будет равна семи:

```
x=5;
y=x+2;
```

Java разрешает менять значение переменной немного необычным способом. Вот например, как можно изменить значение переменной игрек с пяти на шесть:

```
int y=5;
y++;
```

Несмотря на два знака плюс, Java увеличит значение игрека на единичку. А после вот этого примера, значение переменной myScore тоже шесть:

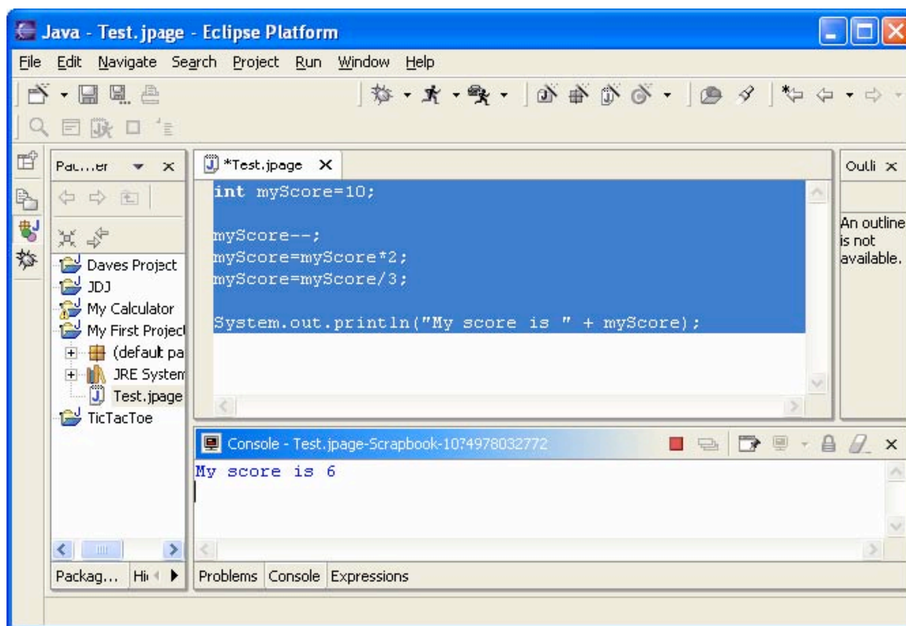
```
int myScore=5;
myScore=myScore+1;
```

Точно также можно делать умножение, деление и вычитание, вот смотрите:

```
int myScore=10;
```

```
myScore--;
myScore=myScore*2;
myScore=myScore/3;
System.out.println("My score is " + myScore);
```

Что-же напечатает этот код (кстати, “My score is” переводится как ”Мой счет”)? У приложения Eclipse, где мы теперь пишем программы, есть классная штука под названием черновик (scrapbook) которая позволяет вам легко тестировать любой кусочек кода даже без создания класса. Выберите меню *File*, *New*, *Scrapbook Page* и напечатайте слово Test – это будет имя вашего файла-черновика. Теперь напечатайте в черновике пять строчек предыдущего примера, высветите их и нажмите на кнопочку с маленьким увеличительным стеклом:



Чтобы получить результат вычислений, просто нажмите на закладку Console внизу экрана:

My score is 6

В этом примере параметр метода `println()` был склеен из двух кусочков – текста “My score is ” и значения переменной myScore, которая была равна шести. Такое “склеивание” строк из кусочков называется *конкатенация* (*concatenation)*. Несмотря на то, что переменная myScore хранит не текст, а число: Java достаточно умна, чтобы преобразовать эту переменную в тип данных `String` и потом уже приклеить ее значение к тексту “*My Score is*”.

Вот ещё несколько примеров того, как можно менять значения переменных:

```
myScore=myScore*2;      // то же что myScore*=2;
myScore=myScore+2;      // то же что myScore+=2;
myScore=myScore-2;      // то же что myScore-=2;
myScore=myScore/2;      // то же что myScore/=2;
```

В языке Java есть восемь простых (примитивных) типов данных, и вам решать какими пользоsваться в вашей программе. Это, конечно, зависит от того, данные какого типа и размера вам нужно хранить в этих переменных:

- ✓ Четыре типа данных для хранения целых чисел – byte, short, int, and long.
- ✓ Два типа данных для значений с десятичной точкой – float и double.
- ✓ Один тип данных для хранения одной буквы – char.
- ✓ Один логический тип, называемый boolean, который может иметь только два значения: true или false (истина и ложь).

Java разрешает присваивать начальное значение переменной во время ее объявления. В народе это называется *инициализация переменных*:

```
char grade = 'A';
intirs = 12;
cha boolean playSound = false;
double nationalIncome = 23863494965745.78;
float gamePrice = 12.50f;
long totalCars =4637283648392l;
```

В последних двух строчках f значит float, а l значит long.

Если вы все-же забудете инициализировать переменные, Java сама присвоит ноль числовым переменным, false переменных типа boolean, и специальный код '\u0000' переменным типа char.

А ещё есть специальное ключевое слово final, и если оно присутствует в объявлении переменной, вам будет разрешено присвоить значение этой переменной только один раз и вы не сможете уже изменить это значение после. В некоторых языках программирования final-

переменные называются *константами*. Принято называть константы большими буквами.

```
final String STATE_CAPITAL="Вашингтон";
```

Помимо примитивных типов данных, вы можете использовать классы для объявления переменных. У каждого примитивного типа данных есть соответствующий класс-обертка, например `Integer`, `Double`, `Boolean`, и другие. Эти классы имеют много полезных методов, чтобы преобразовывать данные из одного типа в другой.

Примитивный тип данных char может хранить только одну букву, но в языке Java существует класс `String` для работы с более длинным текстом:

```
String lastName="Смит";
```

Имена переменных не могут начинаться с цифры и не могут содержать пробелы.

- Бит это самая маленькая порция данных, которая может храниться в памяти. Вы можете хранить в бите только 1 или 0.
- Байт состоит из восьми битов.
- char занимает два байта в памяти компьютера.
- int и float занимают четыре байта памяти.
- Переменным long и double нужно по восемь байтов.

Числовые типы данных, которые занимают больше памяти, могут хранить большие величины.

- 1 килобайт (KB) - это 1024 байтов
- 1 мегабайт (MB) - это 1024 килобайтов
- 1 гигабайт (GB) имеет 1024 мегабайтов

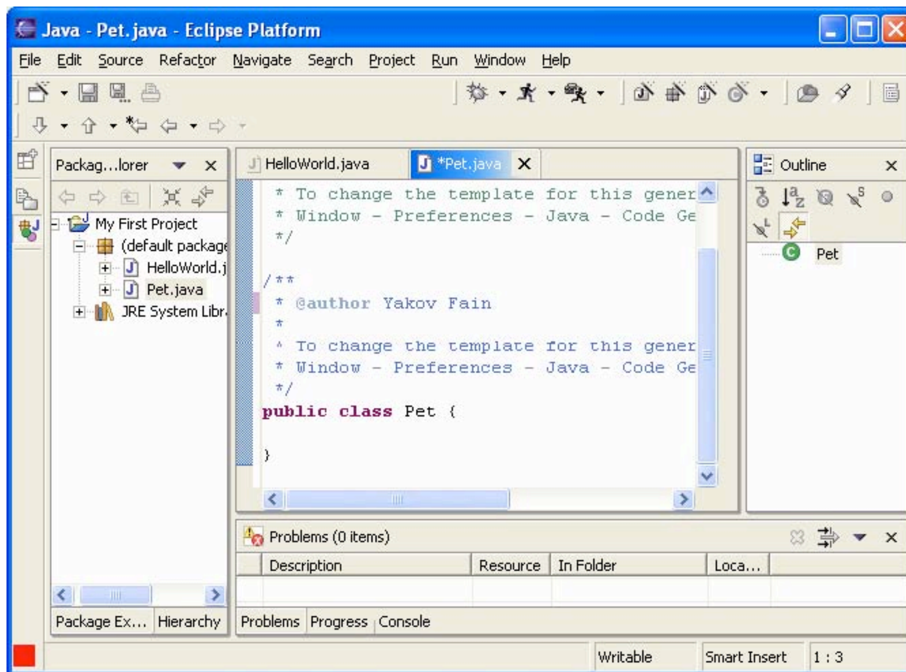
Создаём Домашнее Животное

Давайте придумаем и создадим класс Домашнее Животное (по-английски просто Pet). Сначала нужно решить, какие действия наш Pet сможет выполнять. Как насчет есть, спать и говорить (eat, sleep, say) ?

Мы запрограммируем эти действия в методах класса Pet. А ещё мы дадим нашему домашнему животному такие аттрибуты: возраст (age), рост (height), вес (weight) и цвет (color).

Начнем с создания нового класса по имени Pet в проекте *My First Project* из второй главы, но при этом не ставьте птичку, требующую создания метода `main()`.

Ваш экран будет выглядеть примерно так:



Теперь мы готовы объявить аттрибуты и методы в классе Pet. Код в классах и методах должен быть окружен фигурными скобками. Каждая отрывающая скобка должна иметь свою закрывающую:

```
class Pet{
}
```

Давайте выберем типы данных для аттрибутов нашего класса. Я предлагаю `int` для возраста, `float` для веса и роста, `String` для цвета.

```
class Pet{
```

```
  int age;
  float weight;
  float height;
  String color;
}
```

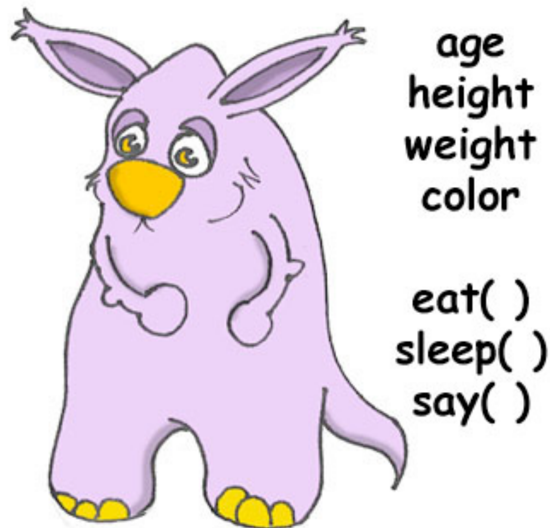
Добавим несколько методов в наш класс. Здесь нужно решить будут ли эти методы иметь параметры и возвращать какие-нибудь данные:

- ✓ Метод sleep() будет просто печатать фразу *Спокойной ночи, до завтра* – ему не нужны никакие параметры и он не будет возвращать никаких значений.
- ✓ То же самое относится и к методу eat() – он будет печатать сообщение *Я очень голоден…давайте перекусим чипсами!*
- ✓ Хотя метод say() тоже будет печатать сообщение, но наше домашнее животное будет ещё и “произносить” слово или фразу, которую мы ему дадим, как *параметр*. Этот метод будет строить фразу, включающую значение этого параметра, после чего этого фраза будет возвращаться вызывающей программе.

Новая версия класса Pet будет выглядет так:

```
public class Pet {
  int age;
  float weight;
  float height;
  String color;

  public void sleep(){
     System.out.println("Спокойной ночи! До завтра");
  }

  public void eat(){ System.out.println(
       "Я очень голоден, давайте перекусим чипсами!");
  }

  public String say(String aWord){
     String petResponse = "Ну ладно!! " +aWord;
     return petResponse;
  }
}
```

Этот класс представляет вот такое симпатичное существо из реального мира:



Давайте поговорим о заголовке (сигнатуре) метода sleep():

```
public void sleep()
```

Этот заголовок говорит нам, что метод sleep() можно вызывать из любого другого класса (`public`) и что метод не возвращает никаких данных (`void`). Пустые скобочки значат, что этот метод не имеет параметров – ему ведь не нужны никакие данные из окружающей среды, потому что он всегда печатает один и тот же текст. Заголовок метода say() выглядит так:

```
public String say(String aWord)
```

Этот метод тоже можно вызывать из любого другого класса, но он ещё и возвращает какой-то текст – это и есть роль ключевого слова `String`, стоящего перед именем метода. А кроме того, этот метод ожидает какие-то текстовые данные извне и для этого в заголовок метода включен параметр `String aWord`.

А как же определить – должен ли метод возращать данные? Если метод выполняет какие-то действия над данными и должен передать результат этих действий вызывающему классу, то он должен возвращать данные. Вы можете возразить, что у класса Pet нет никакого вызывающего класса! Верно, поэтому мы сейчас и создадим класс PetMaster (Хозяин). У этого класса будет метод `main()`, содержащий код для работы с классом Pet.

Создайте новый класс PetMaster, но на этот раз поставьте птичку в Eclipse возле вопроса *создавать-ли метод main()*. Не забывайте, что без этого метода вы не сможете стартовать программу. Добавьте несколько строчек кода к классу, который сделал для вас Eclipse, чтобы он выглядел так:

```
public class PetMaster {
  public static void main(String[] args) {
   String petReaction;
   Pet myPet = new Pet();
   myPet.eat();
   petReaction = myPet.say("Чик!! Чирик!!");
   System.out.println(petReaction);
   myPet.sleep();
  }
}
```

Не забудьте нажать *Ctrl-S* , чтобы сохранить и откомпилировать этот класс! А чтобы стартовать программу PetMaster, выберите следующие меню в Eclipse *Run*, *Run…*, *New* и напечатайте имя главного класса -

PetMaster. Нажмите на кнопку *Run* , и программа напечатает такой текст:

```
Я очень голоден, давайте перекусим чипсами!
Ну ладно!! Чик!! Чирик!!
Спокойной ночи, до завтра
```

PetMaster – это вызывающий класс и он сначала создает экземпляр объекта Pet. Он объявляет переменную myPet and и использует оператор `new`:

```
Pet myPet = new Pet();
```

Эта строчка объявляет переменную myPet типа Pet (да это так, вы можете использовать любые классы, созданные вами, как новые типы данных). Теперь переменная myPet знает место в памяти, где был создан экземпляр объекта Pet, и можно пользоваться этой переменной, чтобы вызывать любые методы класса Pet, например:

```
myPet.eat();
```

Если метод возвращает какое-нибудь значение, его можно вызывать по-другому. Объявите переменную того-же типа, что и возвращаемое значение и вызывайте метод так, чтобы присвоить это значение переменной, например так:

```
String petReaction;
petReaction = myPet.say("Чик!! Чирик!!");
```

Теперь возвращенное значение находится в переменной petReaction и его очень просто можно распечатать:

```
System.out.println(petReaction);
```

Наследование – Рыбка Тоже Домашнее Животное

Наш класс Pet познакомит вас с ещё одной важной особенностью языка Java, которая называется *наследование (inheritance)*. В реальном мире каждый человек наследует что-то от своих родителей. В мире Java вы тоже можете создать новый класс по типу уже существующего.

Класс Pet, и ведет себя, и имеет атрибуты типичные для многих домашних животных – они едят, спят, некоторые из них издают звуки, их кожа имеет цвет и так далее. С другой стороны, домашние животные отличаются друг от друга – собаки лают, рыбки беззвучно плавают, попугайчики разговаривают лучше, чем собаки.

И все-же, все они спят, едят, и имеют рост и вес. Поэтому гораздо легче создать класс Fish (рыба) так, чтобы он унаследовал общие черты и поведение у класса Pet, чем каждый раз создавать с начала классы для собак, попугаев и рыб.

Для этого и существует специальное ключевое слово `extends`:

Теперь вы имеете полное право сказать что Fish - это *подкласс* класса Pet, а класс Pet – это *супер-класс* класса Fish. Мы использовали класс Pet, как свообразный шаблон для создания класса Fish.

```
class Fish extends Pet{
```

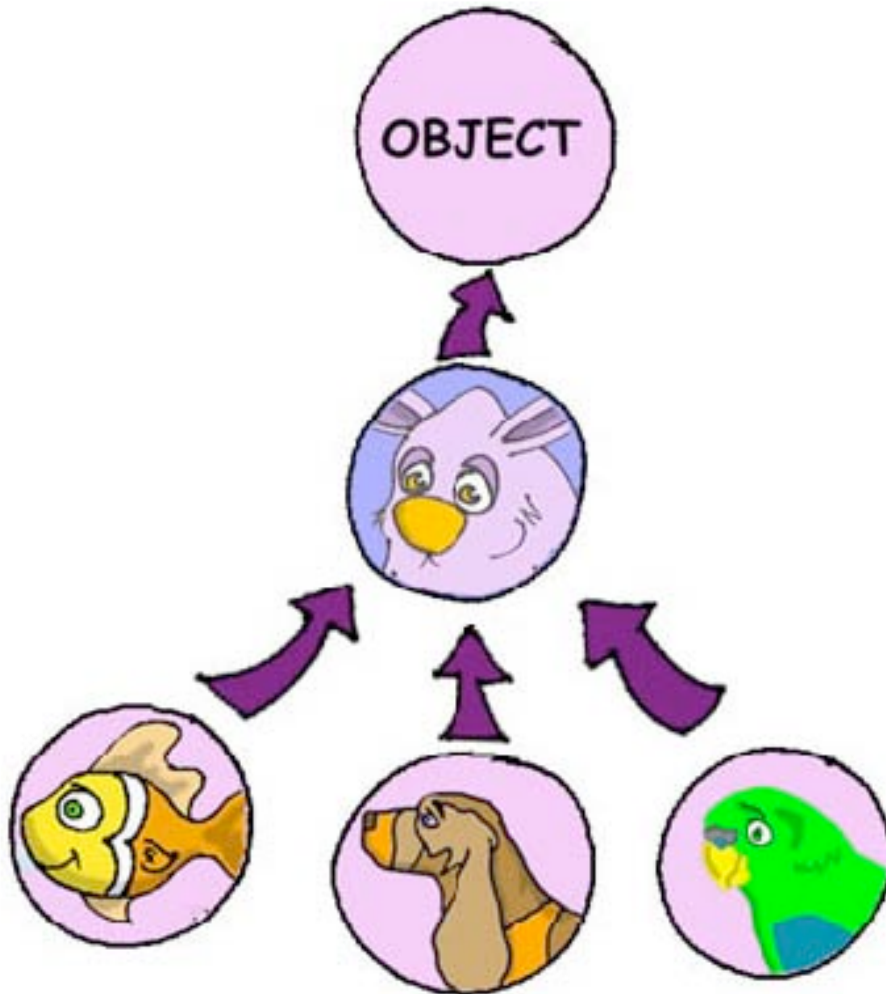
```
}
```

Даже если вы оставите класс Fish таким, как он есть сейчас, всё равно уже можно использовать каждый метод и атрибут, унаследованный из класса Pet. Вот посмотрите:

```
Fish myLittleFish = new Fish();
myLittleFish.sleep();
```

Хоть мы ещё и не объявляли никаких методов в классе Fish, уже можно вызывать метод sleep(), находящийся в его супер-классе!

Нет ничего легче создания подклассов в приложении Eclipse! Выберите меню *File*, *New*, *Class* и напечатайте слово Fish, как имя класса. Замените в поле супер-класс java.lang.Object на слово Pet.



Не забывайте, что мы создаем подкласс класса Pet, чтобы добавить то, что присуще только рыбам, а общий для всех животных код, объявленный в супер-классе, мы просто используем.

Не все домашние животные могут нырять, но рыбки, конечно же, могут. Давайте добавим к классу Fish метод dive() - нырни.

У метода dive() есть параметр howDeep, который "говорит" рыбке, как глубоко она должна нырнуть. А ещё мы объявили переменную currentDepth, куда будем помещать текущее значение глубины при каждом вызове метода dive(). Этот метод возвращает значение переменной currentDepth вызывающему классу.

Сделайте, пожалуйста, вот такой класс FishMaster:

Пора рассказать маленький секрет – все классы в языке Java унаследованы из супер-дупер класса Object, даже если вы и не использовали ключевое слово extends.

В отличие от людей, Java-классы не могут иметь двух родителей.

А если бы у нас это было как в языке Java , дети не были бы "подклассами" своих родителей, а все мальчики происходили бы от Адама, а девочки – от Евы ☺.

```
public class Fish extends Pet {
  int currentDepth=0;
  public int dive(int howDeep){
    currentDepth=currentDepth + howDeep;
    System.out.println("Ныряю на глубину "
                          + howDeep + " футов");
    System.out.println("Я на глубине "
       + currentDepth + " футов ниже уровня моря");
    return currentDepth;
  }
}
```

Метод main() в классе FishMaster создает экземпляр объекта Fish и дважды вызывает его метод dive() с разными параметрами. После этого он вызывает метод sleep().

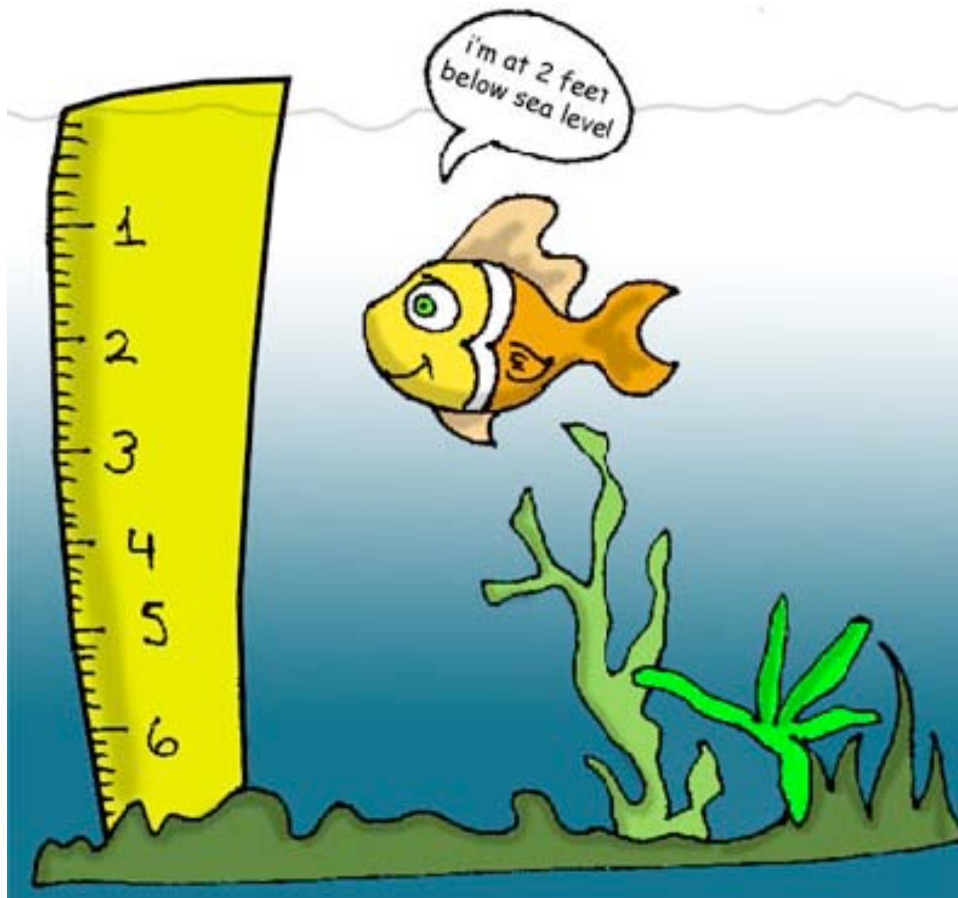
```
public class FishMaster {
  public static void main(String[] args) {
    Fish myFish = new Fish();
    myFish.dive(2);
    myFish.dive(3);
```

```
    myFish.sleep();
  }
}
```

Во время выполнения FishMaster напечатает следующее:

```
Ныряю на глубину 2 футов
Я на глубине 2 футов ниже уровня моря
Ныряю на глубину 3 футов
Я на глубине 5 футов ниже уровня моря
Спокойной ночи, до завтра
```

Вы заметили, что FishMaster вызывает не только методы объявленные в классе Fish, но также и метод sleep() его супер-класса Pet? То-то! В этом и есть вся прелесть наследования – вам не нужно копировать код из класса Pet. Просто напишите слово extends и класс Fish сможет пользоваться методами класса Pet!



Да, вот ещё что, хотя метод dive() и возвращает значение переменной currentDepth, наш FishMaster им не пользуется. Это не беда, просто нашему классу FishMaster оно не нужно. Но каким-нибудь другим классам, которые тоже могут работать с классом Fish, это значение может быть очень даже полезно. Представьте, например, класс FishTrafficDispatcher (регулировщик движения рыб), который должен знать положения других рыб в море, прежде чем разрешить ныряние во избежание дорожно-транспортных происшествий ☺.

Переопределение методов

Вы, конечно, знаете, что рыбы не говорят (по крайней мере он не делают это громко). Но наш класс Fish был унаследован из класса Pet, у которого есть метод say(). Это значит, что вы беспрепятственно можете написать что-то в этом роде:

```
myFish.say();
```

Ну и ну, наши рыбки заговорили… Чтобы избежать этого, в классе Fish нужно переопределить (override) метод say(), обьявленный в классе Pet. Это работает так: если вы объявляете в под-классе метод имеющий точно такой-же заголовок, как в его-же супер-классе, Java выполнит метод под-класса, вместо метода супер-класса. Давайте добавим к классу Fish метод say().

```
public String say(String something){
  return "Ты чё не знаешь, что рыбы не разговаривают?";
}
```

А теперь вызовем метод say() из метода main() класса FishMaster:

```
myFish.say("Привет");
```

Выполните эту программу и она напечатает следующее:

```
Ты чё не знаешь, что рыбы не разговаривают?
```

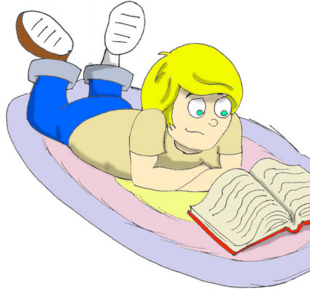
Это подтверждает, что метод say() класса Pet был переопределен.

Вот это да! Мы изучили много нового в этой главе – давайте передохнём.

Если заголовок метода включает ключевое слово final, такой метод переопределить нельзя, например:

```
final public void sleep(){…}
```

Дополнительное чтение



1.Java Data Types:

http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html

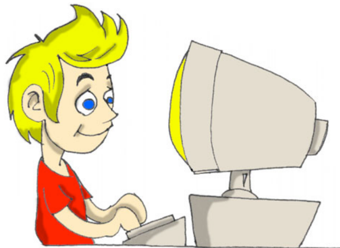
2.About inheritance:

http://java.sun.com/docs/books/tutorial/java/concepts/inheritance.html

Практические упражнения

1. Создайте новый класс Car (автомобиль) и включите в него следующие методы:

```
public void start()
public void stop()
public int drive(int howlong)
```



Метод drive() (едь) должен возвращать общее расстояние пройденное автомобилем за заданное время. Используйте следующую формулу для рассчёта расстояния:

```
distance = howlong*60;
```

2. Создайте ещё один класс CarOwner (хозяин автомобиля), который будет создавать экземпляр объекта Car и вызывать его методы. Результат каждого такого вызова должен быть напечатан с помощью `System.out.println()`.

Практические упражнения для умников и умниц

Сделайте подкласс класса Car, назовите его JamesBondCar (автомобиль Джемса Бонда) и переопределите в нем метод drive(). Используйте следующую формулу дла рассчёта расстояния:

```
distance = howlong*180;
```

Будьте изобретательны! Печатайте смешные сообщения.

Глава 4. Основные конструкции языка Java

В программу, написанную на языке Java, можно добавлять любые текстовые комментарии, чтобы пояснить, для чего нужна конкретная строчка кода, метод или класс. Через некоторое время бывает тяжело вспомнить, почему вы написали программу именно так, а не иначе. Ещё одна причина писать комментарии - это помочь другим программистам понять ваш код.

Комментарии в программе

Существует три типа комментариев:

1. Если ваш комментарий укладывается в одну строку, начните его с двух косых черт:

   ```
   // Этот метод вычисляет расстояние
   ```

2. Более длинные многострочные комментарии должны быть окружены символами /* и */, например:

```
/* следующие 3 строки кода
   нужны для сохранения позиции рыбы.
*/
```

3. Вместе с Java поставляется утилита javadoc, которая позволяет извлечь комментарии из вашей программы в отдельный файл справки. Этот файл может быть использован в качестве технической документации для ваших программ. Такие

комментарии должны находиться между символами /** и */. Только самые важные комментарии, такие как описание класса или метода, следует помещать между этими символами.

```
/**  Этот метод вычисляет размер скидки  в
     зависимости от цены. Если цена больше $100,
     скидка = 20%,в противном случае только 10%.

*/
```

Далее я буду добавлять комментарии в примеры кода, чтобы дать лучшее представление о том, где и как их использовать.

Принятие решений с помощью оператора if

В нашей жизни мы постоянно принимаем решения: *если она скажет мне так – то я отвечу ей вот так, в противном случае я сделаю по-другому*. В Java есть оператор if, который проверяет, является ли некое выражение истинным (`true`) или ложным (`false`).

На основании результатов этого выражения, выполнение программы разветвляется, и только одна соответствующая часть кода будет исполняться.

Например, если условие *Хочу ли я пойти к бабушке?* возвращает true, мы поворачиваем налево, в противном случае идём направо.

Если выражение возвращает истину, JVM будет выполнять код находящийся между первым фигурными скобками, в противном случае выполнится код, находящийся в блоке else. Например, если цена больше 100 долларов, то сделать 20% скидки, в противном случае только 10%.

```
// Более дорогие товары продаются со скидкой 20%

if (price > 100){
```

```
    price=price*0.8;
    System.out.println("Ваша скидка 20%");

}

else{

  price=price*0.9;
  System.out.println("Ваша скидка 10%");

}
```

Давайте изменим метод dive() в классе Fish, чтобы ограничить сотней метров глубину, на которой может плавать наша рыбка:

```
public class Fish extends Pet {

  int currentDepth=0;

  public int dive(int howDeep){

    currentDepth=currentDepth + howDeep;

      if (currentDepth > 100){

          System.out.println("Я маленькая рыбка "+
                    " и не могу плавать глубже 100 метров");

          currentDepth=currentDepth - howDeep;

      }else{

        System.out.println("Погружаюсь ещё на " + howDeep +
                                              " метров");

        System.out.println("Я на глубине " + currentDepth +
                                              " метров");
      }

      return currentDepth;
}

  public String say(String something){

       return "Разве вы не знаете, что рыбы не говорят?";

  }
}
```

Теперь сделаем небольшое изменение в классе `FishMaster` – давайте попробуем погрузить нашу рыбку на глубину больше 100 метров:

```
public class FishMaster {
```

```
  public static void main(String[] args) {

    Fish myFish = new Fish();

    // Попробуем заставить рыбу погрузиться ниже 100 метров

    myFish.dive(2);
    myFish.dive(97);
    myFish.dive(3);

    myFish.sleep();

  }
}
```

Запустите эту программу, и она выдаст следующее:

```
Погружаюсь ещё на 2 метра

Я на глубине 2 метров

Погружаюсь ещё на 97 метров

Я на глубине 99 метров

Я маленькая рыбка и не могу плавать глубже 100 метров

Спокойной ночи, увидимся утром
```

Логические операторы

Иногда, чтобы принять решение, необходимо проверить более одного условного выражения. Например, если название штата Техас (Texas) или Калифорния (California), нужно добавить налог штата к цене каждого товара в магазине. Это пример *логического или* – либо Техас либо Калифорния. Это работает так – если любое из двух условий истинно (`true`), результат всего выражения - тоже истина (`true`).

В Java знак *логического или* обозначается одной или двумя вертикальными линиями. В следующем примере я буду использовать переменную типа String. У этого класса в Java есть метод `equals()` для проверки строк на равенство и я буду использовать его, чтобы проверить равна ли переменная state (штат) *Техас* или *Калифорния*:

```
if (state.equals("Texas") | state.equals("California"))
```

Можно также использовать две вертикальные линии в операторе `if`:

```
if (state.equals("Texas") || state.equals("California"))
```

Разница между этими условиями в том, что если вы используете две вертикальные линии, и выражение стоящее слева истинно (`true`), то правое – даже не проверяется. Если используется одна вертикальная черта, JVM всегда будет вычислять результат обоих выражений.

Логическое и задаётся одним или двумя амперсандами (`&&`). Всё выражение истинно, если истинна каждая часть этого выражения. Например, снимает налог штата, только если этот штат (state) – Нью-Йорк и цена (price) больше 110 долларов. Оба условия должны быть истинными *одновременно*:

```
if (state.equals("New York") && price >110)
```

или

```
if (state.equals("New York") & price >110)
```

Если используется двойной амперсанд и первое выражение ложно (false), то второе выражение даже не будет проверяться, т.к. вне зависимости от него, всё выражение будет ложным. При одном амперсанде будут вычислены оба условия.

Логическое не (negation) обозначается восклицательным знаком. Оно меняет значение выражения на противоположное. Например, если некоторое действие может быть выполнено только, если штат *не* Нью-Йорк, можно написать так:

```
if (!state.equals("New York"))
```

Другой пример – следующие выражения абсолютно идентичны:

```
if (price < 50)

if (!(price >=50))
```

Во втором случае *логическое не* применяется к результату вычисления выражения в скобках.

Условный оператор

Существует ещё один тип оператора if, который называется *условный оператор*. Он используется для того, чтобы присвоить значение переменной, в зависимости от истинности выражения, стоящего перед вопросительным знаком. Если выражение истинно, присваивается

значение стоящее сразу после вопросительного знака, в противном случае – значение стоящего после двоеточия:

```
discount = price > 50? 10:5;
```

Если цена больше пятидесяти, значение переменной discount (скидка) будет равно десяти, в противном случае - пяти. Это просто более короткая форма записи обычного оператора if:

```
if (price > 50){

   discount = 10;

} else {

   discount = 5;

}
```

Использование else if

Также можно создавать составные операторы `if` содержащие несколько `else if` блоков. Давайте создадим класс `ReportCard` (табель по русски), у которого будет метод `main()`, а так же метод с одним аргументом, в который передаётся численный результат некого теста. В зависимости от этого числа, метод будет печатать вашу оценку по буквенной системе оценок (A, B, C, D). Назовём этот метод `convertGrades()`.

```
public class ReportCard {

   /**
   Метод convertGrades принимает один целочисленный аргумент
   - результат теста и возвращает символ A, B, C or D в
   зависимости от этого аргумента.
   */
   public char convertGrades( int testResult){

      char grade;

      if (testResult >= 90){
           grade = 'A';
      }else if (testResult >= 80 && testResult < 90){
           grade = 'B';
      }else if (testResult >= 70 && testResult < 80){
           grade = 'C';
      }else {
           grade = 'D';
      }

      return grade;
```

```
   }

   public static void main(String[] args){

      ReportCard rc = new ReportCard();
      char yourGrade = rc.convertGrades(88);

      System.out.println("Ваша первая оценка " + yourGrade);
      yourGrade = rc.convertGrades(79);
      System.out.println("Ваша вторая оценка " + yourGrade);
   }

}
```

Помимо использования `else if` в этом примере демонстрируется, как использовать переменные типа `char`. Также можно увидеть, что при помощи оператора `&&` можно проверить, попадает ли число в указанный диапазон. Нельзя написать "если `testResult` между 80 и 89", в Java нужно писать, что `testResult` должен быть больше или равен 80, *и (&)* в то же время меньше 89, вот таким образом:

```
testResult >= 80 && testResult < 89
```

Подумайте, почему мы здесь не используем оператор ||.

Оператор `switch` и принятие решений

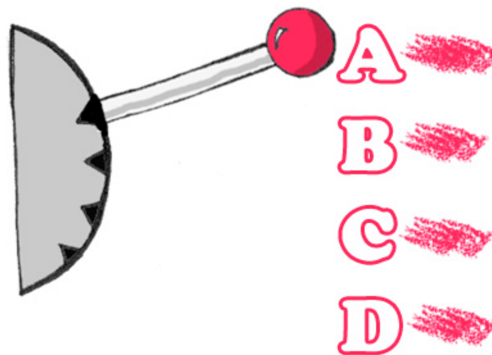
Оператор `switch` иногда используется как альтернатива `if`. Значение переменной стоящей после оператора `switch` вычисляется, и программа переходит только к одному из `case` блоков, аргумент которого совпадает с результатом этого вычисления:

```
public static void main(String[] args){

 ReportCard rc = new ReportCard();

 char yourGrade = rc.convertGrades(88);

  switch (yourGrade){

    case 'A':
       System.out.println("Превосходная работа!");
       break;

    case 'B':
       System.out.println("Хорошая работа!");
       break;

    case 'C':
       System.out.println("Надо подтянуть знания!");
       break;
```

```
    case 'D':
       System.out.println("Будь посерьёзнее!");
       break;
  }
}
```

Не забудьте написать ключевое слово break в конце каждого case блока, чтобы после завершения выполнения его кода, произошёл выход из оператора switch. Если вы не напишете break, то напечатаются все четыре строки, не смотря на то, что значение переменной yourGrade имеет только одно значение.

Раньше оператор switch имел ограничение – переменная, которая в него передаётся, должна была иметь один из следующих типов:
char
int
byte
short
enum



Начиная с Java 1.7, тип String так же может использоваться в операторе switch. Кроме того, благодаря автобоксингу, который появился в Java 1.5 могут использоваться типы обёртки: Character, Byte, Short, и Integer.

Как долго живут переменные?

Внутри метода convertGrades() класса ReportCard объявляется переменная grade. Переменная, объявленная внутри любого метода называется *локальной*. Это означает, что она существует и доступна только *внутри этого метода*. После того, как метод выполнился, локальная переменная автоматически удаляется из памяти.

Программисты так же используют термин *область действия (scope)* , чтобы задать сколько времени та или иная переменная будет существовать.

Если переменная должна быть использована несколькими методами, то её нужно объявить вне всех методов. В классе Fish, currentDepth это *атрибут класса (member variable)*. Срок жизни этих переменных определяется сроком жизни объекта Fish, поэтому они еще называются *атрибутами экземпляра класса (instance variables)*. Такие переменные могут совместно использоваться всеми методами класса, и, в некоторых случаях, даже быть доступными для других классов. Например, в *выражении* `System.out.println()` используется переменная out, которая объявлена в классе `System`.

Минуточку! А разве можно использовать атрибут класса `System`, если мы не создавали экземпляра этого класса? Да можем, если переменная объявлена с помощью ключевого слова static (статический). Если объявление атрибута класса или метода начинается со слова static, то не обязательно создавать экземпляр класса, чтобы их использовать. Статические атрибуты класса используются для хранения значений общих для всех экземпляров класса.

Например, метод convertGrades() может быть объявлен в классе ReportCard как статический, потому что в реализации этого метода для чтения и записи не используются атрибуты, специфичные для конкретного экземпляра класса. Статический метод `sqrt()` из класса `Math` можно вызывать вот так:

```
double squareRoot = Math.sqrt(4.0);
```

Специальные методы: конструкторы

В Java для создания экземпляров классов и выделения под них памяти используется оператор new, например:

```
Fish myFish = new Fish();
```

Круглые скобки после слова Fish говорят о том, что у этого класса определен метод Fish(). Так и есть, существуют специальные методы, которые называются *конструкторами (constructors)* , и у этих методов есть следующие особенности:

- ✓ Конструкторы вызываются только один раз при создании объекта в памяти.
- ✓ Они должны называться так же, как называется класс.
- ✓ Они ничего не возвращают, не нужно даже писать слово void в сигнатуре этого метода.

У класса может быть несколько конструкторов. Если вы не написали ни одного конструктора, во время компиляции Java автоматически создаст за вас так называемый *пустой конструктор по умолчанию (default no-argument constructor)*. Вот почему компилятор никогда не будет "ругаться" на выражение new Fish(), даже если в классе Fish вы не объявили ни одного конструктора.

В основном, конструкторы используются для присваивания начальных значений атрибутам класса, к примеру, следующая версия класса Fish включает конструктор с одним аргументом, который задаёт начальное значение атрибута currentDepth равным значению аргумента конструктора.

```
public class Fish extends Pet {

    int currentDepth;

      Fish(int startingPosition){

            currentDepth=startingPosition;

      }
}
```

Теперь класс FishMaster может создать экземпляр класса Fish и задать начальное положение рыбки. Ниже создаётся экземпляр класса Fish, который изначально погружает рыбку в море на глубину 20 метров:

```
Fish myFish = new Fish(20);
```

Для класса, в котором был определён конструктор с аргументами, конструктор по умолчанию создаваться автоматически не будет. Если вам необходим конструктор без аргументов - напишите его.

Ключевое слово this

Ключевое слово this полезно, когда нужно сослаться на экземпляр класса внутри объекта этого класса. Рассмотрим следующий пример:

```
class Fish {
  int currentDepth ;

  Fish(int currentDepth){
      this.currentDepth = currentDepth;

  }
}
```

Здесь идентификатор this помогает избежать конфликта имён, например this.currentDepth ссылается на атрибут класса currentDepth, в то время как currentDepth ссылается на значение аргумента конструктора.

Другими словами, экземпляр класса Fish указывает на самого себя с помощью слова this.

Другой важный пример использования ключевого слова `this`, вы встретите в главе 6 в секции *Как передавать данные между классами*.

Массивы

Предположим, программа должна сохранить имена четырёх игроков. Вместо того, чтобы объявлять четыре переменные типа `String`, можно объявить *массив*, который содержит четыре *элемента* типа `String`. Массивы обозначаются с помощью квадратных скобок, помещённых после типа данных или после имени переменной:

```
String [] players;
```

или

```
String players[];
```

Эти инструкции сообщают компилятору Java, что вы планируете сохранить несколько строк в массиве players. Каждый элемент массива имеет свой индекс, начиная с нуля. В следующем примере создаётся

массив, который может хранить четыре объекта типа `String`, и затем элементам массива присваиваются значения:

```
String players[] = new String [4];

players[0] = "David";
players[1] = "Daniel";
players[2] = "Anna";
players[3] = "Gregory";
```

Необходимо знать размер массива, перед тем, как задавать значения для его элементов. Если количество элементов заранее неизвестно, массивы не могут быть использованы. В таких случаях, вместо массивов используют другие классы Java, например, `ArrayList`, но давайте сконцентрируемся на массивах.

У каждого массива есть атрибут length, который “помнит” количество элементов в массиве, и вы всегда можете узнать, как много элементов у вас есть:

```
int  totalPlayers = players.length;
```

Если во время инициализации массива, вы знаете значения всех элементов, которые будут в нём храниться, то Java позволяет создать такой массив в одну строку:

```
String [] players = {"David", "Daniel", "Anna", "Gregory"};
```

Представим себе, что в нашей игре победил второй участник и хочется его поздравить. Если имя игрока сохранено в массиве, нужно извлечь второй элемент:

```
String theWinner = players[1];
```

```
System.out.println("Поздравляем, " + theWinner + "!");
```

Этот код выведет на экран следующее:

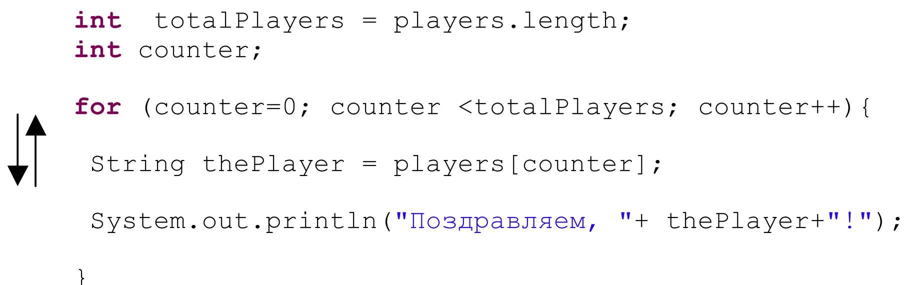
```
Поздравляем, Daniel!
```

Вы знаете, почему второй элемент имеет индекс [1]? Конечно, знаете, потому что индекс первого элемента всегда [0].

Массив игроков в нашем примере *одномерный* (*one-dimensional)*, потому что мы сохраняем их в ряд. Если мы хотим сохранить значения в виде матрицы, мы можем использовать двумерный массив. Java позволяет создавать *многомерные массивы* (*multi-dimensional* arrays). Вы можете сохранять в массивах любые объекты, и я покажу, как это делается в главе 10.

Повторение действий с помощью циклов

Циклы используются, для того чтобы выполнить какое-нибудь действие несколько раз. Например, нужно напечатать поздравление для нескольких победителей. Если вы заранее знаете сколько раз повторить действие - используйте цикл `for`:



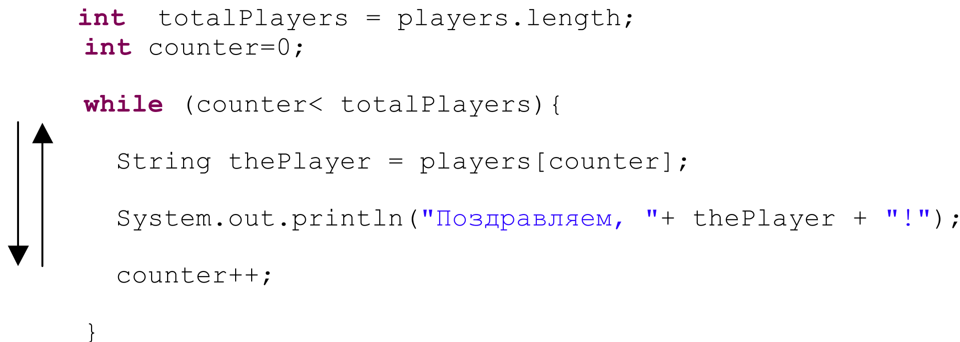
```
int  totalPlayers = players.length;
int counter;

for (counter=0; counter <totalPlayers; counter++){

 String thePlayer = players[counter];

 System.out.println("Поздравляем, "+ thePlayer+"!");

}
```

JVM выполняет каждую строку между фигурными скобками, и затем возвращается к первой строке цикла для того, чтобы увеличить значение counter (счётчика) и проверить условие завершения цикла. Этот код означает следующее:

Напечатать значение элемента массива, чей номер индекса совпадает со значением счётчика. Начать с элемента с номером 0 (counter=0), и увеличивать значение counter на единицу (`counter++`). Продолжать до тех пор, пока counter меньше totalPlayers (`counter<totalPlayers`).

Существует ещё одно ключевое слово для создания циклов - while. В таких циклах не нужно точно знать, сколько раз будет повторяться

действие, но необходимо задать условие окончания цикла. Давайте посмотрим, как можно поздравить участников игры с помощью цикла while, он закончится, когда counter (счётчик) станет равным totalPlayers:



```
int  totalPlayers = players.length;
int counter=0;

while (counter< totalPlayers){

  String thePlayer = players[counter];

  System.out.println("Поздравляем, "+ thePlayer + "!");

  counter++;

}
```

В главе 9 рассказано, как сохранить данные в файл на диске и как считать их обратно в память программы. Если вы читаете результаты игры из файла, лежащего на диске, то заранее неизвестно, как много записей было сохранено в файл. Скорее всего, придется считывать такие записи с помощью цикла `while`.

Так же можно использовать ещё два оператора в циклах: `break` и `continue`.

Ключевое слово `break` используется, чтобы выпрыгнуть из цикла, когда некоторое условие становиться истинным (true). Скажем, мы не хотим печатать более трёх поздравлений, вне зависимости от того, как много игроков учавствовало.

В следующем примере, после вывода элементов массива 0, 1 и 2, выполнение программы выйдет из цикла из-за инструкции break и продолжится со строки после закрывающей фигурной скобки.
Обратите внимание на двойной знак равенства в операторе if. Значение переменной counter *сравнивается* с числом 3. Одинарный знак равно означал бы, что значение 3 *присваивается* переменной counter. Если перепутать == с = в операторе if, то программа будет работать, но неправильно и такую ошибку иногда очень не легко найти:

```
int counter =0;
while (counter< totalPlayers){

  if (counter == 3){

   break; // Выпрыгиваем из цикла

  }
```

```
  String thePlayer = players[counter];
  System.out.println("Congratulations, "+thePlayer+ "!");
  counter++;

}
```

Оператор `continue` позволяет пропустить выполнение строк, стоящих после него и вернуться к началу цикла. Представим, что мы хотим поздравить всех, кроме Дэвида – `continue` вернёт выполнение программы к началу цикла.

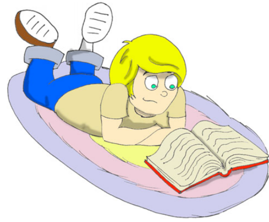
```
while (counter< totalPlayers){

  counter++;
  String thePlayer = players[counter];

  if (thePlayer.equals("David"){

      continue;

  }

  System.out.println("Congratulations, "+ thePlayer+ !");
}
```

Есть ещё один тип оператора `while`, который начинается со слова `do`, например:

```
 do {

  // Здесь ваш код, что-нибудь считающий

 } while (counter< totalPlayers);
```

В таких циклах условие проверяется *после* выполнения кода, стоящего между фигурными скобками. Таким образом, этот код выполнится *хотя бы один раз*. Циклы, в которых `while` стоит вначале, могут не выполниться ни разу, если условие ложно (`false`).

Материалы для дополнительного чтения



Официальный учебник по Java:

http://download.oracle.com/javase/tutorial/java/index.html

Практические упражнения

1. Создайте новый класс и назовите его TemperatureConverter. Добавьте в него метод для преобразования температур, с такой сигнатурой:

```
public String convertTemp
       (int temperature, char convertTo)
```

Если аргумент convertTo равен F, то температура должна быть преобразована в Фаренгейты, если C, то в Цельсии. Когда вы будете вызывать этот метод, поместите значение аргумента типа `char` в одинарные кавычки.

2. Объявите метод convertGrades() класса ReportCard как статический и удалите строку инициализации класса из метода main().

Практические упражнения для умников и умниц

Вы заметили, что в примере с ключевым словом `continue` мы переместили строку с counter++; в начало цикла?

Что бы произошло, если бы мы оставили эту строчку в конце цикла, как это было в примере с `break`?

s

Глава 5. Делаем Графический Калькулятор

Java содержит широкий набор классов, которые позволяют создавать графические приложения. Существует две основные группы классов для создания окон в Java.

AWT и Swing

В первой версии языка Java для работы с графикой имелась только библиотека - AWT. Эта библиотека – простой набор классов, таких, как `Button` (кнопка), `TextField` (текстовое поле), `Label` (текстовая метка или иконка) и другие. Вскоре была создана более совершенная библиотека, которую назвали Swing. Она так же включает в себя кнопки, текстовые поля и другие элементы управления графическими приложениями. Названия компонентов этой библиотеки начинается с буквы J. Например, `JButton`, `JTextField` и так далее.

Всё в Swing чуточку лучше, быстрее и удобнее, но в некоторых случаях наши программы могут быть запущены на компьютерах со старой версией JVM, которая может не поддерживать классов Swing.

Пакеты и ключевое слово import

Java поставляется с большим количеством полезных классов, которые организованы в пакеты (*packages*). Некоторые пакеты содержат классы для рисования графики, другие – классы для работы с интернетом и так далее. Например, класс `String` находится в пакете с названием java.lang и полное имя этого класса `java.lang.String`.

Компилятор Java знает, где найти классы, находящиеся в `java.lang`, поэтому я не указывал явно полное имя `String` в предыдущих примерах кода, но существует много других пакетов с полезными классами и ваша задача сообщить компилятору, в каком пакете содержатся классы, используемые в программе. Например, большинство классов библиотеки Swing находятся в следующих двух пакетах:

```
javax.swing
javax.swing.event
```

Было бы очень утомительно каждый раз, когда используется класс, писать его полное имя. Чтобы избежать этого, вы можете написать ключевое слово `import` всего один раз перед объявлением класса, как показано в примере:

```
import javax.swing.JFrame;
import javax.swing.JButton;

class Calculator{

  JButton myButton = new JButton();

  JFrame myFrame = new JFrame();

}
```

Ключевое слово `import` позволяет использовать короткие имена классов, такие как `JFrame` или `JButton` и сообщает компилятору, где искать эти классы.

Если нужно использовать несколько классов из одного пакета, нет необходимости перечислять каждый из них в строке с `import`, можно просто использовать символ *. В следующем примере с помощью звёздочки, все классы из `javax.swing` становятся *находимыми*:

```
import javax.swing.*;
```

Тем не менее, лучше использовать отдельные операторы import для каждого класса. Это позволяет быстрее видеть, какой класс импортируется из какого пакета. Тема пакетов будет освещена более подробно в главе 10.

Основные элементы Swing

Вот некоторые основные объекты, из которых состоят Swing-приложения:

- Окно или фрейм (*frame),* который может быть создан с помощью класса `JFrame`.

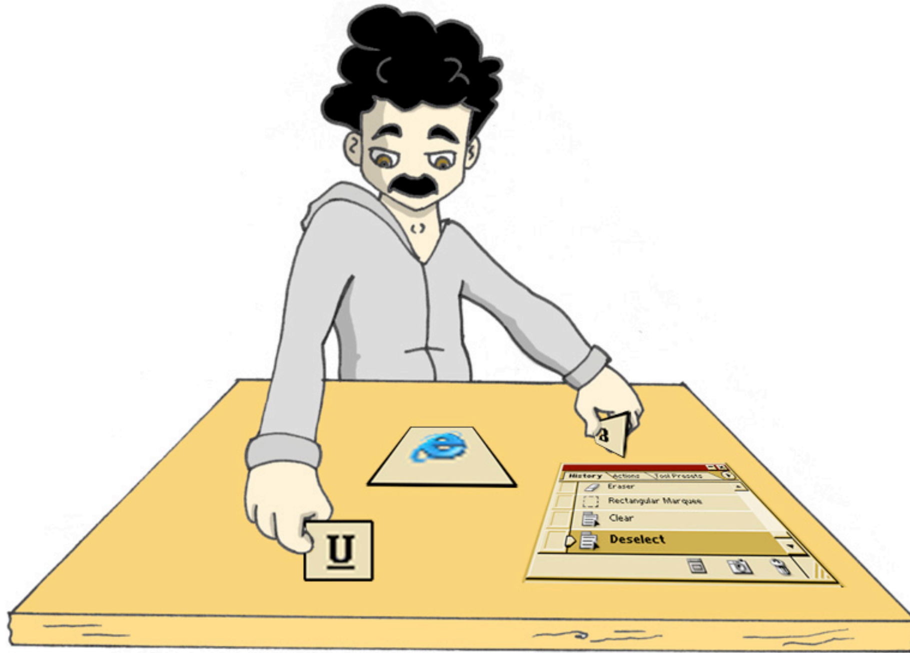
- Невидимая панель (*panel)* или, как ещё её называют, *pane (оконное стекло)* содержит все кнопки, текстовые поля, метки и другие компоненты. Панели создаются с помощью класса `JPanel`.

- Оконные элементы управления, такие как кнопки `JButton`, текстовые поля `JTextfield`, списки `JList`, и так далее.

- *Схемы размещения* (layout managers) компонент, которые помогают организовать все эти кнопки и поля на панели.

Например, можно создать экземпляр класса `JPanel` и назначить для него схему размещения. Затем создайте различные графические компоненты и добавьте их на панель. После этого добавьте панель на

фрейм, задайте его размер и сделайте его видимым.



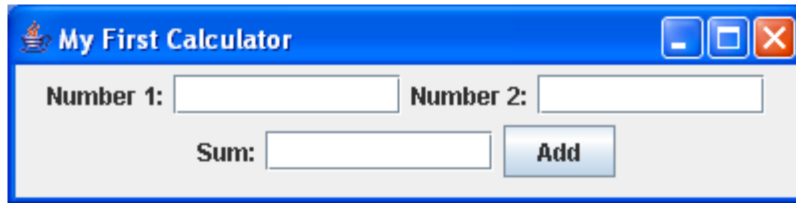
Но отображение фрейма это только половина работы. Нужно ещё добавить обработку различных событий, например нажатий на кнопки.

В этой главе я расскажу, как создавать окна с компонентами, а в следующей – обрабатывать события (*events*), которые могут произойти с компонентами окна.

Наша основная цель в этой главе – написать калькулятор, который позволяет сложить два числа и увидеть результат. Создайте новый проект в Eclipse, назовите его *My Calculator* и добавьте в него новый класс SimpleCalculator со следующим кодом:

```
import javax.swing.*;
import java.awt.FlowLayout;

public class SimpleCalculator {

 public static void main(String[] args) {

  // Создаём панель

      JPanel windowContent= new JPanel();

  // Задаём менеджер отображения для этой панели

      FlowLayout fl = new FlowLayout();
      windowContent.setLayout(fl);

  // Создаём компоненты в памяти

      JLabel label1 = new JLabel("Number 1:");
      JTextField field1 = new JTextField(10);
      JLabel label2 = new JLabel("Number 2:");
      JTextField field2 = new JTextField(10);
      JLabel label3 = new JLabel("Sum:");
      JTextField result = new JTextField(10);
      JButton go = new JButton("Add");

  // Добавляем компоненты на панель

      windowContent.add(label1);
      windowContent.add(field1);
      windowContent.add(label2);
      windowContent.add(field2);
      windowContent.add(label3);
      windowContent.add(result);
      windowContent.add(go);

  // Создаём фрейм и задаём для него панель

      JFrame frame = new JFrame("My First Calculator");
      frame.setContentPane(windowContent);

  // задаём и размер и делаем фрейм видимым

      frame.setSize(400,100);
      frame.setVisible(true);

 }
}
```

Скомпилируйте и запустите эту программу. Должно появиться окно следующего вида:



Это, может быть, не самый красивый в мире калькулятор, но в данном примере видно, как добавлять компоненты и отображать окно. В следующей секции мы попробуем сделать более красивый интерфейс с помощью схем размещения.

Схемы Размещения

В некоторых старомодных языках программирования необходимо было указывать координаты и размеры каждого компонента окна. Это работало хорошо, если было известно разрешающая способность экрана каждого пользователя. Кстати, людей, которые пользуются программами, называют пользователями (users). В Java есть схемы размещения (Layout Managers), которые позволяют разместить компоненты на экране, не зная точных позиций компонентов. Схемы гарантируют, что та часть интерфейса, за которую они отвечают, будет выглядеть правильно вне зависимости от размеров окна и разрешения экрана.

Swing предоставляет следующие схемы:

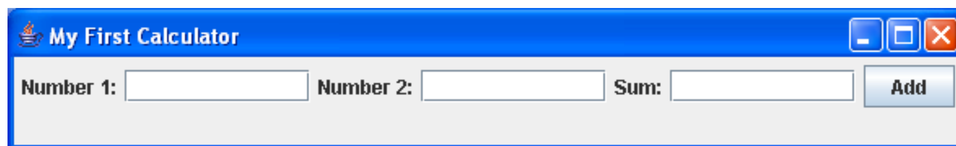
- `FlowLayout`
- `GridLayout`
- `BoxLayout`
- `BorderLayout`
- `CardLayout`
- `GridBagLayout`

Чтобы использовать любой из этих менеджеров, необходимо создать его экземпляр и затем назначить этот объект какому-нибудь

контейнеру (container), например панели, как это было в примере с SimpleCalculator.

FlowLayout - построчное расположение

По этой схеме компоненты размещаются в окне (или другом контейнере) строка за строкой. Например, текстовые метки, иконки, текстовые поля и кнопки будут добавляться в первую условную строку, пока в ней есть место. Когда первая строка заполнится, оставшиеся компоненты будут добавляться в следующую строку и так далее. Если пользователь изменит размер окна, картина может измениться. Просто потяните за угол калькулятора, чтобы поменять его размер. Посмотрите как `java.awt.FlowLayout` переупорядочивает элементы окна во время изменения его размеров.



В следующем примере кода, ключевое слово this представляет экземпляр класса SimpleCalculator.

```
FlowLayout fl = new FlowLayout();
this.setLayoutManager(fl);
```

Согласен, `FlowLayout` не лучшим образом подходит для нашего калькулятора. Давайте теперь попробуем что-нибудь другое.

GridLayout - табличное расположение

Класс `java.awt.GridLayout` позволяет организовать компоненты, как *строки* и *столбцы* в таблице. Компоненты будут добавляться в ячейки условной таблицы. Если размер окна будет увеличен, ячейки станут больше, но положение компонентов относительно друг друга останется прежним. В нашем калькуляторе семь компонентов – три текстовые метки, три текстовых поля и кнопка. Мы можем разместить их в таблице с четырьмя строками и двумя колонками (одна ячейка останется пустой):

```
GridLayout gr = new GridLayout(4,2);
```

Также можно задать расстояние между ячейками по вертикали и горизонтали, например в пять пикселей:

```
GridLayout gr = new GridLayout(4,2,5,5);
```

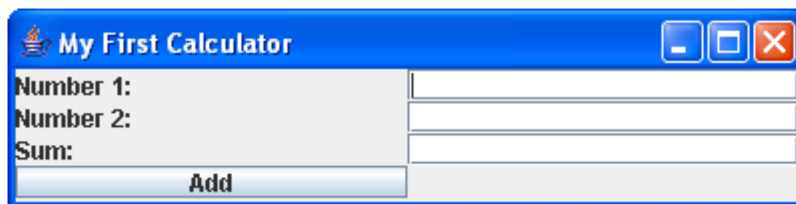
После небольших изменений в нашем калькуляторе (они подсвечены ниже), он станет выглядеть гораздо симпатичнее.

А теперь создайте и скомпилируйте новый класс SimpleCalculatorGrid в проекте *My Calculator*.

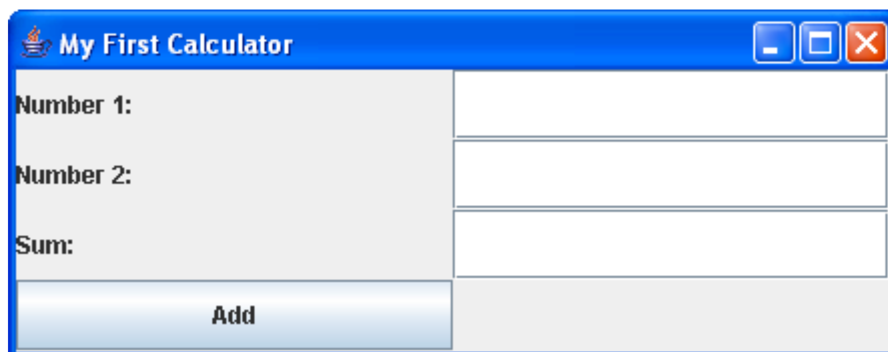
```
import javax.swing.*;
import java.awt.GridLayout;

public class SimpleCalculatorGrid {

 public static void main(String[] args) {

  // Создаём панель

      JPanel windowContent= new JPanel();

  // Задаём менеджер расположения для этой панели

      GridLayout gl = new GridLayout(4,2);

      windowContent.setLayout(gl);

 // Создаём компоненты в памяти

      JLabel label1 = new JLabel("Number 1:");

      JTextField field1 = new JTextField(10);

      JLabel label2 = new JLabel("Number 2:");

      JTextField field2 = new JTextField(10);

      JLabel label3 = new JLabel("Sum:");

      JTextField result = new JTextField(10);

      JButton go = new JButton("Add");

 // Добавляем компоненты в панель

      windowContent.add(label1);
      windowContent.add(field1);
      windowContent.add(label2);
      windowContent.add(field2);
      windowContent.add(label3);
      windowContent.add(result);
      windowContent.add(go);
```

```
  // Создаём фрейм и задаём панель для него
     JFrame frame = new JFrame("My First Calculator");

     frame.setContentPane(windowContent);



 // задаём размер и отображаем окно

      frame.setSize(400,100);
      frame.setVisible(true);

 }
}
```

После запуска программы SimpleCalculatorGrid, вы увидите такое окно:



Попробуйте поменять размеры этого окна – размеры элементов управления будут меняться вместе с ним, но их положение относительно друг друга не изменится:



Ещё одна важная вещь, которую стоит запомнить про табличный компоновщик – в нем все ячейки имеют одинаковую длину и ширину.

BorderLayout - размещение по областям

Класс `java.awt.BorderLayout` разделяет окно на Южную, Западную, Северную, Восточную и Центральную области. Северная область находится наверху окна, Южная – снизу, Западная – слева, а Восточная – справа. Например, в калькуляторе, который будет

продемонстрирован на следующей странице, текстовые поля, которые отображают числа, находятся в Северной области.

Создать BorderLayout и поместить в него текстовые поля можно следующим образом:

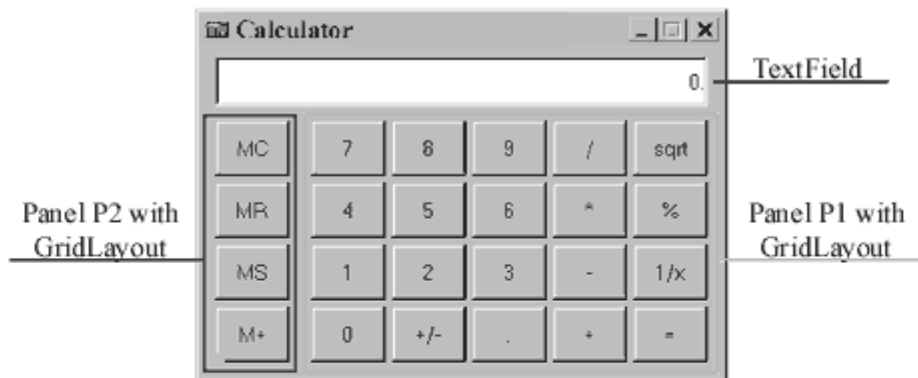
```
BorderLayout bl = new BorderLayout();
this.setLayoutManager(bl);

JTextField  txtDisplay = new JTextField(20);
this.add("North",   txtDisplay);
```

Совсем не обязательно помещать элементы управления во все пять областей. Если необходимы только Северная, Центральная и Южная области, то Центральная станет шире, т.к. Западная и Восточная пустуют.

Я буду использовать BorderLayout чуть позже, в следующей версии нашего калькулятора, который будет называться Calculator.java.

Комбинирование схем размещения

Как вы думаете, можно ли с помощью GridLayout создать калькулятор, который будет выглядеть так же, как стандартный калькулятор в Microsoft Windows?



К сожалению, нет, так как ячейки этого калькулятора имеют разные размеры – текстовое поле больше кнопок. Но содержимое окна можно представить с помощью нескольких панелей, у которых схемы разные.

Попробуем использовать комбинацию нескольких схем в новом калькуляторе. Для этого необходимо выполнить следующие шаги:

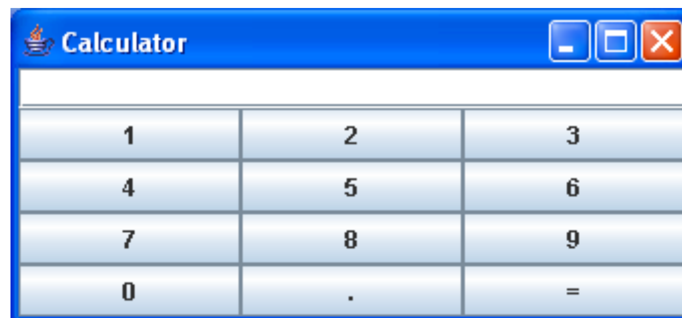
- ✓ Назначить BorderLayout панели фрейма, которая будет основной, и в которой будут содержаться остальные панели.

- ✓ Добавить JTextField в северную часть, для того чтобы отображать введённые числа.

- ✓ Создать панель p1 с `GridLayout`, добавить на неё 20 кнопок и затем поместить эту панель в центральную область основной панели.

- ✓ Создать панель p2 с `GridLayout`, добавить на неё четыре кнопки и затем поместить панель p2 в западную область основной панели.

Давайте начнём с более простой версии калькулятора, которая будет выглядеть вот так:



Создайте новый класс Calculator и запустите программу. Чтобы понять, как она работает, прочитайте комментарии в примере кода, продемонстрированном ниже.

```
import javax.swing.*;
import java.awt.GridLayout;
import java.awt.BorderLayout;
public class Calculator {

// Объявление всех компонентов калькулятора.
JPanel windowContent;
JTextField displayField;
JButton button0;
JButton button1;
JButton button2;
JButton button3;
JButton button4;
JButton button5;
JButton button6;
JButton button7;
```

```
JButton button8;
JButton button9;
JButton buttonPoint;

JButton buttonEqual;
JPanel p1;

      // В конструкторе создаются все компоненты
      // и добавляются на фрейм с помощью комбинации
      // Borderlayout и Gridlayout
      Calculator(){

        windowContent= new JPanel();

      // Задаём схему для этой панели
      BorderLayout bl = new BorderLayout();
      windowContent.setLayout(bl);

      // Создаём и отображаем поле
      // Добавляем его в Северную область окна

      displayField = new JTextField(30);
      windowContent.add("North",displayField);

     // Создаём кнопки, используя конструктор
     // класса JButton, который принимает текст
     // кнопки в качестве параметра

     button0=new JButton("0");
     button1=new JButton("1");
     button2=new JButton("2");
     button3=new JButton("3");
     button4=new JButton("4");
     button5=new JButton("5");
     button6=new JButton("6");
     button7=new JButton("7");
     button8=new JButton("8");
     button9=new JButton("9");
     buttonPoint = new JButton(".");
     buttonEqual=new JButton("=");

   // Создаём панель с GridLayout
   // которая содержит 12 кнопок - 10 кнопок с числами
   // и кнопки с точкой и знаком равно

     p1 = new JPanel();
     GridLayout gl =new GridLayout(4,3);
     p1.setLayout(gl);

   //  Добавляем кнопки на панель p1

      p1.add(button1);
      p1.add(button2);
      p1.add(button3);
      p1.add(button4);
      p1.add(button5);
      p1.add(button6);
      p1.add(button7);
```

```
      p1.add(button8);
      p1.add(button9);
      p1.add(button0);
      p1.add(buttonPoint);
      p1.add(buttonEqual);

   // Помещаем панель p1 в центральную область окна
      windowContent.add("Center",p1);

   //Создаём фрейм и задаём его основную панель
     JFrame frame = new JFrame("Calculator");
     frame.setContentPane(windowContent);

   // делаем размер окна достаточным
   // для того, чтобы вместить все компоненты
     frame.pack();

   // Наконец, отображаем окно
     frame.setVisible(true);

 }

  public static void main(String[] args) {

            Calculator calc = new Calculator();

  }
}
```

BoxLayout - расположение по горизонтали или вертикали

Класс `java.swing.BoxLayout` располагает несколько компонентов окна горизонтально (по оси абсцисс) или вертикально (по оси ординат). В отличие от менеджера `FlowLayout`, когда окно с `BoxLayout` меняет свой размер, его элементы управления не смещаются со своих позиций. `BoxLayout` позволяет элементам окна иметь разные размеры (чего не позволяет `GridLayout`).

Следующие две строки кода задают box layout с вертикальным выравниванием на `JPanel`.

```
JPanel p1= new JPanel();

setLayout(new BoxLayout(p1, BoxLayout.Y_AXIS));
```

Чтобы сделать этот код короче, я не создавал переменную для хранения ссылки на объект `BoxLayout`, вместо этого я создал экземпляр этого класса и сразу же передал его, как аргумент в метод `setLayout()`.

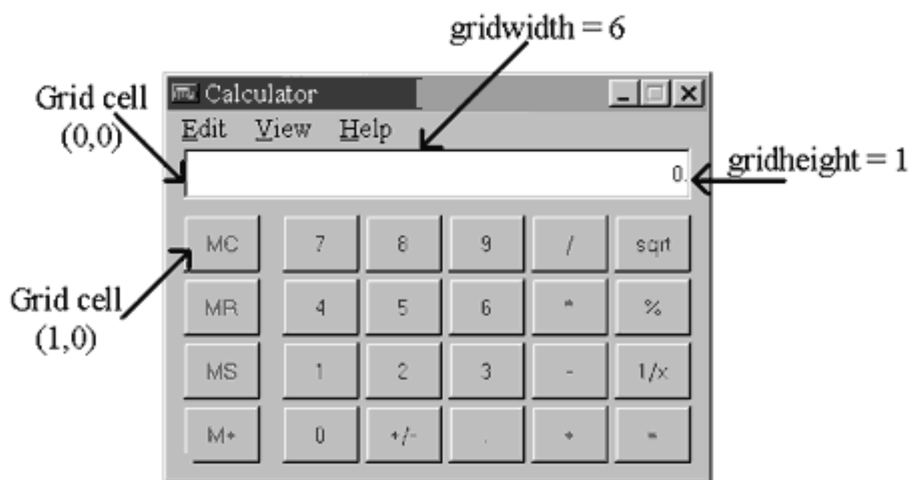
GridBag Layout - более гибкое табличное расположение

А сейчас я покажу ещё один способ создания окна калькулятора. Здесь будет использоваться `java.awt.GridBagLayout` вместо комбинации схем и панелей.

В нашем калькуляторе есть строки и столбцы, но в `GridLayout` они обязаны иметь одинаковые размеры. Это не подходит, так как у нас есть поле для ввода, ширина которого равна ширине трёх кнопок с числами.

`GridBagLayout` - более продвинутая схема размещения. Она позволяет задавать размер ячейки, равным нескольким клеткам таблицы. `GridBagLayout` имеет вспомогательный класс, который называется `GridBagConstraints` (ограничения на клетки таблицы). Эти ограничения не что иное, как атрибуты ячеек, которые необходимо задавать для каждой ячейки таблицы отдельно. Все ограничения должны быть заданы *до того,* как в ячейку помещаются компоненты. Например, один из атрибутов `GridBagConstraints` называется `gridWidth`. Он позволяет задать ширину какой-то одной ячейки, равной ширине нескольких других.

Во время работы с `GridBagLayout` необходимо сначала создать экземпляр класса `GridBagConstraints`, и затем задать значения для его свойств. После того как это сделано, можно добавлять объект в ячейку контейнера.



Следующий пример кода, усыпан комментариями, которые помогут понять, как использовать GridBagLayout.

```
 // Задаём GridBagLayout для панели окна
 GridBagLayout gb = new GridBagLayout();
 this.setLayout(gb);

// Создаём экземпляр класса GridBagConstraints
// Эти строки кода нужно повторить для каждой компоненты
// которая добавляется в ячейку

 GridBagConstraints constr = new GridBagConstraints();

//задаём ограничения для строки ввода калькулятора
// координата x в таблице
constr.x=0;

// координата y в таблице
constr.y=0;

// эта ячейка имеет такую же высоту, как стандартные ячейки
constr.gridheight =1;

// эта ячейка имеет ширину равную ширине 6 стандартных ячеек
constr.gridwidth= 6;

// заполняем всё пространство ячейки
constr.fill= constr.BOTH;

// пропорция по горизонтали, которую будет заниматькомпонент
constr.weightx = 1.0;

// пропорция по вертикали, которую будет занимать компонент
constr.weighty = 1.0;

// позиция компонента внутри ячейки
constr.anchor=constr.CENTER;

displayField = new JTextField();

// устанавливаем ограничения для поля ввода
gb.setConstraints(displayField,constr);

// добавляем поле ввода в окно
windowContent.add(displayField);
```

CardLayout – колода карт

Представьте колоду карт, в которой карты лежат рубашкой вниз так, что вы можете видеть только верхнюю карту. Схема java.awt.CardLayout может быть использована, если необходимо создать компонент, который выглядит, как папка с вкладками.

При нажатии на вкладку содержимое экрана меняется. На самом деле, все панели, необходимые для этого окна, уже предварительно загружены и лежат друг на друге. Когда пользователь щелкает по вкладке, программа просто "переносит эту вкладку" наверх и делает содержимое остальных вкладок невидимыми.

Скорее всего, вы не будете использовать эту схему, потому что библиотека Swing включает готовый компонент для окон с вкладками. Он называется `JTabbedPane`.

Можно ли создавать окна, не используя схемы?

Конечно, можно! Ничто не мешает явно задавать координаты каждого компонента при добавлении на окно. Для этого класс должен чётко указать, что он не использует схемы размещения. В Java есть специальное слово null, которое означает “значение не задано”. Мы будем использовать это ключевое слово довольно часто в будущем, и в следующем примере оно означает, что никакая схема не используется:

```
windowContent.setLayout(null);
```

Но, если отказаться от схем, то необходимо назначить координаты левого верхнего угла, ширину и высоту каждого оконного компонента. В следующем примере показано, как можно установить ширину кнопки в 40 пикселей, высоту в 20, и разместить её на 100 пикселей вправо и 200 пикселей вниз от верхнего левого угла окна:

```
JButton myButton = new Button("New Game");
myButton.setBounds(100,200,40,20);
```

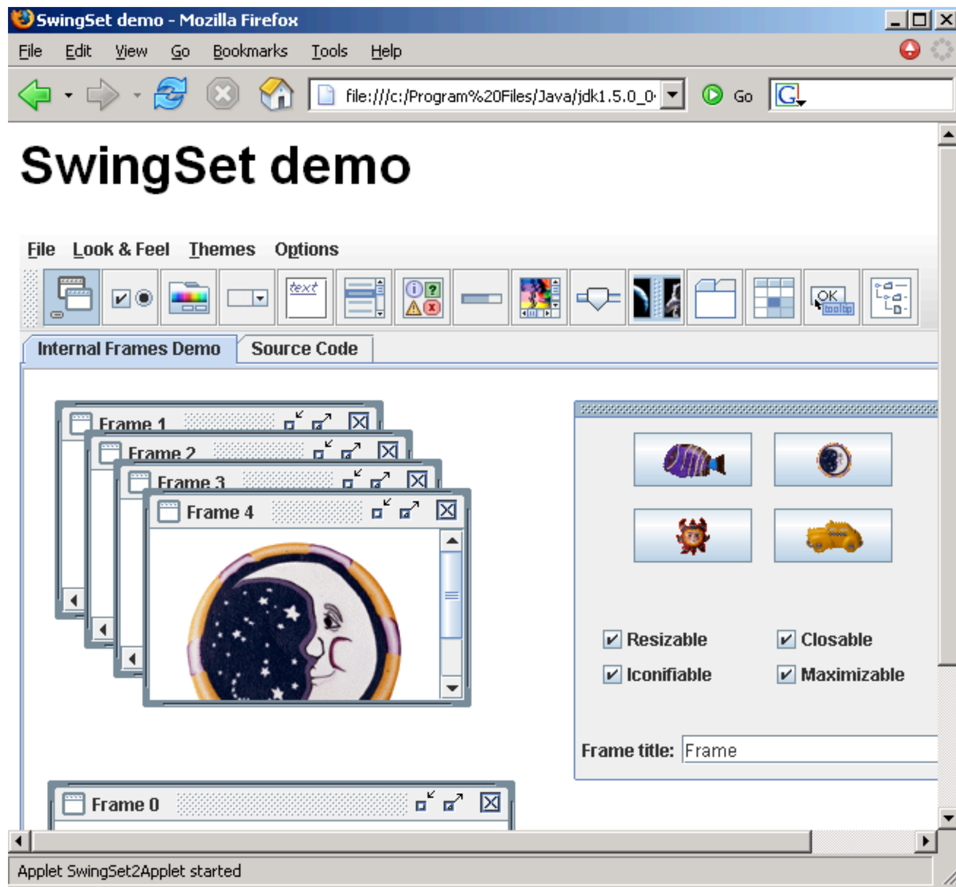
Компоненты окна

Я не буду описывать все компоненты Swing в этой книге, но вы можете найти ссылку на онлайн учебник по Swing, в разделе *Материалы для дополнительного чтения*. В этом руководстве есть подробные описания всех компонентов Swing. Наш калькулятор использует только `JButton`, `JLabel` и `JTextField`. Вот список других доступных компонент:

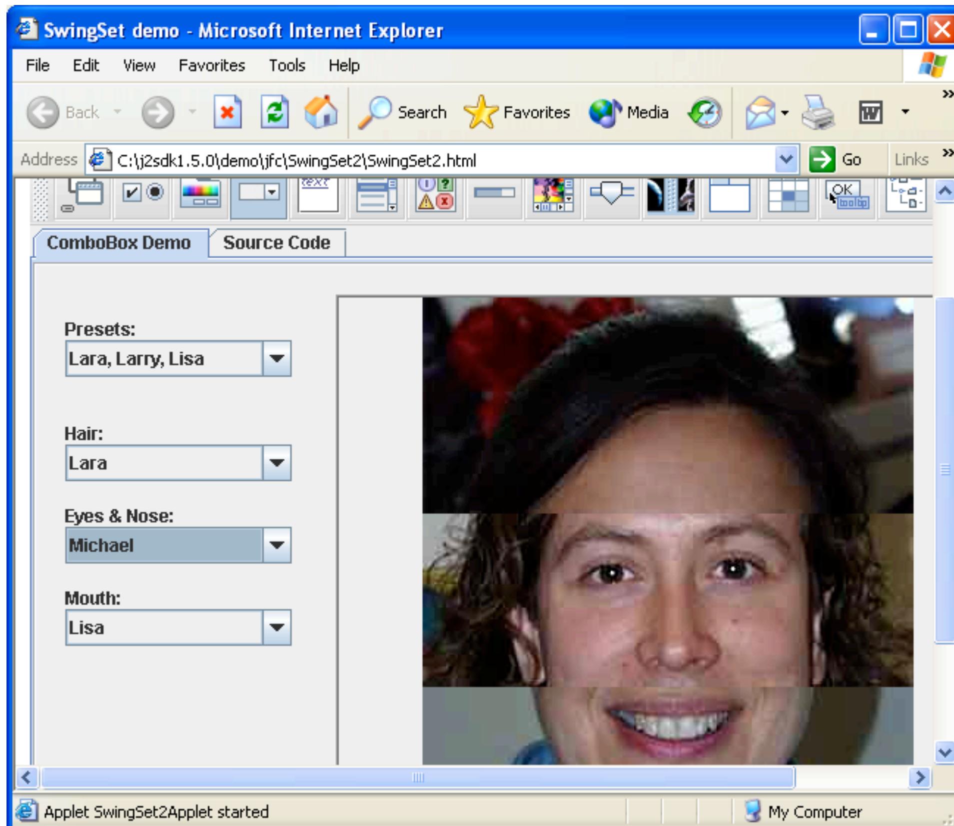
- ✓ JButton
- ✓ JLabel
- ✓ JCheckBox
- ✓ JRadioButton
- ✓ JToggleButton
- ✓ JScrollPane
- ✓ JSpinner
- ✓ JTextField
- ✓ JTextArea
- ✓ JPasswordField
- ✓ JFormattedTextField
- ✓ JEditorPane
- ✓ JScrollBar
- ✓ JSlider
- ✓ JProgressBar
- ✓ JComboBox
- ✓ JList
- ✓ JTabbedPane
- ✓ JTable
- ✓ JToolTip
- ✓ JTree
- ✓ JViewPort
- ✓ ImageIcon

Вы также можете создавать меню (JMenu и JPopupMenu), всплывающие окна, фреймы внутри других фреймов (JInternalFrame) и использовать стандартные окна (JFileChooser, JColorChooser и JOptionPane).

Java поставляется с отличным демонстрационным приложением, которое показывает все доступные компоненты Swing в действии. Оно находится в папке, где вы установили Java SDK, например *C:\Program Files\Java\jdk1.6.0_19\demo\jfc\SwingSet3*. Еще его можно загрузить вот отсюда: https://swingset3.dev.java.net. Просто откройте файл SwingSet3.html, и вы увидите экран, похожий на вот этот:



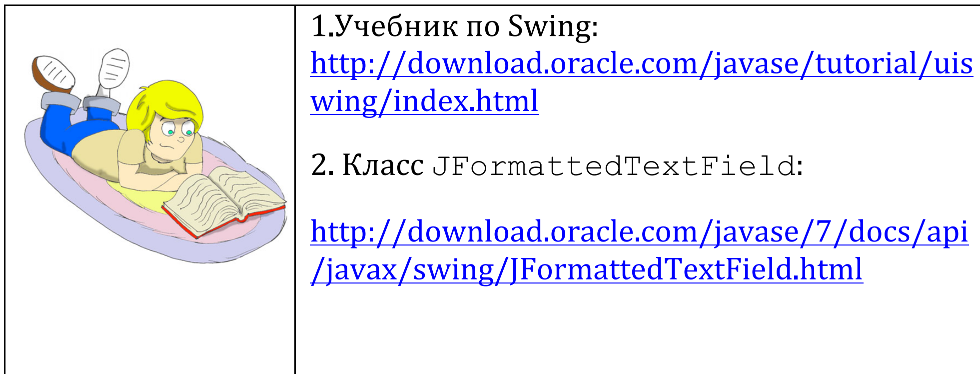
Нажмите на любое изображение на панели инструментов, чтобы увидеть, как та или иная компонента Swing работает. Вы также можете найти примеры кода, который использовался для создания каждого окна, выбрав вкладку Source Code. Например, если нажмёте на иконку combobox (выпадающий список), то увидите окно, которое выглядит следующим образом:



В Swing есть много различных компонентов, чтобы сделать ваши окна симпатичными. В этой главе мы создавали Swing компоненты, просто вводя код, без использования специальных инструментов. Но есть специальные утилиты, которые позволяют выбрать компонент на панели инструментов и перетащить его в создаваемое окно. Эти приложения автоматически генерируют соответствующий Java код для компонентов Swing. Один из таких графических дизайнеров, который позволяет легко создавать Swing приложения, называется Matisse. Другая – Gigloo GUI Builder.

В следующей главе будет рассказано, как окно может реагировать на действия пользователя.

Материалы для дополнительного чтения



1.Учебник по Swing: http://download.oracle.com/javase/tutorial/uiswing/index.html

2. Класс `JFormattedTextField`:

http://download.oracle.com/javase/7/docs/api/javax/swing/JFormattedTextField.html

Практические упражнения

1.Модифицируйте класс Calculator.java добавив в него кнопки +, -, /, и *. Поместите эти кнопки на панель p2, и положите эту панель на Восточную область основной панели.

2. Прочитайте про класс `JFormattedTextField` в интернете и измените исходный код калькулятора так, чтобы этот класс использовался вместо `JTextField`. Целью является создание поля ввода с выравниванием по правому краю, как в настоящих калькуляторах.

Практические упражнения для умников и умниц

Модифицируйте класс Calculator.java так, чтобы все кнопки с цифрами хранились в массиве с десятью элементами, который должен быть объявлен вот так:

```
Buttons[] numButtons= new Buttons[10];
```

Замените 10 строк кода, которые начинаются с `button0=new JButton("0");` циклом, который создаёт кнопки и добавляет их в массив.

Подсказка: загляните в исходный код игры Крестики-Нолики в главе 7.

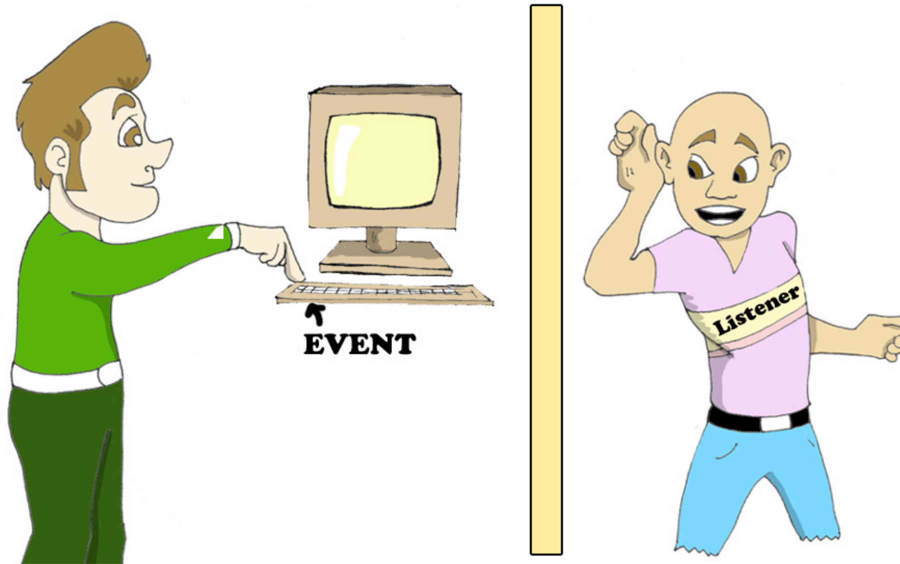
Глава 6. События окна

Во время работы программы могут происходить различные события: пользователь нажмет на кнопку, веб-браузер решит перерисовать окно, и так далее. Я уверен, что вы пытались нажимать на кнопки вашего калькулятора из главы 5, но эти кнопки еще не были готовы реагировать на ваши действия.

Каждый компонент окна может обрабатывать различные события, или, как мы говорим, *слушать* эти события. Вы можете зарегистрировать классы Java, которые называют *слушателями (listeners)*, привязав их к компонентам окна. Вы должны сделать так, чтобы компоненты слушали только те события, которые им нужны.

Например, когда человек перемещает указатель мышки над кнопкой калькулятора, неважно где именно был этот указатель, когда пользователь нажал на кнопку, пока курсор находился над поверхностью кнопки. Поэтому вам не нужно регистрировать слушатель MouseMotionListener для кнопки. С другой стороны, этот слушатель полезен для всевозможных программ для рисования.

Для кнопок калькулятора нужно зарегистрировать класс ActionListener, который умеет обрабатывать нажатия на кнопки. Все эти слушатели – это специальные конструкции Java, которые называются *интерфейсами*.



Интерфейсы

Большинство классов определяют методы, которые реагируют на различные действия, например, *отвечу на нажатие кнопки*, *отвечу на движение мыши*, и так далее. Набор таких действий называется *поведением класса*.

Интерфейсы – это специальные конструкции, которые только объявляют набор определенных действий без кода, который описывает, что именно надо делать в объявленных методах, например:

```
interface MouseMotionListener {

    void mouseDragged(MouseEvent e);
    void mouseMoved(MouseEvent e);

}
```

Как видите, методы `mouseDragged()` и `mouseMoved()` не содержат никакого текста программ – эти методы просто объявлены в интерфейсе, называемом MouseMotionListener. А вот если ваш класс должен реагировать на движение указателя мыши или на перетаскивание мышью, то тогда он должен *реализовать* этот интерфейс.

Слово `implements` означает, что этот класс совершенно точно будет содержать методы, которые могли быть объявлены в интерфейсе, например:

```
import java.awt.event.MouseMotionListener;

class MyDrawingPad implements MouseMotionListener{

     // здесь может идти  текст программы, которая
    //  выполняет функции графического редактора
    mouseDragged(MouseEvent e){

       // здесь будет текст программы, когда
       // мышь что-то перетаскивает

     }

     mouseMoved(MouseEvent e){

       // сюда идет текст программы, когда
       // мышь просто будет здесь двигается

     }
}
```

Должно быть, вам интересно, зачем беспокоиться о создании интерфейсов без текста программы? Причина в том, что интерфейс, сделанный однажды, может использоваться во многих классах. Например, когда другие классы (или сама виртуальная машина JVM) видят, что класс MyDrawingPad реализует интерфейс `MouseMotionListener`, они знают, что в этом классе точно есть методы `mouseDragged()` и `mouseMoved()`.

Каждый раз, когда пользователь двигает мышкой, JVM вызывает метод `mouseMoved()` и исполняет текст программы, который вы там написали. Представьте, что если Иван решит назвать этот метод `mouseMoved()`, Маша назовет его movedMouse(), а Петя предпочтет mouseCrawling()? Тогда JVM запутается и не будет знать, какой же метод вашего класса вызвать, чтобы сообщить о движении мыши.

Класс Java может реализовывать много интерфейсов, например, он может реагировать на движения мыши и на нажатие кнопки:

```
class myDrawingProgram implements MouseMotionListener,

                                                ActionListener {

   // Здесь вы должны написать текст программы для
   // каждого метода обьявленного в обоих интерфейсах

}
```

После того, как вы освоитесь с интерфейсами, которые предоставляет вам Java, вы сможете создавать свои собственные интерфейсы, но это темы посложнее и, пока что, мы не будем этого делать.

Слушатель по имени ActionListener

Давайте вернемся к нашему калькулятору. Если вы сделали задания к предыдущей главе, визуальная часть программы готова. Теперь мы создадим еще один класс-слушатель, который будет что-то делать, когда пользователь будет нажимать на одну из кнопок. Вообще-то, мы могли бы добавить текст программы, обрабатывающий события нажатия на кнопку, сразу в класс Calculator.java, но лучше не смешивать в одном классе визуальную и обрабатывающую части.

Назовем второй класс CalculatorEngine, и скажем, что он должен реализовать интерфейс `java.awt.ActionListener` в котором объявлен только один метод - `actionPerformed(ActionEvent)`. JVM вызывает этот метод в классе, который реализует этот интерфейс каждый раз, когда кто-то нажимает на кнопку.

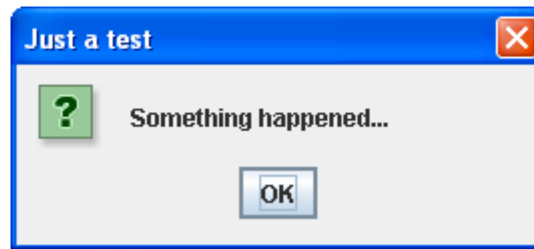
Посмотрим на этот простой класс:

```
import java.awt.event.ActionListener;

public class CalculatorEngine implements ActionListener {

}
```

Если вы попытаетесь его скомпилировать (или просто сохранить его в Eclipse), то возникнет сообщение об ошибке, что, мол, класс должен реализовать метод `actionPerformed(ActionEvent e)`. Давайте исправим эту ошибку:

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class CalculatorEngine implements ActionListener {

   public void actionPerformed(ActionEvent e){

      // Если этот метод можно оставить пустым, ничего не
      // произойдет, когда JVM вызовет его
   }
}
```

Следующая версия этого класса будет открывать *окно сообщения (a message box)* из метода `actionPerformed()`. С помощью класса `JOptionPane` и его метода `showConfirmDialog()` можно показывать пользователю любые сообщения. Например, класс CalculatorEngine может выдать следующее:



Есть разные версии метода showConfirmDialog(), мы будем использовать версию с четырьмя параметрами. В тексте программы ниже null означает, что окно сообщения не имеет родительского окна, второй аргумент – это заголовок окна сообщения, потом идет само сообщение, а четвертый аргумент позволяет выбрать, какие кнопки будут отображаться в окне сообщения (PLAIN_MESSAGE в следующем примере означает, что будет отображаться только одна кнопка - “OK”).

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JOptionPane;

public class CalculatorEngine implements ActionListener {

   public void actionPerformed(ActionEvent e){

      JOptionPane.showConfirmDialog(null,
           "Something happened...", "Just a test",
                   JOptionPane.PLAIN_MESSAGE);

   }
}
```

Теперь я покажу, как скомпилировать и запустить следующую версию нашего калькулятора, которая показывает окно сообщения *“Something happened”.*

Регистрация компонентов с ActionListener

Кто и когда будет вызывать код, написанный в методе `actionPerformed()`? Сама JVM вызовет этот метод, *если вы зарегистрируете* класс CalculatorEngine в кнопках калькулятора (или свяжете их с классом)! Просто добавьте эти две строки в конец конструктора класса Calculator , чтобы зарегистрировать наш слушатель для кнопки “Ноль”:

```
CalculatorEngine calcEngine = new CalculatorEngine();
button0.addActionListener(calcEngine);
```

Теперь каждый раз, когда пользователь нажмет кнопку button0, JVM вызовет метод `actionPerformed()` у объекта CalculatorEngine. Скомпилируйте и запустите класс Calculator и нажмите на кнопку "Ноль" – и на экране появится окно сообщения "*Something happened*"! Другие кнопки пока не реагируют, потому что в них не зарегистрирован наш слушатель. Добавьте такие же строки, чтобы оживить и другие кнопки:

```
button1.addActionListener(calcEngine);
button2.addActionListener(calcEngine);
button3.addActionListener(calcEngine);
button4.addActionListener(calcEngine);
```

и т д.

Из-за кого событие-то?

Следующий шаг – сделать нашего слушателя немного умнее – он будет показывать различные сообщения, в зависимости от того, какая кнопка была нажата. Когда произойдет *событие*, JVM вызовет метод вашего класса-слушателя `actionPerformed(ActionEvent)`, и передаст ему необходимую информацию о событии в аргументе `ActionEvent`. Вы можете получить эту информацию, вызывая соответствующие методы этого объекта-аргумента.

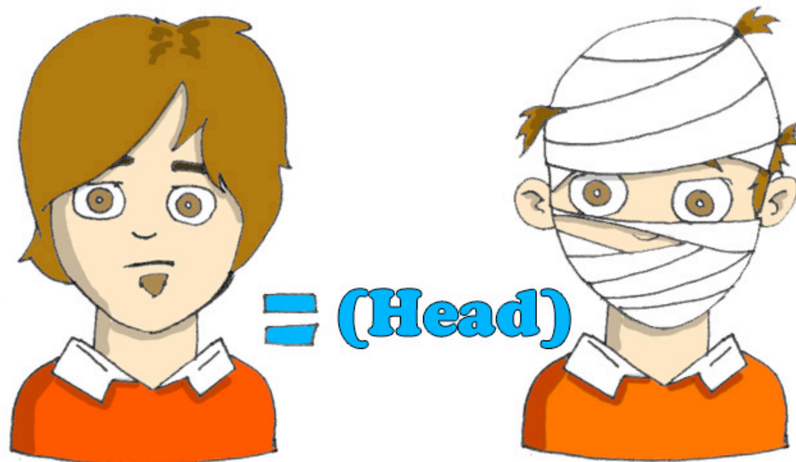
Приведение типов - casting

В следующем примере я покажу, как определить, какая кнопка была нажата, вызывая метод `getSource()` класса `ActionEvent`. Переменная evt – это ссылка на объект-событие, который живет где-то в памяти компьютера. Но, как написано в документации Java, метод `getSource()` возвращает источник события как экземпляр типа `Object`, который является предком всех классов Java, включая компоненты окна.

Так сделано для того, чтобы этот метод был универсальным, и работал для любых компонент. Но мы-то знаем наверняка, что в нашем окне единственной причиной такого события могут быть только кнопки! Поэтому мы *приводим тип (we cast the type)* возвращаемого Object к типу `JButton`, размещая тип в скобках перед именем метода:

```
JButton clickedButton =  (JButton) evt.getSource();
```

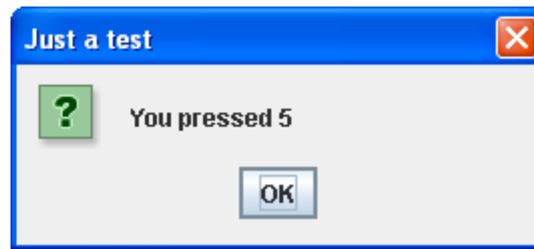
Слева от знака равенства объявлена переменная типа `JButton` и, хотя метод `getSource()` возвращает данные типа `Object`, мы будто говорим JVM: *Не волнуйся, я знаю наверняка, что это – экземпляр JButton*.



Только после приведения типа `Object` к `JButton` нам разрешается вызвать метод `getText()`, который принадлежит классу `JButton`.

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JOptionPane;
import javax.swing.JButton;

public class CalculatorEngine implements ActionListener {

   public void actionPerformed(ActionEvent e){

      // Получаем источник события
      JButton clickedButton= (JButton)e.getSource();

      // Получаем надпись на кнопке
      String clickedButtonLabel = clickedButton.getText();

      // Добавляем надпись на кнопке к тексту окна сообщения
      JOptionPane.showConfirmDialog(null,"You pressed " +
          clickedButtonLabel,
          "Just a test", JOptionPane.PLAIN_MESSAGE);
   }
}
```

Например, если вы нажмете кнопку “Пять”, то увидите это окно сообщения:



Но что, если события окна будут вызываться не только кнопками, но и какими-нибудь другими компонентами? Не всегда нужно приводить каждый объект к типу JButton! Для этих случаев вы должны использовать специальный оператор Java, называемый instanceof , чтобы правильно сделать приведение типа. Следующий пример сначала проверяет, объект какого типа вызвал событие, а потом делает приведение типа к JButton или к JTextField:

```
public void actionPerformed(ActionEvent evt){

 JTextField myDisplayField=null;
 JButton clickedButton=null;

 Object eventSource = evt.hetSource();

 if (eventSource instanceof JButton){
     clickedButton = (JButton) eventSource;

 }else if (eventSource instanceof JTextField){
      myDisplayField = (JTextField)eventSource;

 }
}
```

Нашему калькулятору нужно исполнять разные части программы для разных кнопок, и следующий пример показывает, как это сделать.

```
public void actionPerformed(ActionEvent e){

     Object src = e.getSource();

     if (src == buttonPlus){
      //Здесь должен быть текст программы,складывающий числа

    } else if (src == buttonMinus){
      // Здесь должен быть текст программы, вычитающий числа

    }else if (src == buttonDivide){
      // Здесь - деление чисел

    } else if (src == buttonMultiply){
      // Здесь – текст программы, умножающий числа
    }
}
```

Как передавать данные между классами

Вообще-то, когда вы нажимаете кнопку с цифрой на настоящем калькуляторе, он не показывает окно сообщения, а показывает эту цифру на своем дисплее. Возникает задача – нам нужно получить доступ к полю displayField класса Calculator из метода `actionPerformed()` класса CalculatorEngine. Это можно сделать, создав в классе CalculatorEngine переменную, которая будет хранить *ссылку на экземпляр* объекта Calculator.

В следующей версии класса CalculatorEngine мы добавим конструктор. У этого конструктора будет один аргумент типа Calculator. Не удивляйтесь, аргументы у методов могут иметь тип классов, созданных вами!

JVM исполняет конструктор класса CalculatorEngine во время создания этого экземпляра в памяти. Класс Calculator создает CalculatorEngine, и передает его конструктору *ссылку на себя*:

```
CalculatorEngine calcEngine = new CalculatorEngine(this);
```

Эта ссылка указывает на то место в памяти, где находится экземпляр класса Calculator. Конструктор CalculatorEngine сохраняет это значение в переменной parent, чтобы потом использовать его в методе `actionPerformed()` для доступа к дисплею калькулятора.

```
parent.displayField.getText();

…

parent.displayField.setText(dispFieldText + clickedButtonLabel);
```

Эти две строчки были взяты из следующего примера:

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JButton;

public class CalculatorEngine implements ActionListener {

  Calculator parent; // ссылка на Calculator

  // Конструктор сохраняет ссылку на окно калькулятора
  // в переменной класса "parent"

  CalculatorEngine(Calculator parent){

    this.parent = parent;
```

```
   }

   public void actionPerformed(ActionEvent e){

     // Получить источник текущего действия
       JButton clickedButton =  (JButton) e.getSource();

    // Получить текущий текст из поля вывода ("дисплея")
    // калькулятора
       String dispFieldText = parent.displayField.getText();

     // Получить надпись на кнопке
       String clickedButtonLabel = clickedButton.getText();

       parent.displayField.setText(dispFieldText +
                                   clickedButtonLabel);

    }
  }
```

Когда вы объявляете переменную для хранения ссылки на экземпляр какого-либо класса, эта переменная должна иметь, либо тип этого класса, либо тип одного из его суперклассов.

Любой класс в Java происходит от класса `Object`, и если, например, класс Fish – это наследник класса Pet, то каждая из этих строк правильная:

```
Fish myFish   = new Fish();
Pet  myFish   = new Fish();

Object myFish = new Fish();
```

Доделываем калькулятор

Давайте определимся с несколькими правилами (с *алгоритмом*), по которым должен работать наш калькулятор:

1. Сначала пользователь вводит все цифры первого числа.

2. Если пользователь нажмет какую-нибудь из кнопок арифметического действия +, -, / or * , то надо сохранить первое число и выбранное действие в полях класса, и стереть число из дисплея калькулятора.

3. Потом пользователь вводит второе число и нажимает кнопку *равно* .

4. Сконвертировать строковое значение из дисплея в числовой тип double, чтобы иметь возможность хранить большие дробные числа. Произвести арифметическое действие с помощью выбранного действия и первого числа, сохраненных в шаге 2.

5. Показать результат шага 4 на дисплее калькулятора и сохранить это значение в переменной, которая использовалась в шаге 2.

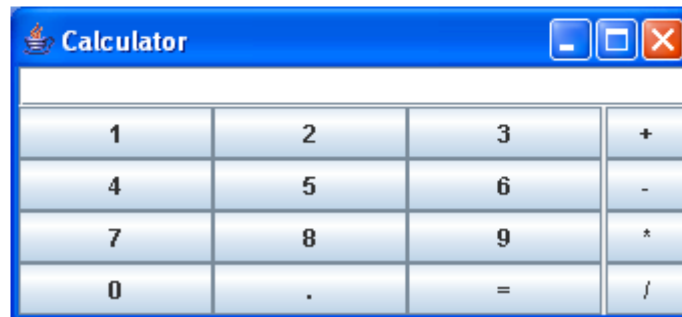
Все эти шаги мы запрограммируем в классе CalculatorEngine. Пока вы будете читать следующий текст программы, помните, что метод `actionPerformed()` будет вызываться после каждого нажатия на кнопку и данные между вызовами этого метода будут храниться в переменных selectedAction и currentResult.

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JButton;

public class CalculatorEngine implements ActionListener {

 Calculator parent;  //ссылка на окно калькулятора

 char selectedAction = ' ';  // +, -, /, или *
 double currentResult =0;

// Конструктор сохраняет ссылку на окно калькулятора
// в переменной экземпляра класса

 CalculatorEngine(Calculator parent){

   this.parent = parent;

 }

 public void actionPerformed(ActionEvent e){

   // Получить источник действия

   JButton clickedButton =  (JButton) e.getSource();

   String dispFieldText=parent.displayField.getText();

   double displayValue=0;

   // Получить число из дисплея калькулятора,
   // если он не пустой.
   // Восклицательный знак - это оператор отрицания

   if (!"".equals(dispFieldText)){

     displayValue= Double.parseDouble(dispFieldText);
```

```
}

Object src = e.getSource();

// Для каждой кнопки арифметического действия
// запомнить его: +, -, /, или *, сохранить текущее число
// в переменной currentResult, и очистить дисплей
// для ввода нового числа

if (src == parent.buttonPlus){

   selectedAction = '+';
   currentResult=displayValue;
   parent.displayField.setText("");

} else if (src == parent.buttonMinus){

   selectedAction = '-';
   currentResult=displayValue;
   parent.displayField.setText("");

 }else if (src == parent.buttonDivide){

   selectedAction = '/';
   currentResult=displayValue;
   parent.displayField.setText("");

 } else if (src == parent.buttonMultiply){

   selectedAction = '*';
   currentResult=displayValue;
   parent.displayField.setText("");

 } else if (src == parent.buttonEqual){

 // Совершить арифметическое действие, в зависимости
 // от selectedAction, обновить переменную currentResult
 // и показать результат на дисплее

    if (selectedAction=='+'){

      currentResult+=displayValue;

      // Сконвертировать результат в строку, добавляя его
      // к пустой строке и показать его
      parent.displayField.setText(""+currentResult);

    }else if (selectedAction=='-'){

      currentResult -=displayValue;
      parent.displayField.setText(""+currentResult);

    }else if (selectedAction=='/'){

      currentResult /=displayValue;
      parent.displayField.setText(""+currentResult);

    }else if (selectedAction=='*'){
```

```
        currentResult*=displayValue;
        parent.displayField.setText(""+currentResult);

      }

    } else{

      // Для всех цифровых кнопок присоединить надпись на
      // кнопке к надписи в дисплее

      String clickedButtonLabel= clickedButton.getText();
      parent.displayField.setText(dispFieldText +
                                  clickedButtonLabel);

    }
```

Наша заключительная версия окна калькулятора будет выглядеть как-то так:



Класс Calculator делает следующие действия:

- ✓ Создает и показывает все компоненты окна.
- ✓ Создает экземпляр слушателя CalculatorEngine.
- ✓ Передает CalculatorEngine ссылку на себя.
- ✓ Регистрирует этого слушателя во всех компонентах, которые создают события.

Вот последняя версия класса Calculator:

```
import javax.swing.*;
import java.awt.GridLayout;
import java.awt.BorderLayout;

public class Calculator {

      // Объявляем и инициализируем компоненты окна

      JButton button0=new JButton("0");
      JButton button1=new JButton("1");
      JButton button2=new JButton("2");
```

```
 JButton button3=new JButton("3");
 JButton button4=new JButton("4");
 JButton button5=new JButton("5");
 JButton button6=new JButton("6");
 JButton button7=new JButton("7");
 JButton button8=new JButton("8");
 JButton button9=new JButton("9");
 JButton buttonPoint = new JButton(".");
 JButton buttonEqual=new JButton("=");

 JButton buttonPlus=new JButton("+");
 JButton buttonMinus=new JButton("-");
 JButton buttonDivide=new JButton("/");
 JButton buttonMultiply=new JButton("*");

 JPanel windowContent = new JPanel();

 JTextField displayField = new JTextField(30);

// Конструктор
Calculator(){

// Установить менеджер расположения для панели
BorderLayout bl = new BorderLayout();

windowContent.setLayout(bl);

// Добавляем дисплей в верхней части окна

windowContent.add("North",displayField);


// Создаем панель с менеджером расположения GridLayout
// в которой будет 12 кнопок - 10 цифр, и
// кнопки "точка" и "равно"

 JPanel p1 = new JPanel();
 GridLayout gl =new GridLayout(4,3);

 p1.setLayout(gl);
 p1.add(button1);
 p1.add(button2);
 p1.add(button3);
 p1.add(button4);
 p1.add(button5);
 p1.add(button6);
 p1.add(button7);
 p1.add(button8);
 p1.add(button9);
 p1.add(button0);

 p1.add(buttonPoint);
 p1.add(buttonEqual);

// Добавляем панель p1 в центр окна

 windowContent.add("Center",p1);
```

```
      // Создаем панель с менеджером расположения GridLayout
      // на которой будет 4 кнопки -
      // Плюс, Минус, Разделить и Умножить

      JPanel p2 = new JPanel();
      GridLayout gl2 =new GridLayout(4,1);

      p2.setLayout(gl2);
      p2.add(buttonPlus);
      p2.add(buttonMinus);
      p2.add(buttonMultiply);
      p2.add(buttonDivide);

      // Добавляем панель p2 в правую часть окна
      windowContent.add("East",p2);

      // Создаем frame и добавляем в него содержимое        JFrame
frame = new JFrame("Calculator");
      frame.setContentPane(windowContent);

      // Устанавливаем размер окна, так чтобы уместились
      // все компоненты
      frame.pack();

      // Показываем окно
      frame.setVisible(true);

      // Создаем экземпляр слушателя событий и
      // регистрируем его в каждой кнопке
      CalculatorEngine calcEngine = new CalculatorEngine(this);

      button0.addActionListener(calcEngine);
      button1.addActionListener(calcEngine);
      button2.addActionListener(calcEngine);
      button3.addActionListener(calcEngine);
      button4.addActionListener(calcEngine);
      button5.addActionListener(calcEngine);
      button6.addActionListener(calcEngine);
      button7.addActionListener(calcEngine);
      button8.addActionListener(calcEngine);
      button9.addActionListener(calcEngine);

      buttonPoint.addActionListener(calcEngine);
      buttonPlus.addActionListener(calcEngine);
      buttonMinus.addActionListener(calcEngine);
      buttonDivide.addActionListener(calcEngine);
      buttonMultiply.addActionListener(calcEngine);
      buttonEqual.addActionListener(calcEngine);

  }

  public static void main(String[] args) {

     // Создаем экземпляр класса "Калькулятор"
     Calculator calc = new Calculator();

  }
}
```

Теперь скомпилируем проект и запустим класс Calculator. Она работает почти так же, как и настоящие калькуляторы. Поздравляю! Это ваша первая программа, которая может пригодиться многим людям – подарите ее своим друзьям.

Для того, чтобы лучше понимать, как работает эта программа, я рекомендую вам освоить *отладку* программ. Пожалуйста, прочитайте Приложение А про отладчик Eclipse, а потом продолжайте чтение дальше.

Некоторые другие слушатели событий

Существуют и другие слушатели в Java в пакете java.awt, которые неплохо было бы знать:

- ✓ Слушатель фокуса (`FocusListener`) посылает сигнал вашему классу, когда компонент окна получает или теряет *фокус*. Например, говорят, что текстовое поле имеет фокус, если в нем мигает курсор.
- ✓ Слушатель элемента (`ItemListener`) реагирует на выбор элементов в обычном или выпадающем списке.
- ✓ Слушатель клавиш (KeyListener) реагирует на нажатия клавиш.
- ✓ Слушатель мыши (`MouseListener`) реагирует, когда нажимают на кнопку мыши, или она входит в область компонента окна или выходит из нее.
- ✓ Слушатель движений мыши (`MouseMotionListener`) сообщает вам, когда мышь двигается или что-то тащит. *Тащить (drag)* означает двигать мышь с нажатой клавишей.
- ✓ Слушатель окна (`WindowListener`) дает вам шанс уловить моменты, когда пользователь открывает, закрывает, уходит из окна или заходит в него.

В следующей таблице вы увидите имена интерфейсов слушателей, и методы, которые эти интерфейсы объявляют.

Interface	**Methods to implement**
FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
ItemListener	itemStateChanged(ItemEvent)
KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
MouseListener	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
WindowListener	windowActivated (WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated (WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)

Напрмер, интерфейс FocusListener объявляет два метода: focusGained() и focusLost(). Это означает, что даже если ваш класс заинтересован только в обработке события получения фокуса каким-либо элементом окна, вы все равно должны включить пустой метод focusLost(). Это может раздражать, поэтому Java предоставляет специальные *классы-адаптеры* для каждого слушателя, чтобы упростить обработку событий.

Как использовать адаптеры

Скажем, вам нужно сохранить какую-нибудь информацию на диск, когда пользователь закрывает окно. В соответствии с предыдущей таблицей, класс, который реализует интерфейс WindowsListener должен включать семь методов. Это значит, что вам придется писать текст программы в методе windowClosing() и еще включить шесть пустых методов.

В пакете java.awt есть адаптеры, которые являются классами, уже реализовавшими все необходимые методы (правда, эти методы пустые внутри). Один из таких классов – так называемый WindowAdapter. Вы можете унаследовать класс, который обрабатывает события, от класса WindowAdapter и просто переопределить методы, которые вам нужны, например метод windowClosing().

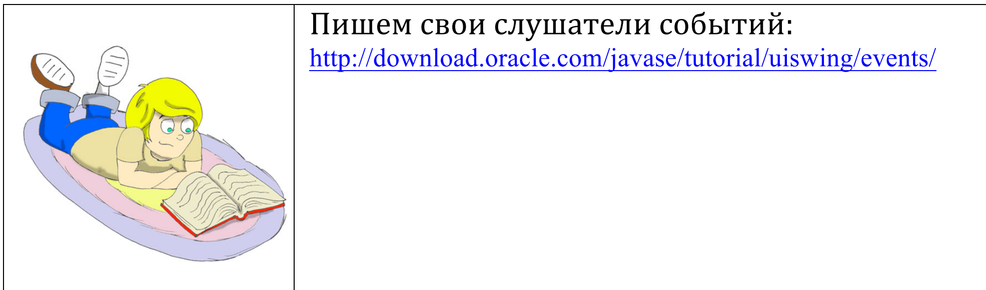
```
class MyEventProcessor extends java.awt.WindowsAdapter {

  public void windowClosing(WindowEvent e) {

     // здесь находится ваш текст программы,
     // который сохраняет данные на диск

  }

}
```

Остальное просто – просто зарегистрируйте этот класс, как слушатель событий окна в вашем классе:

```
MyEventProcessor myListener =  new MyEventProcessor();
addWindowListener(myListener);
```

Такого же результата можно достичь, используя так называемые *анонимные внутренние классы*, но эта тема немного сложновата для этой книги.

Материалы для дополнительного чтения



Пишем свои слушатели событий:
http://download.oracle.com/javase/tutorial/uiswing/events/

Практические упражнения

Попробуйте разделить число на ноль с помощью нашего калькулятора - дисплей покажет слово Infinity. Измените класс CalculatorEngine, чтобы отображалось сообщение *"На ноль делить нельзя"*, если пользователь нажмет на кнопку *"Разделить"*, когда дисплей калькулятора будет пуст.

Практические упражнения для умников и умниц

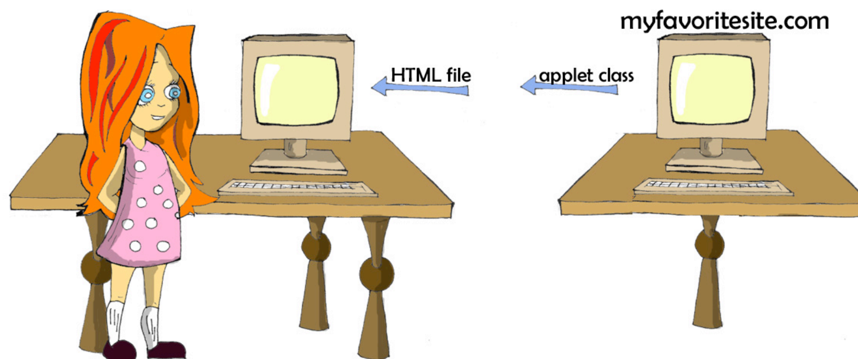
Измените класс CalculatorEngine , чтобы запретить вводить больше одной точки в числе.

Подсказка: прочтите про метод `indexOf()` класса `String`, чтобы узнать, есть ли уже в числе точка.

Глава 7. Апплет Крестики-Нолики

Когда вы заходите на свой любимый сайт, есть небольшая вероятность, что некоторые из игр или других приложений на сайте были написаны на Java с помощью технологии так называемых *апплетов.* Эти специальные приложения живут и работают внутри окна веб-браузера. Веб-браузеры понимают простой язык разметки, который называется HTML, который позволяет вам вставлять специальные метки (*"теги")* в текстовые файлы, чтобы они красиво отображались в браузерах. Кроме текста, вы можете вставлять в файлы HTML специальный тег <applet>, который подскажет браузеру, где найти и как правильно показать апплет Java.

Java-апплеты загружаются на ваш компьютер из интернета, как часть веб-странички, а браузер достаточно умен, чтобы запустить свою JVM для того, чтобы запустить эти апплеты.



В этой главе вы научитесь, как создавать апплеты на своем компьютере, а в Приложении В узнаете, как опубликовать свои веб-

странички в интернете, чтобы другие люди тоже могли ими пользоваться.

Люди бродят в интернете, не зная, содержат ли веб-страницы апплет Java или нет, но они хотят быть уверены, что их компьютерам не угрожают какие-нибудь плохие парни, которые добавили какой-нибудь зловредный апплет на страничку. Поэтому апплеты были разработаны со следующими ограничениями:

- Апплеты не могут получить доступ к файлам на вашем диске, пока вы не скачали себе на диск специальный *сертификат,* который дает им такое право.

- Апплеты могут подсоединяться по сети только к тому компьютеру, с которого они были скачаны.

- Апплеты не могут запускать никакие другие программы на вашем компьютере.

Чтобы запустить апплет, вам понадобится особым образом написанный класс Java, текстовый файл HTML, который содержит тег <applet>, указывающий на этот класс, и веб-браузер, поддерживающий Java. Также можно тестировать апплеты в Eclipse или используя специальную программу, называемую *appletviewer*. Но прежде чем учиться делать апплеты, давайте потратим 15 минут на то, чтобы познакомиться с некоторыми тегами HTML.

Изучаем HTML за 15 минут

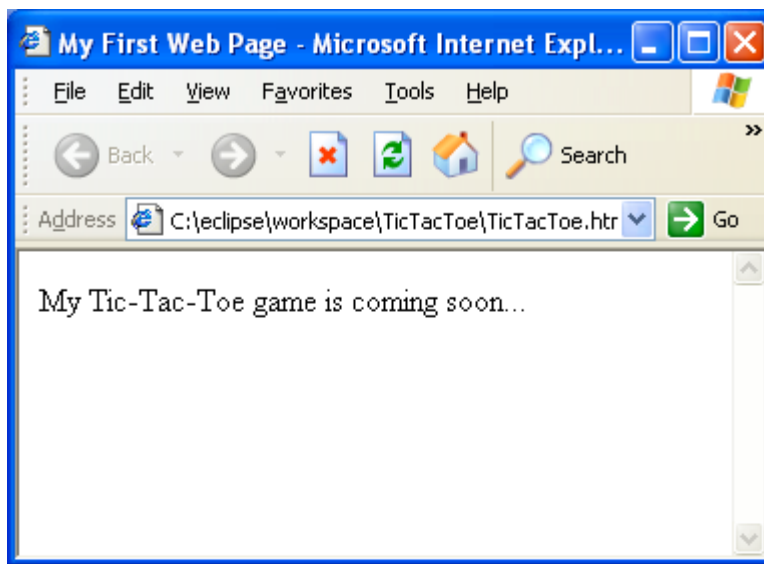
Представьте на секунду, что вы написали и скомпилировали игру-апплет под названием Крестики-Нолики (TicTacToe). Теперь вам нужно создать файл HTML с информацией о нем. Сначала создайте текстовый файл и назовите его TicTacToe.html (кстати, Eclipse может создавать и текстовые файлы тоже).

Имена файлов HTML заканчиваются на *.html* или *.htm*. Внутри у них обычно есть секции "заголовок" (*header)* и "тело" (*body)*. Большинство тегов HTML имеют соответствующие закрывающие теги, которые начинаются с прямого слэша ("/"), например <Head> и </Head>. Файл TicTacToe.html может выглядеть вот так:

```
<HTML>

    <HEAD>
       <TITLE>My First Web Page</TITLE>
```

```
    </HEAD>

    <BODY>
      My Tic-Tac-Toe game is coming soon
    </BODY>

</HTML>
```

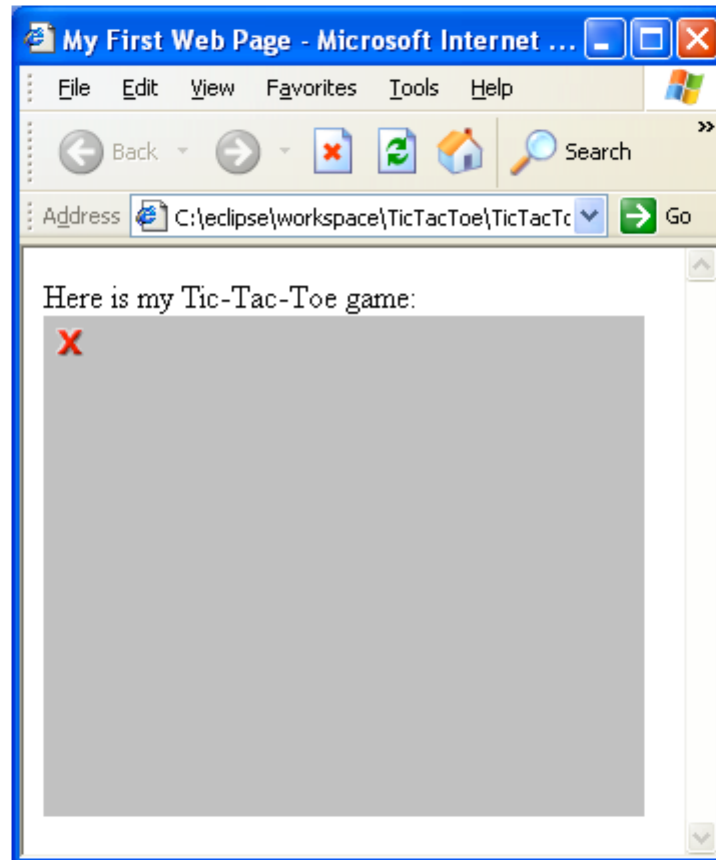
Вы можете располагать теги на одной строке, как мы сделали с тегами <Title> и </Title>, или же на разных строках. HTML - не Java, здесь не важно, как писать – большими или маленькими буквами. Откройте этот файл в своем веб-браузере с помощью меню *Файл* и *Открыть*. В заголовке окна будет написано *My First Web Page,* а на самой страничке вы увидите слова *My Tic-Tac-Toe game is coming soon…*:



Теперь изменим этот файл, добавив туда тег для апплета “Крестики-нолики”:

```
<HTML>
<BODY>
  Here is my Tic-Tac-Toe game:
  <APPLET code="TicTacToe.class" width=300 height=250>
  </APPLET>
</BODY>
</HTML>
```

Теперь экран выглядит по-другому:



Это неудивительно: поскольку веб-браузер не смог найти класс TicTacToe, он просто показал серый прямоугольник. Мы создадим этот класс чуть позже.

Теги HTML заключаются в угловые скобки, и некоторые из тегов могут иметь дополнительные *атрибуты*. Тег <APPLET> в нашем примере использует следующие атрибуты:

- code - это имя класса Java-апплета.

- width – это ширина в *пикселях* прямоугольной области экрана, где будет располагаться апплет. Изображения на экране компьютера состоят из маленьких точек, которые называются пикселями.

- height - это высота области экрана, где будет располагаться апплет.

Если Java-апплет состоит из нескольких классов, упакуйте их все в один архивный файл с помощью программы *jar,* которая поставляется вместе с JDK (Java Developer Kit – Комплект Разработчика Java). Если вы так сделаете, атрибут “архив” (*“archive”*) должен иметь значение, равное имени этого архива. Вы можете прочитать про архивы jar в Приложении А.

Апплеты и AWT

Зачем использовать библиотеку AWT для того, чтобы писать апплеты, если Swing лучше? Можно ли писать апплеты, используя классы Swing? Да, но вы должны знать о некоторых нюансах.

Веб-браузеры поставляются со своими собственными версиями JVM, которые поддерживают AWT, и могут не поддерживать классы Swing, которые включены в ваш апплет. Конечно же, пользователи могут скачать и установить последнюю версию JVM, и есть даже специальные конвертеры HTML, которые изменят файл HTML так, чтобы их браузеры могли скачать эту новую версию JVM, но действительно ли вы хотите попросить пользователей сделать это? После того, как ваша страничка будет опубликована в Интернете, вы не будете знать, кто станет ей пользоваться.

Представьте себе старичка где-нибудь в пустыне с компьютером десятилетней давности – он просто уйдет с вашей странички, вместо того, чтобы проходить через все эти неприятности с установкой. Представьте, что наш апплет помогает продавать онлайн-игры, и мы не хотим потерять этого человека – он может быть нашим потенциальным покупателем (у людей в пустыне тоже бывают кредитные карточки).

Используйте AWT, если вашими апплетами будут пользоваться неадекваты, работающие на компьютерах прошлого века.

С другой стороны, все не так уж плохо. Последние версии Java включают так называемый плагин следующего поколения. Теперь апплеты не обязаны выполняться в JVM, которая идет с вашим Веб браузером – они выполняются в отдельной JVM, которая запускается этим плагином. Апплет все также живет внутри окошка браузера, но уже не зависит от желания (или нежелания) производителей браузеров включать самые свежие JVM в свои поставки. Подробнее обо всем этом можно почитать здесь: https://jdk6.dev.java.net/plugin2 .

Чтобы проверить или поменять установки этого плагина в Microsoft Windows зайдите в Java Control Panel - нажмите на иконку Java в системном меню Start | Control Panel. Под закладкой Advanced найдите Java Plug-in и убедитесь, что выбрана установка Enable the next-generation Java Plug-in. Процесс подключения плагина на Макбуках описан здесь: http://blogs.sun.com/thejavatutorials/entry/enabling_the_next_generation_java .

Между нами, уже и Swing устарел для написания графических програм для интернет приложений. Создатели Java придумали новый язык специально для этих целей. Он называется JavaFX и, если кто интересуется, то вам сюда: http://javafx.com/.

Как писать апплеты

Java-апплеты AWT должны быть унаследованы от класса `java.applet.Applet`, например:

```
class TicTacToe extends java.applet.Applet {

}
```

Если использовать Swing, то наследоваться нужно от класса `JApplet`:

```
class TicTacToe extends javax.swing.JApplet {

}
```

В отличие от обычных программ Java, апплетам не нужен метод main(), потому что веб-браузер сам *скачает* и запустит их, как только встретит на страничке тег <applet>. Также браузер будет посылать сигналы апплетам, когда будут происходить важные события, например запуск апплета, перерисовка апплета, и так далее. Чтобы убедиться, что апплет реагирует на эти события, вы должны написать специальные *методы обратного вызова ("callback methods")*: `init()`, `start()`, `paint()`, `stop()`, и `destroy()`. JVM веб-браузера будет вызывать эти методы в следующих случаях:

- ✓ `init()` вызовется, когда апплет загружается браузером. Он вызывается только один раз, таким образом, этот метод играет роль конструктора в обычных классах Java.
- ✓ `start()` вызовется сразу после `init()`. Он также вызывается, когда пользователь возвращается на эту страничку после посещения другой странички.

✓ `paint()` вызовется, когда потребуется показать или обновить окно апплета после каких-либо действий на экране. Например, апплет перекрывается каким-то другим окном и браузеру нужно его перерисовать.

- `stop()` вызовется, когда пользователь покидает веб-страничку, содержащую апплет.

- `destroy()` – вызовется, когда браузер уничтожает апплет. Вам придется писать текст программы в этом методе только, если апплет использует некоторые внешние ресурсы, например, он поддерживает соединение с компьютером, с которого он был загружен.

И хотя вы не обязаны писать все эти методы, в каждом апплете должен быть хотя бы один из этих методов: `init()` или `paint()`. Вот текст программы апплета, который показывает слова “*Привет, Мир!*”. В этом апплете есть только один метод `paint()`, который получает экземпляр объекта `Graphics` от JVM веб-браузера. У этого объекта есть целый набор методов для рисования. В следующем примере используется метод `drawString()`, чтобы нарисовать текст “*Привет, Мир!*”.

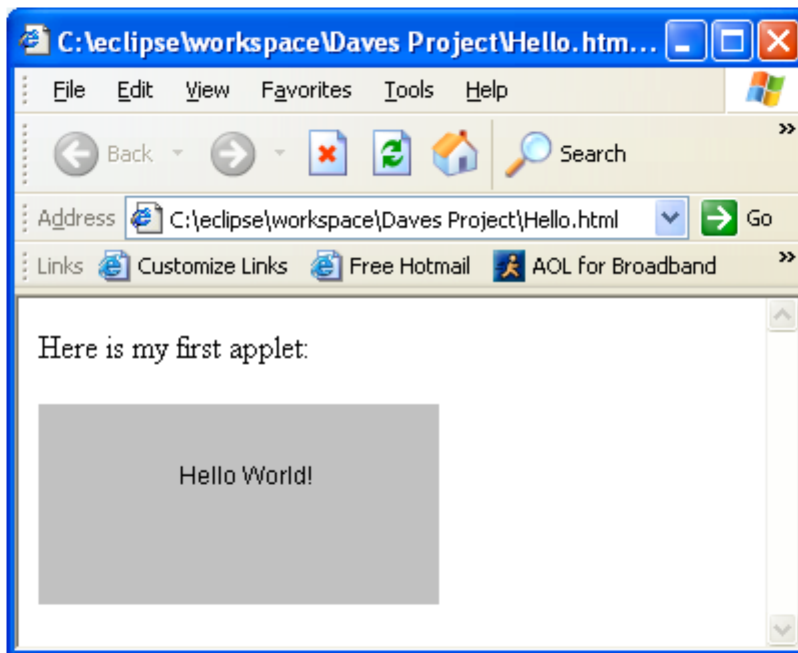
```
public class HelloApplet extends java.applet.Applet {
      public void paint(java.awt.Graphics graphics) {

      graphics.drawString("Привет, Мир!", 70, 40);

      }
}
```

Создайте этот класс в Eclipse. Затем в окне “*Run*” выберите “*Java Applet*” в левом верхнем углу, нажмите кнопку “*New*”, и введите HelloApplet в поле “*Applet Class*”.

Чтобы протестировать этот апплет в веб-браузере, создайте файл *Hello.html* в той же папке, где находится ваш апплет:

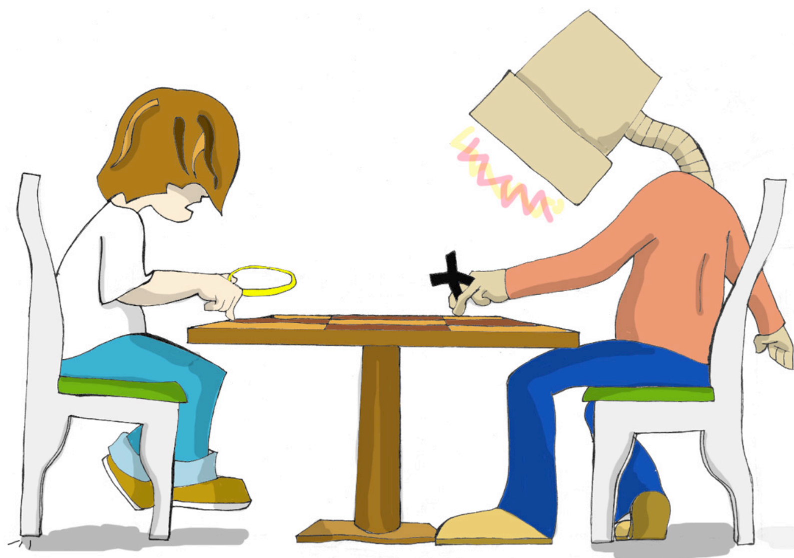
```
<HTML>
 <BODY>
  Here is my first applet:<P>
 <APPLET code=”HelloApplet.class” width=200 height=100>
 </APPLET>
 </BODY>
</HTML>
```

Теперь запустите ваш веб-браузер и откройте файл *Hello.html* с помощью пунктов меню *File* и *Open*. Окно должно выглядеть примерно так:



Как вы думаете, после этого простого примера мы сможем написать игру? Еще бы! Пристегните ваши ремни…

Пишем игру Крестики-нолики



Стратегия

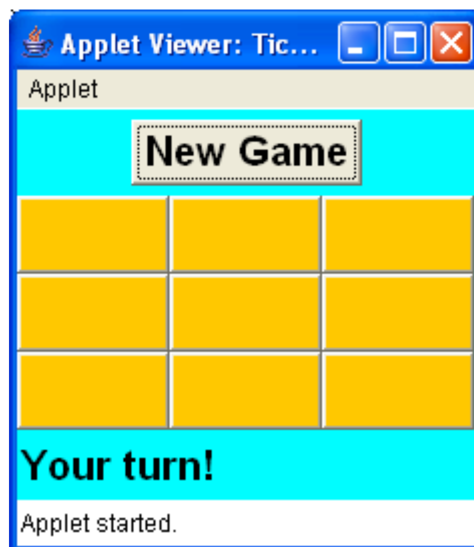
Каждая игра использует какой-либо алгоритм – набор правил или стратегию, которая применяется в зависимости от действий игрока. Алгоритмы для одной и той же игры могут быть простыми или очень сложными. Когда вы слышите, что чемпион мира по шахматам Гарри Каспаров играет против компьютера, на самом деле он играет против программы. Целые команды экспертов пытаются изобрести изощренные алгоритмы, чтобы обыграть его. Игра Крестики-нолики также может быть запрограммирована с помощью разных стратегий, ну а мы используем самую простую:

- ✓ У нас будет доска размером 3x3. Человек будет играть крестиками, а компьютер будет играть ноликами.
- ✓ Чтобы победить, надо полностью заполнить ряд, колонку или диагональ поля одинаковыми символами.
- ✓ После каждого хода программа должна проверять, есть ли победитель. Если есть победитель, то выигрышная комбинация должна выделяться другим цветом и игра должна заканчиваться.
- ✓ Игра также должна заканчиваться, если больше не осталось свободных клеток. Чтобы начать новую игру, человек должен нажать кнопку “New Game”.
- ✓ Когда компьютер принимает решение, куда поставить следующий нолик, он должен попытаться найти ряд, колонку или диагональ, в которой уже есть два нолика, и поставить там третий.
- ✓ Если таких рядов, колонок или диагоналей нет, то компьютер должен попытаться найти такие же ряды с двумя крестиками, и поставить там нолик, чтобы заблокировать выигрышный ход игрока.
- ✓ Если не было найдено ни выигрышного, ни блокирующего хода, то компьютер должен попытаться занять центральную клетку, или же занять любую случайную свободную клетку.

Текст программы

Здесь я коротко опишу программу, потому что в тексте программы апплета есть много комментариев, которые помогут вам понять, как она работает.

Апплет будет использовать `BorderLayout`, в верхней части окна будет раположена кнопка “*New Game*”. В центре окна будут располагаться девять кнопок, представляющие клетки поля, а в нижней части будут отображаться сообщения:



Все компоненты окна будут создаваться в методе апплета `init()`. Все события будут обрабатываться слушателем `ActionListener` в методе `actionPerformed()`. Метод lookForWinner() будет вызываться после каждого хода, чтобы проверить, закончилась ли игра.

Последние три правила нашей стратегии запрограммированы в методе computerMove(), которому может понадобиться генерировать *случайные числа*. Это делается с помощью класса Java `Math` и его метода `random()`.

Вам также может показаться необычным синтаксис, когда несколько методов вызываются в одном *выражении*, например:

```
if(squares[0].getLabel().equals(squares[1].getLabel())){…}
```

Эта строка делает программу короче, потому что в ней производятся такие же действия, которые могли быть сделаны в следующих строках:

```
String label0 = squares[0].getLabel();
String label1 = squares[1].getLabel();

if(label0.equals(label1)){…}
```

Java вычисляет выражение в скобках, прежде чем производить любые другие вычисления. Короткая версия этого текста программы сначала получает результат выражения в скобках, и сразу же использует его как аргумент для метода `equals()`, который применяется для результата первого вызова `getLabel()`.

Хотя текст программы игры занимает несколько страниц, он не должен быть сложным для понимания. Просто читайте код и все комментарии в программе.

```
/**
*  Игра крестики-нолики на доске 3x3
*/

import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public class TicTacToe extends Applet implements ActionListener{

Button squares[];
Button newGameButton;
Label score;
int emptySquaresLeft=9;

/**
 * Метод init – это конструктор апплета
 */

 public void init(){

 //Устанавливаем менеджер расположения апплета, шрифт и цвет

  this.setLayout(new BorderLayout());
  this.setBackground(Color.CYAN);

  // Изменяем шрифт апплета так, чтобы он был жирным
  // и имел размер 20

  Font appletFont=new Font("Monospased",Font.BOLD, 20);
  this.setFont(appletFont);

  // Создаем кнопку “New Game” и регистрируем в ней
  // слушатель действия
```

```
 newGameButton=new Button("New Game");

 newGameButton.addActionListener(this);

 Panel topPanel=new Panel();
 topPanel.add(newGameButton);

     this.add(topPanel,"North");

     Panel centerPanel=new Panel();
     centerPanel.setLayout(new GridLayout(3,3));
     this.add(centerPanel,"Center");

     score=new Label("Your turn!");
     this.add(score,"South");

  // создаем массив, чтобы хранить ссылки на 9 кнопок

     squares=new Button[9];

 // Создаем кнопки, сохраняем ссылки на них в массиве
 // регистрируем в них слушатель, красим их
 // в оранжевый цвет и добавляем на панель

     for(int i=0;i<9;i++){

      squares[i]=new Button();
      squares[i].addActionListener(this);
      squares[i].setBackground(Color.ORANGE);
      centerPanel.add(squares[i]);

    }
}

 /**
 * Этот метод будет обрабатывать все события
 * @param ActionEvent объект
 */

 public void actionPerformed(ActionEvent e) {

  Button theButton = (Button) e.getSource();

  // Это кнопка New Game ?

  if (theButton ==newGameButton){

    for(int i=0;i<9;i++){

      squares[i].setEnabled(true);
      squares[i].setLabel("");
      squares[i].setBackground(Color.ORANGE);

    }

   emptySquaresLeft=9;
   score.setText("Your turn!");
   newGameButton.setEnabled(false);
```

```
   return;  // выходим из метода
}

 String winner = "";

 // Это одна из клеток?

 for ( int i=0; i<9; i++ ) {

  if (  theButton == squares[i] ) {

    squares[i].setLabel("X");
    winner = lookForWinner();

    if(!"".equals(winner)){

       endTheGame();

    } else {

       computerMove();
       winner = lookForWinner();

       if ( !"".equals(winner)){
          endTheGame();
       }
    }

    break;
   }
 } // конец цикла for

 if ( winner.equals("X") ) {
     score.setText("You won!");

 } else if (winner.equals("O")){
     score.setText("You lost!");

 } else if (winner.equals("T")){
     score.setText("It's a tie!");

 }
} // конец метода actionPerformed



/**
 *  Этот метод вызывается после каждого хода, чтобы узнать,
 *  есть ли победитель. Он проверяет каждый ряд, колонку
 *  и диагональ, чтобы найти три клетки с одинаковыми надписями
 *  (не пустыми)
 *  @return "X", "O", "T" – ничья, "" - еще нет победителя
 */

 String lookForWinner() {

  String theWinner = "";
  emptySquaresLeft--;
```

```
  if (emptySquaresLeft==0){
    return "T";  // это ничья. T от английского слова tie

  }



// Проверяем ряд 1 - элементы массива 0,1,2
if (!squares[0].getLabel().equals("") &&

  squares[0].getLabel().equals(squares[1].getLabel()) &&
  squares[0].getLabel().equals(squares[2].getLabel())) {

    theWinner = squares[0].getLabel();
    highlightWinner(0,1,2);

// Проверяем ряд 2 - элементы массива 3,4,5
} else if (!squares[3].getLabel().equals("")  &&

 squares[3].getLabel().equals(squares[4].getLabel()) &&
 squares[3].getLabel().equals(squares[5].getLabel())) {

    theWinner = squares[3].getLabel();
    highlightWinner(3,4,5);

// Проверяем ряд 3 - элементы массива 6,7,8
} else if ( ! squares[6].getLabel().equals("") &&

 squares[6].getLabel().equals(squares[7].getLabel()) &&
 squares[6].getLabel().equals(squares[8].getLabel())) {

    theWinner = squares[6].getLabel();
    highlightWinner(6,7,8);

// Проверяем колонку 1 - элементы массива 0,3,6
} else if ( ! squares[0].getLabel().equals("") &&

 squares[0].getLabel().equals(squares[3].getLabel()) &&
 squares[0].getLabel().equals(squares[6].getLabel())) {

    theWinner = squares[0].getLabel();
    highlightWinner(0,3,6);

// Проверяем колонку 2 - элементы массива 1,4,7
} else if ( ! squares[1].getLabel().equals("") &&

 squares[1].getLabel().equals(squares[4].getLabel()) &&
 squares[1].getLabel().equals(squares[7].getLabel())) {

     theWinner = squares[1].getLabel();
     highlightWinner(1,4,7);

// Проверяем колонку 3 - элементы массива 2,5,8
} else if (  ! squares[2].getLabel().equals("") &&

 squares[2].getLabel().equals(squares[5].getLabel()) &&
 squares[2].getLabel().equals(squares[8].getLabel())) {
```

```
     theWinner = squares[2].getLabel();
     highlightWinner(2,5,8);

// Проверяем первую диагональ - элементы массива 0,4,8

} else if ( ! squares[0].getLabel().equals("") &&

  squares[0].getLabel().equals(squares[4].getLabel()) &&
  squares[0].getLabel().equals(squares[8].getLabel())) {

     theWinner = squares[0].getLabel();
     highlightWinner(0,4,8);

// Проверяем вторую диагональ - элементы массива 2,4,6
} else if ( ! squares[2].getLabel().equals("") &&

 squares[2].getLabel().equals(squares[4].getLabel()) &&
 squares[2].getLabel().equals(squares[6].getLabel())) {

     theWinner = squares[2].getLabel();
     highlightWinner(2,4,6);
}

 return theWinner;

}

/**
 * Этот метод применяет набор правил, чтобы найти
 * лучший компьютерный ход. Если хороший ход
 * не найден, выбирается случайная клетка.
 */

 void computerMove() {

   int selectedSquare;

    // Сначала компьютер пытается найти пустую клетку
    // рядом с двумя клетками с ноликами, чтобы выиграть

     selectedSquare = findEmptySquare("O");

    // Если он не может найти два нолика, то хотя бы
    // попытается не дать оппоненту сделать ряд из 3-х
    // крестиков,поместив нолик рядом с двумя крестиками
    if ( selectedSquare == -1 )

        selectedSquare =  findEmptySquare("X");

    }

    // если selectedSquare все еще равен -1, то
    // попытается занять центральную клетку
    if ( (selectedSquare == -1)
             &&(squares[4].getLabel().equals("")) ){

       selectedSquare=4;

    }
```

```
    // не повезло с центральной клеткой...
    // просто занимаем случайную клетку
    if ( selectedSquare == -1 ){

       selectedSquare = getRandomSquare();

    }

    squares[selectedSquare].setLabel("O");
}

/**
 * Этот метод проверяет каждый ряд, колонку и диагональ
 * чтобы узнать, есть ли в ней две клетки
 * с одинаковыми надписями и пустой клеткой.
 * @param   передается X - для пользователя и O - для компа
 * @return  количество свободных клеток,
 *          или -1, если не найдено две клетки
 *          с одинаковыми надписями
 */
 int findEmptySquare(String player) {

   int weight[] = new int[9];

   for ( int i = 0; i < 9; i++ ) {

        if ( squares[i].getLabel().equals("O") )
           weight[i] = -1;
        else if ( squares[i].getLabel().equals("X") )
           weight[i] = 1;
        else
           weight[i] = 0;
   }

   int twoWeights = player.equals("O") ? -2 : 2;

    // Проверим, есть ли в ряду 1 две одинаковые клетки и
    // одна пустая.
    if ( weight[0] + weight[1] + weight[2] == twoWeights ) {
        if ( weight[0] == 0 )
            return 0;
        else if ( weight[1] == 0 )
            return 1;
        else
            return 2;
    }

    // Проверим, есть ли в ряду 2 две одинаковые клетки и
    // одна пустая.
    if (weight[3] +weight[4] + weight[5] == twoWeights) {
        if ( weight[3] == 0 )
            return 3;
        else if ( weight[4] == 0 )
            return 4;
        else
            return 5;
    }
```

```
 // Проверим, есть ли в ряду 3 две одинаковые клетки и
 // одна пустая.
 if (weight[6] + weight[7] +weight[8] == twoWeights ) {

      if ( weight[6] == 0 )
          return 6;
      else if ( weight[7] == 0 )
          return 7;
      else
          return 8;
}

// Проверим, есть ли в колонке 1 две одинаковые клетки и
// одна пустая.
if (weight[0] + weight[3] + weight[6] == twoWeights) {

      if ( weight[0] == 0 )
          return 0;
      else if ( weight[3] == 0 )
          return 3;
      else
          return 6;
}

// Проверим, есть ли в колонке 2 две одинаковые клетки
// и одна пустая.
if (weight[1] +weight[4] + weight[7] == twoWeights ) {

      if ( weight[1] == 0 )
          return 1;
      else if ( weight[4] == 0 )
          return 4;
      else
          return 7;
}

 // Проверим, есть ли в колонке 3 две одинаковые клетки
 // и одна пустая.
if (weight[2] + weight[5] + weight[8] == twoWeights ){

      if ( weight[2] == 0 )
          return 2;
      else if ( weight[5] == 0 )
          return 5;
      else
          return 8;

}

 // Проверим, есть ли в диагонали 1 две одинаковые клетки
 // и одна пустая.
 if (weight[0] + weight[4] + weight[8] == twoWeights ){
```

```
      if ( weight[0] == 0 )
          return 0;
      else if ( weight[4] == 0 )
          return 4;
      else
          return 8;
 }

 // Проверим, есть ли в диагонали 2 две одинаковые клетки
 // и одна пустая.
 if (weight[2] + weight[4] + weight[6] == twoWeights ){

      if ( weight[2] == 0 )
         return 2;
      else if ( weight[4] == 0 )
         return 4;
      else
         return 6;
  }

 // Не найдено двух одинаковых соседних клеток
 return -1;

} // конец метода findEmptySquare()

/**
 * Этот метод выбирает любую пустую клетку
 * @return случайно выбранный номер клетки
 */

 int getRandomSquare() {

  boolean gotEmptySquare = false;

  int selectedSquare = -1;

  do {

    selectedSquare = (int) (Math.random() * 9 );

    if (squares[selectedSquare].getLabel().equals("")){

       gotEmptySquare = true; // чтобы закончить цикл

    }

  } while (!gotEmptySquare );

  return selectedSquare;

 } // конец метода getRandomSquare()

/**
 * Этот метод выделяет выигравшую линию.
 * @param первая, вторая и третья клетки для выделения
 */
void highlightWinner(int win1, int win2, int win3) {
```

```
    squares[win1].setBackground(Color.CYAN);
    squares[win2].setBackground(Color.CYAN);
    squares[win3].setBackground(Color.CYAN);
}

// Делаем недоступными клетки и доступной кнопку "New Game"
void endTheGame(){

    newGameButton.setEnabled(true);

    for(int i=0;i<9;i++){

            squares[i].setEnabled(false);

    }
 }
} // конец класса
```

Поздравляю! Вы создали вашу первую игру на Java!

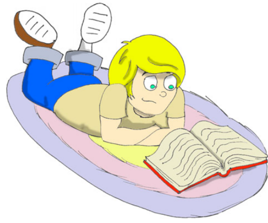
При использовании библиотеки Swing, нужно поменять `Button` на `JButton`, `Panel` на `JPanel`, и т.д. Более детальная информация, правда по-анлийски, здесь: http://download.oracle.com/javase/tutorial/deployment/applet.

Этот апплет можно запустить или прямо из Eclipse, или открыв файл TicTacToe.html, который мы создали в начале главы, только скопируйте файлы *TicTacToe.html* и *TicTacToe.class* в одну папку. В нашем классе TicTacToe есть маленькая ошибка – возможно, вы ее даже не заметили, но я уверен, что она исчезнет после того, как вы сделаете второе задание ниже.

Наш класс TicTacToe использует простую стратегию, потому что наша цель – просто научиться писать программы, но если вы хотите улучшить эту игру, изучите так называемую *стратегию минимакса,* которая позволит компьютеру выбрать наилучший ход. В этой книге нет описания стратегии минимакса, зато оно доступно онлайн.

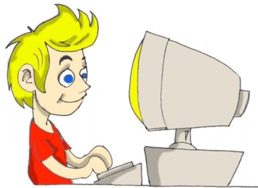
Материалы для дополнительного чтения

Апплеты Java:
http://download.oracle.com/javase/tutorial/deployment/applet/



Класс `Math`
http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Math.html

Практические упражнения



1. Добавьте на верхнюю панель класса TicTacToe две надписи для подсчета выигрышей и проигрышей. Для этого объявите две переменные в классе и увеличивайте соответствующую переменную каждый раз, когда человек выигрывает или проигрывает. Счет должен обновляться сразу после того, как программа выводит сообщение “*You won*” или “*You lost*”.

2. Наша программа позволяет щелкать на клетке, в которой уже есть крестик или нолик. Это ошибка! Программа продолжает работать, как будто вы сделали правильный ход. Измените текст программы так, чтобы нажатия на такие клетки игнорировались.

3. Добавьте метод main() к классу TicTacToe, чтобы иметь возможность запускать игру не как апплет, а как Java-приложение.

Практические упражнения для умников и умниц

1. Перепишите TicTacToe, чтобы заменить одномерный массив, который хранит кнопки

```
JButton squares[]
```

на двумерный массив 3x3:

```
JButton squares[][]
```

2. Почитайте про многомерные массивы в интернете.

Глава 8. Исключения – ошибки в программах

Скажем, вы забыли закрыть фигурные скобки в своем Java-коде.

Это приведет к ошибке компиляции, которую можно легко исправить. Но существуют ещё ошибки времени исполнения *(run-time errors)*, когда совершенно неожиданно, программа перестаёт работать, как положено. Например, Java-класс считывает файл со счётом в игре. Что произойдёт, если кто-то удалит этот файл? Остановится ли программа с длинным и страшным сообщением об ошибке, или продолжит работать, выдав дружелюбное сообщение, типа:

```
Дорогой друг, по какой-то причине, мне не удалось прочитать файл scores.txt. Пожалуйста, проверь, существует ли этот файл?
```

Создавайте программы, готовые к необычным ситуациям. Во многих языках программирования обработка ошибок зависит только от доброй воли программиста. Но Java заставляет добавлять код обработки ошибок, иначе программа даже не скомпилируется.

Ошибки времени выполнения в Java называются *исключениями (exceptions)*, а обработка таких ошибок называется *обработкой исключений (exception handling)*. Код, который может вызывать исключения, помещайте в так называемый блок `try/catch`. Это как будто сказать JVM следующее: *Попробуй прочитать файл со счётом игры, но если что-то пойдёт не так, перехвати ошибку и запусти код, который с ней разберётся:*

```
try{

  fileScores.read();

} catch (IOException e){

  System.out.println(
   "Дорогой друг, мне не удалось прочитать файл
```

```
scores.txt");
}
```

Вы научитесь работать с файлами в главе 9, а сейчас запомните новый термин *I/O* или *ввод/вывод (input/output)*. Операции чтения и записи (на диск или другое устройство) называются вводом/выводом, отсюда `IOException` — это класс, который содержит информацию об ошибках ввода/вывода.

Метод *генерирует (throws) исключение* в случае ошибки. Для разных типов ошибок генерируются разные исключения. Если в программе есть блок `catch` для данного конкретного типа ошибки, то она перехватится и программа перейдет к выполнению кода в блоке `catch`. Программа продолжит работу, и будет считать, что об этом исключении позаботились.

В коде, приведенном выше, вывод текста произойдет только в случае ошибки чтения файла.

Чтение трассировки стека

Если случается непредусмотренное исключение, которое не обрабатывается программой, на экран выводится многострочное сообщение об ошибке. Такое сообщение называется *трассировкой стека (stack trace)*. Если ваша программа вызывала несколько методов перед тем, как столкнулась с проблемой, трассировка стека поможет отследить проблему и найти строчку кода, которая вызвала ошибку.

Давайте напишем программу `TestStackTrace`, которая специально делила бы на ноль (номера строк не являются частью кода).

```
1  class TestStackTrace{
2     TestStackTrace()
3     {
4       divideByZero();
5     }
6
7     int divideByZero()
8     {
9       return 25/0;
10    }
11
12    static void main(String[]args)
13    {
14      new TestStackTrace();
15    }
16 }
```

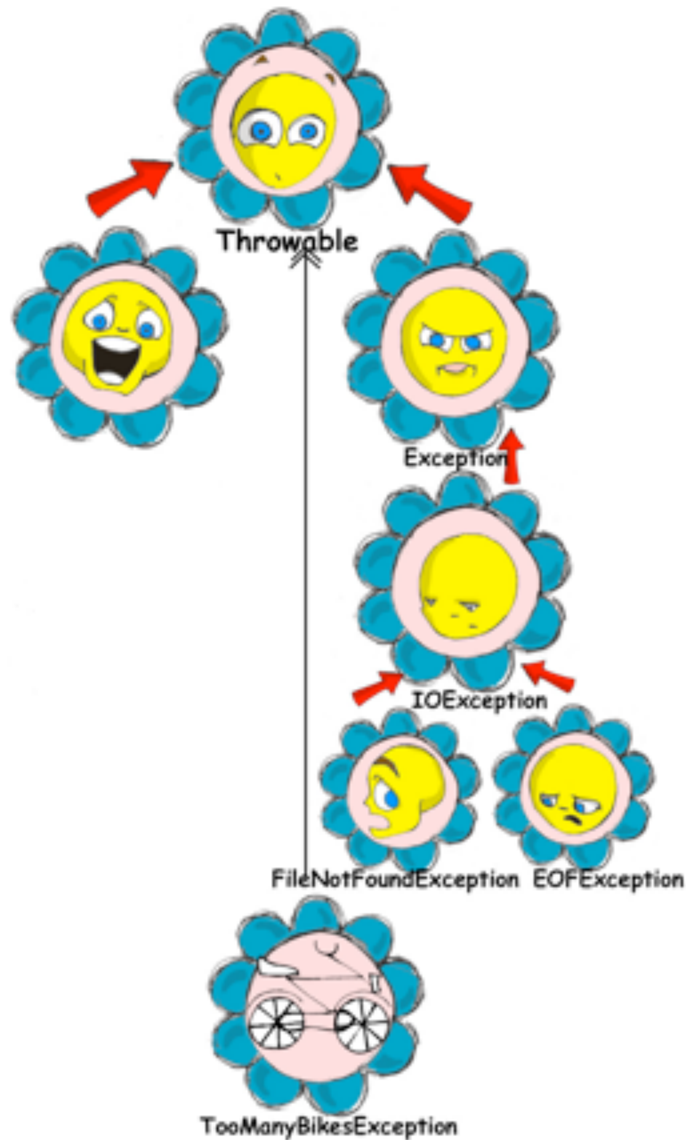
Вывод этой программы показывает последовательность методов, вызванных до момента возникновения ошибки времени выполнения. Начинайте читать этот вывод с последней строчки, продвигаясь вверх.

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
   at TestStackTrace.divideByZero(TestStackTrace.java:9)
   at TestStackTrace.<init>(TestStackTrace.java:4)
   at TestStackTrace.main(TestStackTrace.java:14)
```

Это означает, что выполнение программы началось с метода `main()`, потом перешло к `init()` — конструктору, затем был вызван метод `divideByZero()`. Числа 14, 4 и 9 указывают, в каких строчках программы эти методы были вызваны. После этого, было вызвано исключение `ArithmeticException`, потому что в строке 9 программа пыталась разделить на ноль.

Генеалогическое дерево исключений

Исключения в Java — это тоже классы, и некоторые из них представлены вот в такой иерархии (стрелка вверх указывает на родителя):



Потомки класса `Exception` называются *некритичными исключениями (checked exceptions)*, и вы должны обрабатывать их в своём коде. Т.е. программа как-то будет продолжать работать после таких исключений, но полученные результаты будут не те, что нам всем бы хотелось.

Потомки класса `Error` — это фатальные ошибки JVM, с обработкой которых не может справиться выполняемая программа.

`TooManyBikesException` (слишком много велосипедов) — это пример исключения, которое может быть создано программистом.

Как программисту узнать заранее, что некий Java-метод некоего класса может вызвать исключение, и что необходимо использовать блок

try/catch? Не стоит беспокоиться! Если вы вызываете метод, который может привести к исключению, компилятор Java выдаст предупреждение об ошибке, вроде такого:

```
"ScoreReader.java":  unreported exception:
java.io.IOException; must be caught or declared to be
thrown at line 57
```

Дополнительную информацию о том, какие исключения могут быть вызваны теми или иными методами, можно найти в документации по Java. В оставшейся части главы вы научитесь обращаться с этими исключениями.

Блок try/catch

Для обработки ошибок в Java могут быть использованы пять ключевых слов: try, catch, finally, throw и throws.

После одного блока try можно поставить несколько блоков catch, если предполагается, что может произойти более чем одна ошибка. Например, когда программа пытается прочитать файл, его может не оказаться на месте, и возникнет исключение FileNotFoundException. Или же, если файл найден, но программа продолжает считывать после конца этого файла, вы получите исключение EOFException.

Следующий фрагмент кода выведет сообщение на русском языке, если программа не сможет найти файл со счётом игры или достигнет конца файла. Для других ошибок чтения код выведет сообщение *Проблема при чтении файла* и техническое описание проблемы.

```
public void getScores(){
 try{
    fileScores.read();
    System.out.println("Счёт игры успешно загружен");
 }catch(FileNotFoundException e){
    System.out.println("Не найден файл Scores");
 }catch(EOFException e1){
    System.out.println("Достигнут конец файла");
 }catch(IOException e2){
    System.out.println("Проблема при чтении файла " +
e2.getMessage());
 }
}
```

Если выполнение метода read() прервется, программа перепрыгнет через строчку с вызовом println() прямо в блок catch для подходящей ошибки. Если такой блок найден, то выполнится соответствующий println(), а если подходящий блок catch найти не удастся, метод getScores() перенаправит это исключение методу, его вызвавшему.

Если вы пишете несколько блоков catch, вам следует располагать их в порядке, согласно тому, как соответствующие исключения унаследованы друг от друга. Например, так как EOFException — это подкласс IOException, нужно расположить блок catch для подкласса вначале. Если же первым поместить catch для IOException, тогда программа никогда не достигнет FileNotFound или EOFException, так как первый catch будет их перехватывать.

Начиная с Java 7, можно отлавливать сразу несколько исключений в одном блоке catch:

```
public void getScores(){
 try{
    fileScores.read();
    System.out.println("Счёт игры успешно загружен");
 }catch(FileNotFoundException | EOFException |
                                            IOException e){

    System.out.println("Проблема при чтении файла " +
                                          e.getMessage());
 }
}
```

Лентяи запрограммируют метод getScores() вот так:

```
public void getScores(){

 try{

   fileScores.read();

 }catch(Exception e){

   System.out.println(("Проблема при чтении файла " +
                                            e.getMessage());
 }
}
```

Это пример плохого стиля программирования. При написании программы, всегда нужно помнить, что кто-то другой может захотеть её прочитать, и вам может быть стыдно за свой код.

Блок catch получает объект типа Exception, который содержит короткое описание возникшей проблемы, а его метод getMessage() возвращает это описание. Иногда, если описание ошибки не до конца понятно, попробуйте использовать метод toString():

```
catch(Exception e){
 System.out.println("Проблема при чтении файла " + e.toString());
}
```

Если нужна более детальная информация об исключении, используйте метод printStackTrace(). Он выведет последовательность вызовов, которая привела к возникновению исключения, такую же, как в примере из раздела *Чтение трассировки стека*.

Давайте попробуем «убить» программу–калькулятор из главы 6. Запустите класс Calculator и введите с клавиатуры *abc*. Нажмите любую из кнопок арифметических действий и вы получите на экране консоли что-то вроде этого:

```
java.lang.NumberFormatException: For input string: "abc" at
java.lang.NumberFormatException.forInputString(NumberFormat
Exception.java:48)

	at
java.lang.FloatingDecimal.readJavaFormatString(FloatingDeci
mal.java:1213)

	at java.lang.Double.parseDouble(Double.java:202)

	at
CalculatorEngine.actionPerformed(CalculatorEngine.java:27)

	at
javax.swing.AbstractButton.fireActionPerformed(AbstractButt
on.java:1764)
```

Это был пример реакции программы в случае необработанного исключения. В методе actionPerformed() класса CalculatorEngine есть следующая строчка:

```
displayValue= Double.parseDouble(dispFieldText);
```

Если переменной dispFieldTest присвоено не числовое значение, метод parseDouble() не сможет конвертировать его в значение типа double и вызовет исключение NumberFormatException.

Давайте обработаем это исключение и отобразим сообщение об ошибке, которое объяснит ситуацию пользователю. Строка, содержащая `parseDouble()` должна быть помещена в блок `try/catch`, и Eclipse в этом поможет. Выделите эту строку и щелкните на ней правой кнопкой мыши. В появившемся меню выберите пункт *Source and Surround with try/catch block.* Вуаля! Код изменился:

```
try {
   displayValue= Double.parseDouble(dispFieldText);
} catch (NumberFormatException e1) {
   // TODO Auto-generated catch block
   e1.printStackTrace();
}
```

Замените строку с `printStackTrace()` на следующий код:

```
javax.swing.JOptionPane.showConfirmDialog(null,
       "Пожалуйста, введите число", "Неправильный ввод",
                 javax.swing.JOptionPane.PLAIN_MESSAGE);

return;
```

Так мы избавились от устрашающей трассировки стека в сообщении об ошибке и теперь отображаем простое для понимания сообщение *Пожалуйста, введите число*:

Теперь исключение `NumberFormatException` будет обрабатываться.

Ключевое слово throws

В некоторых случаях, более целесообразно обрабатывать исключение не в том методе, где оно возникло, а в том, который его вызвал. В таких

случаях, в описании метода необходимо объявить (предупредить), что он может вызывать некоторое исключение. Это делается с помощью специального ключевого слова `throws`. Давайте используем тот же пример с чтением файла.

Так как метод `read()` может вызывать исключение `IOException`, вы должны или обрабатывать его, или объявить его в описании метода. В следующем примере мы объявим, что метод `getAllScores()` может вызывать исключение `IOException`:

```
class MySuperGame{

  void getAllScores() throws IOException{

   // Не используйте try/catch, если вы не
   // обрабатываете исключения внутри этого метода

   file.read();

  }

  public static void main(String[] args){

    MySuperGame msg = new MySuperGame();
    System.out.println("Список результатов игры");

    try{

     // Так как getAllScores() объявляет исключение,
     // мы обрабатываем его здесь
     msg.getAllScores();

    }catch(IOException e){

       System.out.println(
        "Извините, список результатов игры недоступен");
    }
}
```

Так как мы даже не пытаемся перехватывать исключения в методе `getAllScores()`, исключение `IOException` *перенаправится* из него в вызвавший его метод `main()`. Теперь это исключение должен обработать главный метод.

Ключевое слово finally

Выполнение любого кода внутри блока `try/catch` может закончиться одним из следующих вариантов:

- ✓ Выполнение кода в блоке `try` закончилось успешно, и программа продолжает работу.
- ✓ Код внутри блока `try` достиг выражения `return` и вышел из метода.
- ✓ Код внутри блока `try` вызвал исключение, и контроль перешел к соответствующему блоку `catch`, который или обрабатывает ошибку, или перенаправляет исключение к методу, вызвавшему текущий.

Если есть код, который необходимо выполнить в любом случае, поместите его после ключевого слова `finally`:

```
try{
      file.read();

}catch(Exception e){

      printStackTrace();

}finally{

  // код, который всегда должен выполняться
  // помещается сюда, например file.close();

}
```

Если написать код для закрытия файла в блок `finally`, то он выполнится независимо от того, прошла ли операция чтения успешно или нет. Обычно, в блок `finally` помещается код, который освобождает какие-либо ресурсы компьютера, например, производит отключение от сети или закрытие файла.

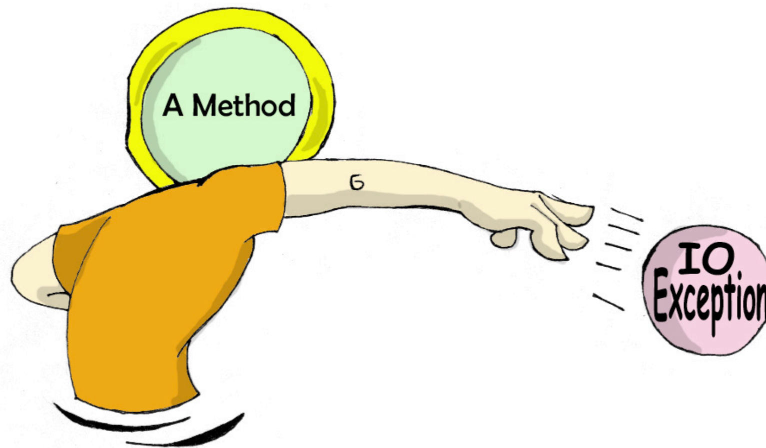
Если вы не планируете обрабатывать исключения в текущем методе, они будут переданы в вызвавший метод. В таком случае, вы можете использовать ключевое слово `finally` даже без блока `catch`:

```
void myMethod () throws IOException{

 try{

   // поместите сюда код, который считывает файл
 }

 finally{

   // поместите сюда код, который закрывает файл
 }
}
```

Ключевое слово throw

Если в методе возникло исключение, но его должен обработать вызвавший метод, просто перенаправьте его этому методу. Иногда вам может понадобится перехватить одно исключение, а отправить дальше другое, с другим описанием ошибки, как в приведенном далее коде.

Выражение `throw` (бросать) предназначено для "бросания" (отправления) Java-объектов. Объекты, которые бросаются программой, должны быть *бросаемыми*. Это означает, что вы можете бросать только объекты, прямо или косвенно унаследованные от класса `Throwable`. Все исключения в Java являются потомками этого класса.

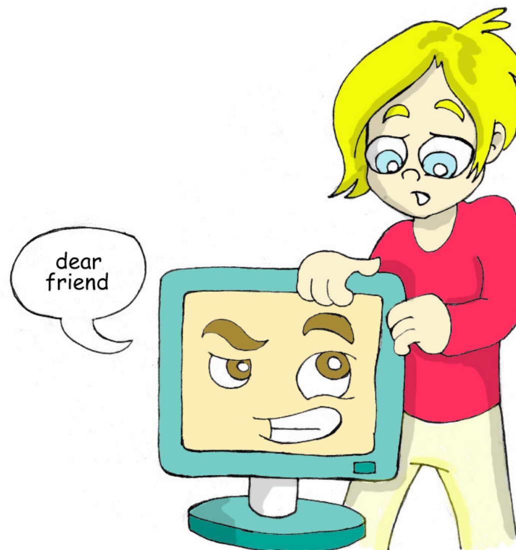


В следующем фрагменте кода метод `getAllScores()` перехватывает исключение `IOException`, создает новый объект типа `Exception` с более понятным описанием ошибки и отправляет его в метод `main()`. Теперь, метод `main()` не скомпилируется, если вы не поместите строки с вызовом метода `getAllScores()` в блок `try/catch`, так как этот метод может вызывать исключение `Exception`, которое должно быть, либо обработано, либо перенаправлено дальше. Метод `main()` не должен создавать никаких исключений, поэтому должен обрабатывать их.

```
class ScoreList{

  // Для компиляции этого класса нужен дополнительный код
```

```
  static void getAllScores() throws Exception{

     try{

      file.read();//эта строка может вызвать исключение

     } catch (IOException e) {

         throw new  Exception (
           "Дорогой друг, в файле Scores есть проблемы");

     }
}

  public static void main(String[] args){

     System.out.println("Scores");

     try{

         getAllScores();

     }catch(Exception e1){

         System.out.println(e1.getMessage());
     }
  }
}
```

В случае ошибки работы с файлом, главный метод обработает её, а метод `e1.getMessage()` выведет сообщение *Дорогой друг…*



Создание своих исключений

Программисты могут создавать классы исключений, которых, изначально, не было в Java. Такие классы должны быть унаследованы

от одного из существующих исключений. Предположим, вы занимаетесь продажей велосипедов и должны *проверять* заказы покупателей. В зависимости от моделей, в ваш грузовичок может поместиться разное количество велосипедов.

Например, вы можете загрузить в него не более трех велосипедов FireBird. Создайте подкласс класса Exception под названием TooManyBikesException. Теперь, если кто-то закажет более трех таких велосипедов, вызовите исключение:

```
class TooManyBikesException extends Exception{

  // Конструктор
  TooManyBikesException (){

    // Просто вызовите конструктор суперкласса
    // и передайте ему сообщение, которое нужно отобразить
    super(
"Мы не сможем доставить столько велосипедов за один раз.");

  }
}
```

Этот класс содержит только конструктор, получающий сообщение, которое описывает ошибку. Это сообщение конструктор передаёт для хранения суперклассу. Когда в блок catch попадает это исключение, можно узнать, что именно произошло, вызвав метод getMessage().

Представьте, что пользователь выбирает в окне OrderWindow несколько велосипедов и нажимает кнопку *Разместить заказ*. Как вы помните из шестой главы, это действие повлечет вызов метода actionPerformed(), который проверит, может ли заказ быть выполнен.

В следующем примере кода показано, как метод checkOrder() этого окна объявляет, что он может вызвать исключение TooManyBikesException. Если заказ не помещается в грузовик, этот метод вызывает исключение, которое затем перехватывается блоком catch. Сообщение об ошибке отображается в текстовом поле окна.

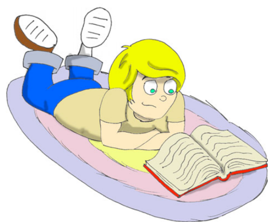
```
class OrderWindow implements ActionListener{

// Здесь нужно поместить код для создания компонентов окна.
// Пользователь нажал на кнопку Разместить заказ

   String selectedModel = txtFieldModel.getText();

   String selectedQuantity = txtFieldQuantity.getText();

   int quantity = Integer.parseInt(selectedQuantity);
```

```
   void actionPerformed(ActionEvent e){

    try{

      bikeOrder.checkOrder("FireBird", quantity);

   //следующая строка не выполняется в случае исключения
      txtFieldOrderConfirmation.setText(
                 "Размещение вашего заказа завершено");

    } catch(TooManyBikesException e){

      txtFieldOrderConfirmation.setText(e.getMessage());

    }
  }

   void checkOrder(String bikeModel, int quantity)

                              throws TooManyBikesException{

//Напишите код, который проверяет, помещается ли требуемое
//количество велосипедов заданной модели в грузовик.
//Если не помещается, сделать следующее:

    throw new TooManyBikesException("Невозможно доставить"
       + quantity + " велосипедов модели " + bikeModel +
                              " за одну доставку" );

   }
}
```

В идеальном мире любая программа должна работать без проблем. В реальности мы должны быть готовы к неожиданным ситуациям. То, что Java заставляет писать код, готовый к таким ситуациям, действительно полезно.

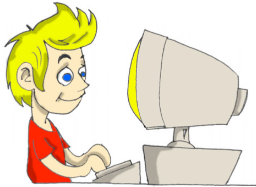
Материалы для дополнительного чтения

Обработка ошибок с помощью исключений:



http://download.oracle.com/javase/tutorial/essential/exceptions/index.html

Практические упражнения



1. Создайте Swing-приложение для размещения заказов на покупку велосипедов. Оно должно содержать два текстовых поля *Модель велосипеда* и *Количество*, кнопку *Разместить заказ* и сообщение для подтверждения заказа.

2. Используйте код из примеров с `OrderWindow` и `TooManyBikesException`. Попробуйте несколько комбинаций моделей велосипедов и количества, которые вызывали бы исключение.

Практические упражнения для умников и умниц

1. Измените приложение из предыдущего задания, заменив текстовое поле *Модель велосипеда* выпадающим меню с несколькими моделями, чтобы пользователь мог выбрать из списка, вместо того, чтобы вводить с клавиатуры.

2. Прочитайте в Интернете про Swing-компоненты `JComboBox` и `ItemListener` для обработки событий, когда пользователь выбирает модель велосипеда.

Глава 9. Сохранение счёта игры

После того, как программа заканчивает работу, она выгружается из его оперативной памяти. Это означает, что все классы, методы и переменные перестают существовать до тех пор, пока программа не будет запущена снова. Чтобы сохранить результаты работы программы, их нужно записать на жесткий диск, флэшку, или другое устройство, способное хранить данные долгое время. В этой главе вы узнаете, как работать с данными на диске, используя *потоки (streams)*.

Попросту говоря, открывается поток между программой и диском (или другим устройством, даже удаленным). Если нужно считать с диска, это должен быть *поток ввода*, если же необходимо записать данные на диск, то *поток вывода*. Например, если игрок завершает игру и необходимо запомнить его результат, можно записать его в файл под названием *scores.txt*, используя поток вывода.

Программа считывает и записывает данные в поток *последовательно*, байт за байтом, символ за символом и т.д. Программа может использовать разные типы данных, например, `String`, `int`, `double` и другие, поэтому необходимо выбирать соответствующий тип потока Java, например, поток байтов, поток символов, поток данных.

Классы для работы с потоками находятся в пакетах `java.io.` и `java.nio.`

Независимо от того, какой тип файловой системы будет использован, необходимо произвести три действия:

- ✓ Открыть поток, указывающий на некоторый файл.
- ✓ Считать или записать данные в этот поток.
- ✓ Закрыть поток.

Байтовые потоки

При написании программы, которая считывает данные из файла, а затем отображает их на экране, необходимо знать, данные какого типа в этом файле содержатся. С другой стороны, программа, которая просто копирует файлы из одного места в другое, может даже не знать, что в них - изображения, текст или же музыка. Такая программа считывает первоначальный файл в оперативную память, а затем записывает его в папку назначения байт за байтом, используя классы Java `FileInputStream` или `FileOutputStream`.

Следующий пример показывает, как использовать класс `FileInputStream` для чтения графического файла *abc.gif* из директории *c:\practice*. В операционной системе Microsoft Windows, для избежания путаницы со специальными символами Java, которые начинаются с обратного слэша, для разделения названий директорий и файлов следует использовать двойной слэш: *c:\\practice*. Эта небольшая программа выводит на экран не изображение, а всего лишь цифры, показывающие, в каком закодированном виде оно хранится на диске. Каждому байту соответствует положительное значение от 0 до 255. Класс `ByteReader` выводит эти значения, разделенные пробелом.

Обратите внимание, что класс `ByteReader` закрывает поток в блоке `finally`. Никогда не вызывайте метод `close()` внутри блока `try/catch` сразу после завершения чтения из файла, делайте это в блоке `finally`. В случае возникновения исключения при чтении файла, выполнение перешагнет через вычеркнутый вызов `close()` и поток не будет закрыт! Чтение прекращается, когда метод `FileInputStream.read()` возвращает негативное значение.

```
import java.io.FileInputStream;
import java.io.IOException;

public class ByteReader {

  public static void main(String[] args) {

   FileInputStream myFile = null;

   try {

      // Открытие байтового потока, указывающего на файл

        myFile = new FileInputStream("c:\\temp\\abc.gif");
```

```
        while (true) {
          int intValueOfByte = myFile.read();
          System.out.print(" " + intValueOfByte);
            if (intValueOfByte  == -1){
             // достигнут конец файла нужно выйти из цикла
               break;
            }
        } // конец цикла while
          // myFile.close(); не помещайте этот вызов здесь
      } catch (IOException e) {
               System.out.println("Невозможно прочитать файл: "
                                               + e.toString());
      } finally{
          try{
               myFile.close();
          } catch (Exception e1){
                    e1.printStackTrace();
          }
          System.out.println(
                   "Чтение файла завершено успешно");
          }
     }
}
```

Следующий фрагмент кода записывает несколько байт, представленных целочисленными значениями, в файл *xyz.dat*, используя класс `FileOutputStream`:

```
int somedata[]= {56,230,123,43,11,37};
  FileOutputStream myFile = null;
  try {
      // Открывается файл xyz.dat, в который
      // записываются данные из массива
       myFile = new  FileOutputStream("xyz.dat");
       for (int i = 0; i <some data.length; i++){
            file.write(data[i]);
```

```
        }
 } catch (IOException e) {
   System.out.println("Невозможно записать данные в файл: "+
                                                   e.toString());
  } finally{
        try{
         myFile.close();
        } catch (Exception e1){
              e1.printStackTrace();
        }
  }
```

Буферизированные потоки

Итак, сейчас мы считываем и записываем файлы побайтно. Это означает, что для чтения файла длиной в 1000 байт, программа `ByteReader` должна будет обратиться к диску 1000 раз. Но работа с данными на диске происходит значительно медленнее, чем в оперативной памяти. Для минимизации количества обращений к диску, Java предоставляет так называемые буферы, нечто вроде "резервуаров для данных".

Класс `BufferedInputStream` позволяет быстро заполнить буфер в памяти данными из `FileInputStream`. Буферизированный поток считывает большую порцию байтов из файла в память за раз, после чего метод `read()` получает отдельные байты из буфера намного быстрее.

В программе можно соединять потоки, так же, как водопроводчик соединяет трубы. Давайте немного изменим пример, который считывает файл. Сначала, данные из потока `FileInputStream` будут поступать в `BufferedInputStream`, а потом — методу `read()`:

```
  FileInputStream myFile = null;
 BufferedInputStream buff =null;
 try {
     myFile = new  FileInputStream("abc.dat");
```

```
        // соединяем потоки FileInputStream и BufferedInputStream

        buff = new BufferedInputStream(myFile);

        while (true) {

              int byteValue = buff.read();
              System.out.print(byteValue + " ");

              if (byteValue  == -1)
                  break;
      }

  } catch (IOException e) {

        e.printStackTrace();

  }finally{

       try{

        buff.close();
        myFile.close();

       } catch(IOException e1){

           e1.printStackTrace();
       }
  }
```

А какого объема такой буфер? Это зависит от версии JVM, например 600 байт, но его можно установить и вручную. Например, для установки размера буфера в 5000 байтов, используйте конструктор с двумя аргументами:

```
BufferedInputStream buff = new BufferedInputStream(myFile, 5000);
```

Буферизированные потоки не изменяют способ чтения, а просто ускоряют его.

`BufferedOutputStream` действует аналогично, но только он не читает, а пишет в буфер, используя класс `FileOutputStream`.

```
int somedata[]= {56,230,123,43,11,37};

  FileOutputStream myFile = null;
  BufferedOutputStream buff =null;

  try {

        myFile = new  FileOutputStream("abc.dat");

        // соединяем потоки
        buff = new BufferedOutputStream(myFile);

        for (int i = 0; i <somedata.length; i++){
```

```
            buff.write(somedata[i]);

         }

   } catch (IOException e) {

         e.printStackTrace();

   }finally{

          try{

               buff.flush();
               buff.close();
               myFile.close();

          } catch(IOException e1){
               e1.printStackTrace();
          }
   }
```

Чтобы быть уверенным, что все байты из буфера были переданы в файловый поток, после окончания записи в `BufferedOutputStream`, вызовите метод `flush()`.

Аргументы командной строки

Программа `ByteReader` хранит имя файла abc.gif прямо в своем коде, или, как говорят программисты, имя файла *жёстко зашито* (на сленге *захардкодировано*) в программу. Это означает, что для получения похожей программы, которая бы считывала файл xyz.gif, нужно было бы изменить код и перекомпилировать его, что не очень удобно. Было бы намного лучше передавать имя файла в командной строке, при запуске программы.

Любую Java-программу можно запустить с аргументами командной строки, например:

```
java ByteReader xyz.gif
```

Здесь методу main() передается всего один аргумент — xyz.gif. Если помните, у метода `main()` есть аргумент:

```
public static void main(String[] args) {…}
```

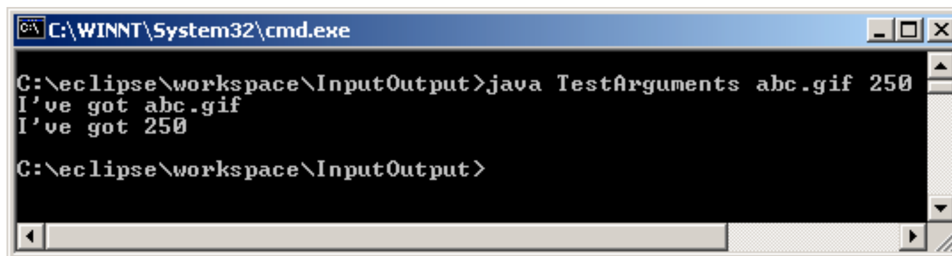
Да, это массив элементов типа `String`, который JVM передает в главный метод. Если запустить программу без аргументов, этот массив

останется пустым. Иначе, массив будет иметь столько элементов, сколько аргументов командной строки было передано программе.

Давайте посмотрим, как эти аргументы командной строки можно использовать в очень простом классе, который просто выведет их на экран:

```
public class TestArguments {

      public static void main(String[] args) {

        // Сколько получено аргументов?
        int numberOfArgs = args.length;

        for (int i=0; i<numberOfArgs; i++){

          System.out.println("I've got " + args[i]);

        }
      }
}
```

На следующем скриншоте показано, что произойдет, если запустить программу с двумя аргументами — xyz.gif и 250. JVM поместит значение xyz.gif в элемент `args[0]`, а второй аргумент попадет в `args[1]`.



Аргументы командной строки всегда передаются в программу как строки (`String`). Программа самостоятельно должна конвертировать данные в подходящий формат, например:

```
int myScore = Integer.parseInt(args[1]);
```

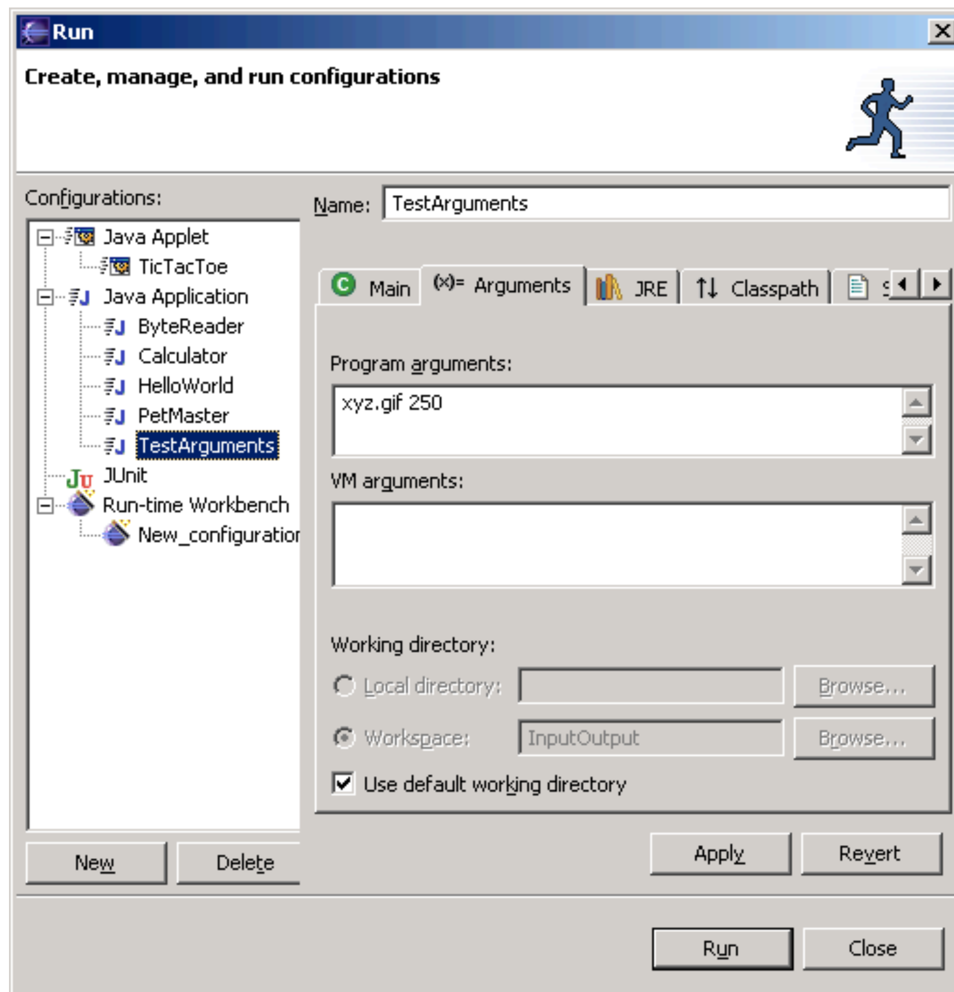
Советую проверять, правильное ли число аргументов передано в командной строке. Делайте это прямо в начале метода `main()`. Если количество аргументов не соответствует ожидаемому, то программа должна выдать короткое сообщение об этом и незамедлительно закончить работу, используя специальный метод `System.exit()`:

```
public static void main(String[] args) {

 if (args.length != 2){
```

```
  System.out.println(
            "Пожалуйста, передайте параметры, например:");

  System.out.println("java TestArguments xyz.gif 250");

  // Выход из программы
  System.exit(0);

 }
}
```

В конце этой главы вам предстоит написать программу для копирования файлов. Для того, чтобы программа работала с любыми файлами, имена исходного и конечного файлов должны будут передаваться в программу через аргументы командной строки.

В Eclipse, для целей тестирования, тоже можно указывать аргументы командной строки для всех запускаемых программ. В окне *Run Configurations*, выберите закладку с надписью `(x)=Arguments` и введите необходимые аргументы в поле *Program Arguments*.



Поле *VM arguments* позволяет указать аргументы для JVM. Это позволит, например, запросить больше памяти для выполнения программы, настроить саму JVM, и т.д. В разделе *Материалы для дополнительного чтения* есть ссылка на сайт с более подробным описанием этих параметров.

Чтение текстовых файлов

Java использует двухбайтовые символы для хранения букв, а классы `FileReader` и `FileWriter` предназначены для удобной работы с текстовыми файлами. Эти классы могут считывать файлы посимвольно, используя метод `read()`, или же построчно, с помощью метода `readLine()`. У классов `FileReader` и `FileWriter` также есть

аналоги, BufferedReader и BufferedWriter, позволяющие ускорить работу с файлами.

Следующий класс ScoreReader считывает содержимое файла scores.txt строка за строкой. Выполнение программы завершается, когда метод readLine() возвращает null, что указывает на конец файла.

Используя любой текстовый редактор, создайте файл *c:\scores.txt*, содержащий следующие четыре строки:

David 235

Brian 190

Anna 225

Zachary 160

Запустите программу ScoreReader, которая выведет содержимое этого файла на экран. Добавьте еще несколько строк с результатами игр. Теперь запустите программу снова, чтобы убедиться, что добавленные строки тоже будут выводиться на экран.

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class ScoreReader {

  public static void main(String[] args) {

      FileReader myFile = null;
      BufferedReader buff = null;

      try {

         myFile=new FileReader("c:\\scores.txt");
         buff = new BufferedReader(myFile);

         while (true) {

          // считывается строка из файла scores.txt
          String line = buff.readLine();

          // проверка достижения конца файла
          if (line == null) break;

          System.out.println(line);

        } // конец цикла while
```

```
        }catch (IOException e){

                 e.printStackTrace();

        } finally {

              try{

                 buff.close();
                 myFile.close();

              }catch(IOException e1){
                 e1.printStackTrace();

              }
       }

   } // конец метода main
}
```

Если программа должна записывать текстовые данные на диск, используйте один из перегруженных методов `write()` класса `FileWriter`. Эти методы позволяют записать в файл символ, строку `String` или целый массив символов.

У класса `FileWriter` есть несколько перегруженных (overloaded) конструкторов. Если открыть файл, задав только его имя, то он будет перезаписан заново при каждом запуске программы:

```
FileWriter fOut = new FileWriter("Scores.txt");
```

Если нужно дозаписать данные в уже существующий файл, используйте конструктор с двумя аргументами (`true` означает режим добавления, если файл существует):

```
FileWriter fOut = new FileWriter("Scores.txt", true);
```

В следующем примере, класс `ScoreWriter` записывает три строки из массива `scores` в файл c:\scores.txt.

```
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.IOException;

public class ScoreWriter {

      public static void main(String[] args) {

      FileWriter myFile = null;
      BufferedWriter buff = null;

      String[] scores = new String[3];
```

```
      // заполнение массива результатами игры

      scores[0] = "Mr. Smith 240";
      scores[1] = "Ms. Lee 300";
      scores[2] = "Mr. Dolittle 190";

      try {

        myFile = new FileWriter("c:\\scores2.txt");

        buff = new BufferedWriter(myFile);

        for (int i=0; i < scores.length; i++) {

         // запись строк из массива в файл scores2.txt
         buff.write(scores[i]);

         System.out.println("Записывается  " + scores[i] );

        }

       System.out.println("Запись файла завершена");

      }catch (IOException e){

              e.printStackTrace();

      } finally {

           try{

              buff.flush();
              buff.close();
              myFile.close();

           }catch(IOException e1){

                 e1.printStackTrace();
          }
      }

  } // конец метода main
}
```

Вывод программы будет выглядеть вот так:

```
Записывается  Mr. Smith 240

Записывается  Ms. Lee 300

Записывается  Mr. Dolittle 190

Запись файла завершена
```

Класс File

Класс `java.io.File` содержит множество удобных методов, которые позволяют переименовать файл, удалить файл, проверить существование файла и т.д. Предположим, программа сохраняет некоторые данные в файл и нужно выдать пользователю предупреждение, если такой файл уже существует. Для этого, необходимо создать экземпляр класса `File`, указав имя файла, а затем вызвать метод `exists()`. Если метод вернет `true`, значит файл abc.txt найден на диске и необходимо вывести предупреждение, иначе такой файл еще не существует:

```
File aFile = new File("abc.txt");

if (aFile.exists()){

   // Сюда идет код для вывода в консоль или через JOptionPane

   // для отображения предупреждения в окошке

}
```

Конструктор класса `File`*, на самом деле, не создает файл* — он просто создает в памяти экземпляр этого класса, который указывает на реальный файл. Если действительно нужно создать файл, используйте для этого метод `createNewFile()`.

Ниже приведены некоторые полезные методы класса `File`.

Имя метода	**Предназначение метода**
`createNewFile()`	Создает новый пустой файл с именем, указанным при создании объекта типа `File`. Новый файл создается только, если он еще не существует.
`delete()`	Удаляет файл или директорию
`renameTo()`	Переименовывает файл
`length()`	Возвращает размер файла в байтах
`exists()`	Возвращает `true`, если файл существует
`list()`	Возвращает массив строк с именами

	файлов/директорий, содержащихся в указанной директории
lastModified()	Возвращает время последнего изменения файла
mkDir()	Создает директорию

Следующий фрагмент кода переименовывает файл customers.txt в customers.txt.bak. Если файл .bak уже существует, он будет перезаписан.

```
File file = new File("customers.txt");
File backup = new File("customers.txt.bak");

if (backup.exists()){

        backup.delete();

}

file.renameTo(backup);
```

В версии Java 7 появился новый класс Files, который упрощает создание, удаление и копирование файлов. В отличие от класса File, Files создает или удаляет реальные файлы на диске, а не в памяти. А новый класс Path – это программное представление полного имени файла, вне зависимости от операционний системы пользователя. Вот как это выглядит для файла Customers.txt под Windows и на Маке:

```
 Path pathCustomers=
FileSystems.getDefault().getPath(".","c\\Customers.txt");

Path pathCustomers=
FileSystems.getDefault().getPath(".","/Customers.txt");
```

А так можно читать файл в коллекцию строк:

```
List customers=
       Files.readAllLines(pathCustomers,Charset.defaultCharset());
```

Прочесть файл, как набор байтов можно так:

```
byte[] customers=Files.readAllBytes(pathCustomers);
```

Все также можно использовать буфер:

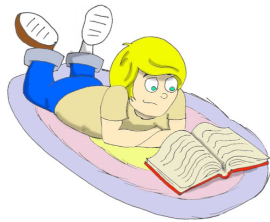
```
Reader reader=
    Files.newBufferedReader(pathCustomers, Charset.defaultCharset());
```

А создать новый файл *Customers.txt* можно так:

```
Path fileName=  Paths.get("c:\\Customers.txt");
Path customers=Paths.createFile(fileName);
```

Эта глава была посвящена только работе с файлами на диске вашего компьютера, но Java позволяет создавать потоки, указывающие на другие компьютеры в сети. Такие компьютеры могут находиться достаточно далеко друг от друга. Например, NASA использовала Java для управления марсоходами. Я уверен, что для этого они просто направили свои потоки на Марс. ☺

Материалы для дополнительного чтения

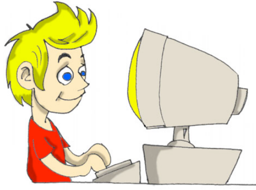


1. Параметры командной строки JVM

http://download.oracle.com/javase/tutorial/essential/environment/cmdLineArgs.html

2. Использование файловых потоков
http://java.sun.com/docs/books/tutorial/essential/io/filestreams.html

Практические упражнения



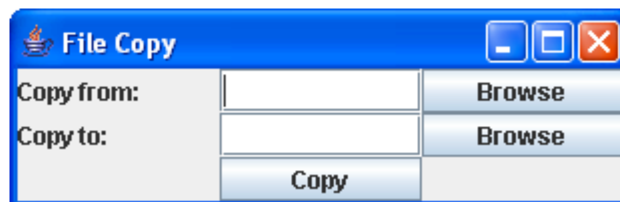
Напишите программу копирования файлов под названием `FileCopy`, объединив для этого фрагменты кода из раздела про байтовые потоки.

Откройте два потока (ввода и вывода) и вызовите методы `read()` и `write()` *в одном и том же цикле*. Используйте аргументы командной строки для передачи программе имён исходного и конечного файлов, например:

```
java FileCopy c:\\temp\\scores.txt
c:\\backup\\scores2.txt
```

Практические упражнения для умников и умниц

Создайте Swing–приложение, позволяющее пользователям выбирать имена файлов для копирования, используя класс `JFileChooser`, который создает стандартное окно выбора файла. Это окно должно открываться при нажатии пользователем одной из кнопок *Browse*. Нужно будет добавить несколько строчек кода для отображения выбранного имени файла в текстовом поле.



После того, как пользователь нажимает кнопку Copy, код в методе `actionPerformed()` должен скопировать выбранный файл. Попробуйте заново

использовать код из предыдущих заданий, но без прямого копирования/вставки.

Глава 10. Разные Полезные Штучки

У нас была возможность использовать разные элементы языка Java в предыдущих главах. Мы даже создали игру в крестики-нолики. Однако, я упустил некоторые важные приемы и элементы Java и самое время рассказать о них.

Работа с датами и временем

В каждом компьютере есть внутренние часы. Любая программа Java может узнать текущие дату и время, и отобразить их в различных форматах. Например, 15.06.2011 или 15 июля 2011. В Java есть множество классов, которые работают с датами. Но два из них — `java.util.Date` и `java.text.SimpleDateFormat` — охватывают большую часть ваших потребностей при работе с датами и временем.

Очень просто создать объект, который хранит текущую системную дату и время с точностью до миллисекунд:

```
Date today = new Date();
System.out.println( "Дата: " + today );
```

Выходные данные данного фрагмента кода могут выглядеть следующим образом:

```
Дата: Sat Oct 08 20:41:44 MSD 2011
```

Класс `SimpleDateFormat` позволяет отобразить дату и время в различных форматах. Первое, вам необходимо создать экземпляр данного класса с требуемым форматом. Затем вызвать у него метод `format()`, которому в качестве аргумента следует передать объект

Date. Следующая программа выполняет форматирование и печать текущей даты в нескольких форматах.

```
import java.util.Date;
import java.text.SimpleDateFormat;

public class MyDateFormat {

 public static void main( String [] args ) {

  // Создается объект Date
  // и выполняется печать в формате по умолчанию

  Date today = new Date();
  System.out.println( "Дата: " + today );

  // Формат, который выводит дату в виде 10-08-11
  SimpleDateFormat sdf= new SimpleDateFormat("MM-dd-yy");

  String formattedDate=sdf.format(today);

  System.out.println( "Дата(мм-дд-гг): " + formattedDate );

  // Формат, который выводит дату в виде 08-10-2011
  sdf = new SimpleDateFormat("dd-MM-yyyy");

  formattedDate=sdf.format(today);

  System.out.println( "Дата(дд-мм-гггг): " + formattedDate );

  // Формат, который выводит дату в виде Пт, окт 27, '11
  sdf = new SimpleDateFormat("EEE, MMM d, ''yy");

  formattedDate=sdf.format(today);

  System.out.println(

  "Дата (день недели, мес д, 'гг) "+ formattedDate);

  // Формат, который выводит время в виде 07:18:51 AM
  sdf = new SimpleDateFormat("hh:mm:ss a");

  formattedDate=sdf.format(today);

  System.out.println( "Время (чч:мм:сс) "+ formattedDate );

    }
}
```

Откомпилируйте и запустите класс MyDateFormat, в результате он выведет на экран что-то похожее на:

```
Дата: Sat Oct 08 20:54:37 MSD 2011

Дата(мм-дд-гг): 10-08-11

Дата(дд-мм-гггг): 08-10-2011
```

```
Дата (день недели, мес д, 'гг) Сб, окт 8, '11

Время (чч:мм:сс) 08:54:37 PM
```

В документации Java для класса `SimpleDateFormat` описаны другие форматы. Дополнительные методы, которые работают с датами, можно найти в другом классе Java с именем `java.util.Calendar`.

Перегрузка методов

Класс может содержать несколько методов с одинаковым именем, но имеющих различные списки аргументов. Такую возможность называют *перегрузкой методов (method overloading)*. Например, метод `println()` класса `System` может быть вызван с аргументами различного типа: `String`, `int`, `char` и другими.

```
System.out.println("Привет!");

System.out.println(250);

System.out.println('A');
```

Несмотря на то, что это выглядит как три вызова одного и того же метода `println()`, фактически же вызываются три различных метода. Вы можете спросить, почему бы не создать методы с разными именами, например, `printString()`, `printInt()`, `printChar()`? Одна из причин в том, что намного проще запомнить одно имя метода для печати, чем запоминать несколько имен. Есть и другие причины для использования перегрузки методов, но они немного более сложные, чтобы объяснять их тут. Эти причины (polymorfism) нужно рассматривать в более сложных учебниках.

Как вы помните, наш класс `Fish` из главы 4, содержал метод `dive()`, который принимал один аргумент:

```
public int dive(int howDeep)
```

Давайте создадим еще одну версию этого метода, которая не будет требовать аргументов. Этот метод заставит рыбку нырять на 5 метров, пока глубина погружения не превысит 100 метров. Новая версия класса `Fish` содержит `final` переменную `DEFAULT_DIVING` (глубина ныряния по умолчанию), значение которой будет равным пяти метрам.

Теперь класс Fish содержит два перегруженных метода `dive()`.

```
public class Fish extends Pet {

  int currentDepth=0;
  final int DEFAULT_DIVING = 5;

  public int dive(){

    currentDepth=currentDepth + DEFAULT_DIVING;

    if (currentDepth > 100){

        System.out.println("Я маленькая рыбка и " +

                        " не могу нырять глубже 100 метров");

        currentDepth=currentDepth - DEFAULT_DIVING;

    }else{

        System.out.println("Погружаюсь на " +

                              DEFAULT_DIVING + " м");

        System.out.println("Я на " + currentDepth +

                        " метров ниже уровня моря");

    }

            return currentDepth;
}

  public int dive(int howDeep){

    currentDepth=currentDepth + howDeep;

    if (currentDepth > 100){

         System.out.println("Я маленькая рыбка и " +

                          " не могу нырять глубже 100 метров");

         currentDepth=currentDepth - howDeep;

    }else{

       System.out.println("Погружаюсь на "+howDeep+" м.");

       System.out.println("Я на " + currentDepth +

                             " м. ниже уровня моря");

    }

       return currentDepth;

  }
```

```
      public String say(String something){
        return "Ты не знаешь, что рыбы не умеют говорить?";
      }

      // constructor
      Fish(int startingPosition){
            currentDepth=startingPosition;
      }
  }
```

Теперь класс FishMaster может вызывать любой из перегруженных методов dive():

```
public class FishMaster {

      public static void main(String[] args) {
            Fish myFish = new Fish(20);
            myFish.dive(2);
            myFish.dive();  // новый перегруженный метод
            myFish.dive(97);
            myFish.dive(3);
            myFish.sleep();
      }
}
```

Конструкторы также могут быть перегружены, но при создании объекта будет использоваться только один из них. JVM будет выполнять вызов конструктора с соответствующим списком аргументов. Например, если в класс Fish добавить конструктор по умолчанию (без аргументов), класс FishMaster сможет создать его экземпляр с помощью одного из следующих способов:

```
Fish myFish = new Fish(20);
```

или

```
Fish myFish = new Fish();
```

Чтение данных с клавиатуры

В этом разделе вы узнаете, как программа может печатать вопросы в командном окне и понимать ответы, которые пользователь вводит с клавиатуры. В этот раз мы удалим из класса FishMaster все жестко заданные значения, которые он передает классу Fish. Теперь программа будет задавать вопрос: «*На какую глубину?*», а рыба будет погружаться в соответствии с ответом пользователя.

Вы уже много раз пользовались *стандартным выводом данных* System.out. Между прочим, переменная out имеет тип java.io.OutputStream. Сейчас я вам объясню, как работать со *стандартным выводом данных* System.in. Как вы, наверное, догадались, переменная in имеет тип java.io.InputStream.

Следующая версия класса FishMaster выводит в системную консоль *строку ввода* и ожидает ответа от пользователя. После того, как пользователь введет один или несколько символов и нажмет клавишу *Enter*, JVM размещает эти символы в объект класса InputStream и передает их программе.

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class FishMaster {

  public static void main(String[] args) {

   Fish myFish = new Fish(20);
   String feetString="";

   int feet;

   // Создаем обработчик чтения входного потока InputStreamReader,
   // который подключен к System.in и передаем его буферизованному
   // обработчику чтения BufferedReader

      BufferedReader stdin = new BufferedReader
                               (new InputStreamReader(System.in));

   // Погружаемся несколько раз пока пользователь не нажмет
   //  клавишу "Q"

      while (true) {

        System.out.println("Готова к погружению. На какую глубину?");
```

```
        try {

            feetString = stdin.readLine();

            if (feetString.equals("Q")){

            // Выход из программы
              System.out.println("Пока!");
              System.exit(0);

            }else {

        // Преобразуем feetString в целое число и погружаемся
        // на глубину, которая определяется переменной feet

             feet = Integer.parseInt(feetString);
             myFish.dive(feet);

            }

         } catch (IOException e) {

                  e.printStackTrace();

         }
      } // Конец while
   } // Конец main
 }
```

Диалог между пользователем и программой FishMaster может выглядеть как:

```
Готова к погружению. На какую глубину?

14

Погружаюсь на 14 м.

Я на 34 м. ниже уровня моря

Готова к погружению. На какую глубину?

30

Погружаюсь на 30 м.

Я на 64 м. ниже уровня моря

Готова к погружению. На какую глубину?

Q

Пока!
```

Сначала класс `FishMaster` создает поток `BufferedReader`, который подключен к стандартному входному потоку `System.in`. После этого

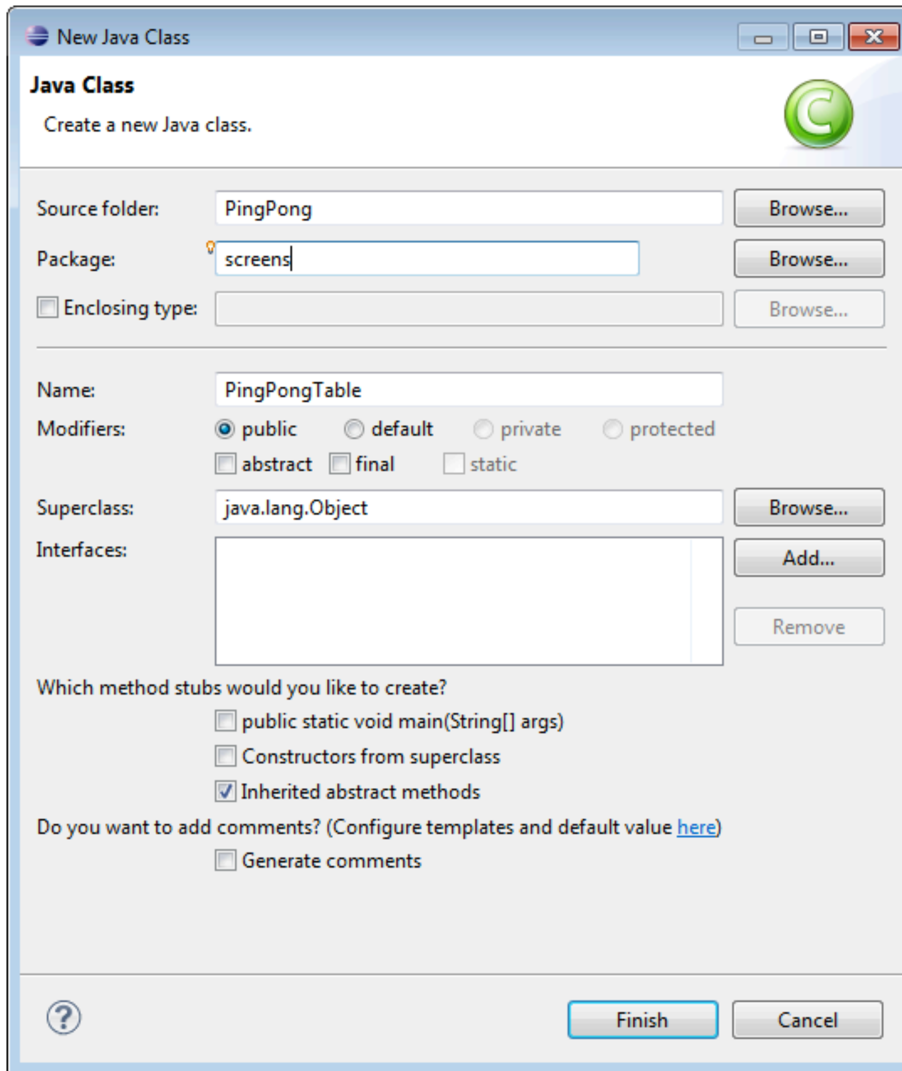
он отображает сообщение «*Готова к погружению. На какую глубину?*» и метод `readLine()` приостанавливает выполнение программы, пока пользователь не нажмет клавишу *Enter*. Введенное значение будет иметь тип `String`, поэтому класс `FishMaster` преобразует это значение в тип int и вызывает метод `dive()` класса `Fish`. Данное действие выполняется в цикле, пока пользователь не введет символ *Q*, чтобы выйти из программы. Строка `feetString.equals("Q")` сравнивает значение переменной `feetString` типа String и символ *Q*.

Чтобы полностью получить строку, введенную пользователем, за одну операцию мы использовали метод `readLine()`. Однако есть и метод `System.in.read()`, который позволяет обрабатывать данные, вводимые пользователем, посимвольно.

Тебе пакет

Когда программисты работают над большим проектом, который содержит огромное количество классов, они, как правило, группируют их в различных *пакетах*. Например, один пакет может содержать все классы, которые отображают окна (формы), а другой пакет может содержать обработчики событий. Язык Java также хранит свои классы в пакетах. Например, в пакете `java.io` содержатся классы, которые отвечают за обработку операций ввода/вывода. В пакете `javax.swing` содержатся классы графических компонентов Swing.

Давайте создадим в Eclipse новый проект под названием *PingPong*. Этот проект будет содержать классы в двух пакетах: `screens` и `engine`. Теперь создайте новый класс PingPongTable и в поле *Package* введите название пакета `screens`:

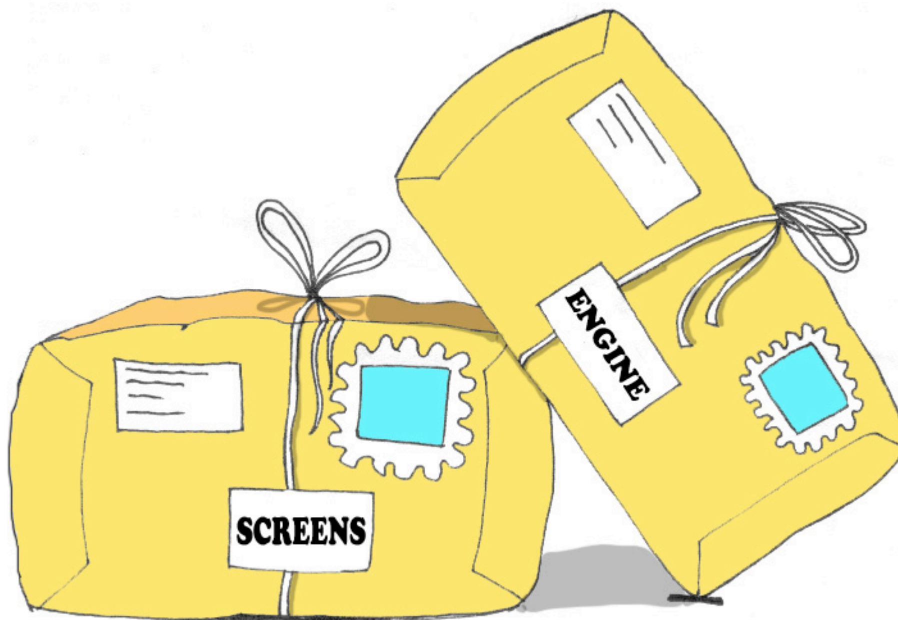


Нажмите кнопку *Finish* («Готово») и Eclipse сгенерирует код, который будет содержать строку с именем пакета.

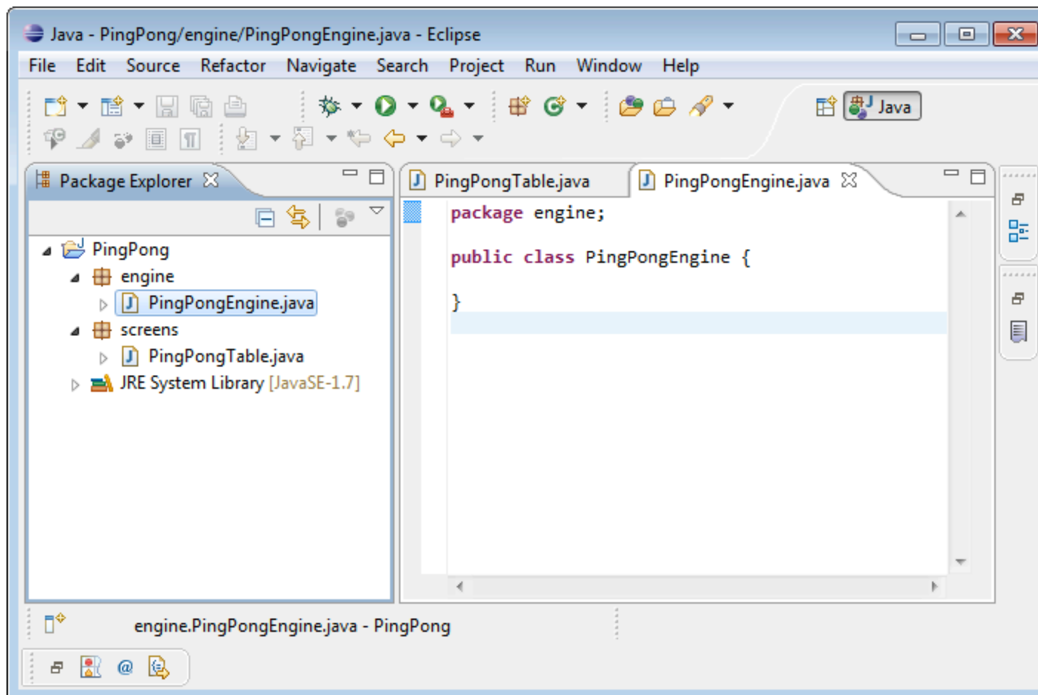
```
package screens;

public class PingPongTable {

	public static void main(String[] args) {

	}
}
```

Кстати, если в вашем классе есть строка с ключевым словом `package`, выше этой строки разрешается размещать только комментарии и ничего более.

Так как каждый пакет хранится в отдельном каталоге на диске, Eclipse создаст каталог screens и разместит в нем файл *PingPongTable.java*. Проверьте — на диске должен быть каталог *c:\eclipse\workspace\PingPong\screens* с файлами *PingPongTable.java* и *PingPongTable.class*.



Теперь создайте другой класс с названием `PingPongEngine`, а в качестве имени пакета введите `engine`. Проект `PingPong` теперь содержит два пакета:



Так как наши два класса расположены в двух разных пакетах (и, соответствтнно, директориях), класс `PingPongTable` не будет видеть `PingPongEngine`, пока не добавить в него выражение `import`.

```
package screens;

import engine.PingPongEngine;

public class PingPongTable {

  public static void main(String[] args) {

     PingPongEngine gameEngine = new PingPongEngine();
  }
}
```

Пакеты Java помогают не только структурировать ваши классы. Их можно использовать для ограничения доступа к классам в пакете для внешних классов, которые располагаются в других пакетах.

Уровни доступа

Классы Java, методы и переменные класса могут иметь следующие уровни доступа: `public`, `private`, `protected` и *package*. Наш

класс PingPongEngine имеет уровень доступа public. Это значит, что у любого класса есть доступ к нему. Давайте проведем простой эксперимент — удалим ключевое слово public из объявления класса PingPongEngine. Теперь класс PingPongTable не будет компилироваться, указывая на ошибки *PingPongEngine cannot be resolved to a type* (Невозможно определить тип PingPongEngine) и *The type engine.PingPongEngine is not visible* (Тип данных engine.PingPongEngine невидим). Это значит, что класс PingPongTable *не видит* больше класс PingPongEngine.

Если уровень доступа не указывается явно, то подразумевается уровень доступа *package*. Это значит, что класс будет доступен только для классов, которые находятся в одном с ним пакете (директории).

Точно также, если вы забудете дать доступ к методам класса PingPongEngine, класс PingPongTable так же укажет вам на это, сообщив, что эти методы для него невидимы. В следующей главе в процессе создания игры в пинг-понг вы увидите, как используются уровни доступа.

Уровень доступа private используется для сокрытия методов или переменных класса от внешнего мира. Представьте себе автомобиль. Большая часть людей понятия не имеют, какое количество деталей находится у него под капотом, а также о том, что реально происходит, когда водитель нажимает на педаль тормоза.

Взгляните на следующий фрагмент кода. В языке Java мы можем сказать, что объект Car *показывает* только один public метод — brake(), внутри которого могут быть вызваны несколько других методов, знать о которых водителю нет никакой необходимости.

Например, если водитель чересчур сильно нажимает на педаль тормоза, компьютер автомобиля может включить специальные антиблокировочные тормоза. Я уже упоминал ранее, что программы на Java управляют такими сложными роботами, как марсианские вездеходы, не говоря уже о простых автомобилях.

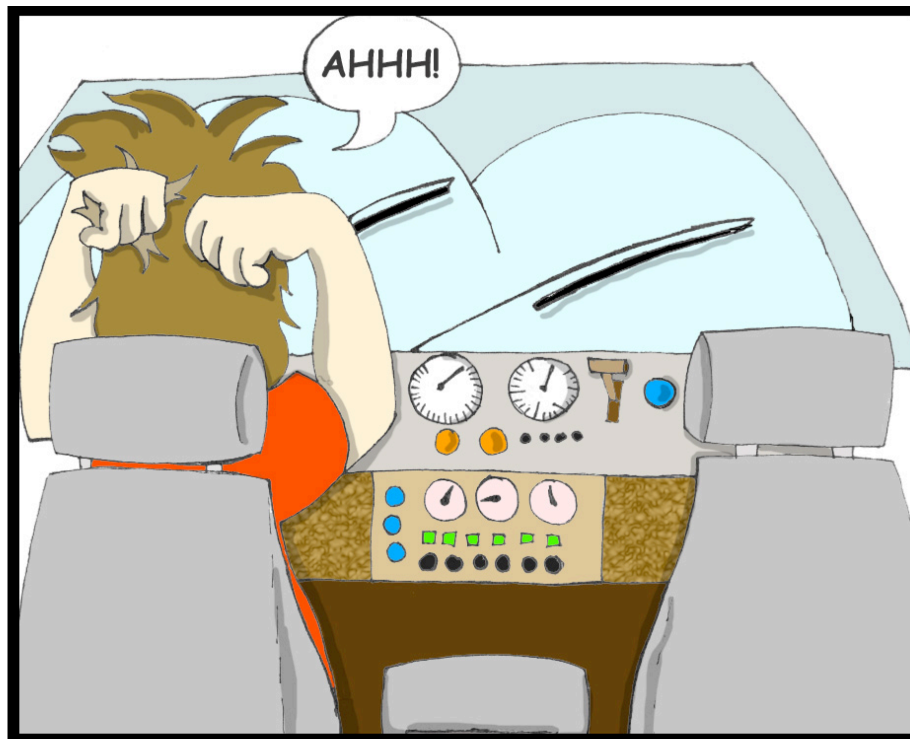
```
public class Car {

 // Эта private переменная может использоваться
 // только внутри класса
 private String brakesCondition;

 // public метод brake() вызывает private методы,
 // чтобы решить, какие тормоза использовать
 public void brake(int pedalPressure){

    boolean useRegularBrakes;

    useRegularBrakes=checkForAntiLockBrakes (pedalPressure);

    if (useRegularBrakes==true){

      useRegularBrakes();

    }else{

      useAntiLockBrakes();

    }
  }

  // Этот private метод, проверяющий тормоза с авто-блокировкой
  // может быть вызван только внутри этого класса
  private boolean checkForAntiLockBrakes(int pressure){

    if (pressure > 100){

        return true;

    }else {

        return false;

    }
  }



  // Этот private метод может быть вызван
  // только внутри этого класса
   private void useRegularBrakes(){

   // здесь будет код, который посылает сигнал обычным тормозам

  }
```

```
  // Этот private метод может быть вызван
  // только внутри этого класса
  private void useAntiLockBrakes(){

   // код, который посылает сигнал антиблокировочным тормозам   }

}
```

Есть еще одно ключевое слово в Java — `protected` — которое также устанавливает уровень доступа. Если вы используете это ключевое слово в сигнатуре метода, то этот метод будет доступен в классах-наследниках, а также в других классах, которые расположены в этом же пакете. Однако он не будет доступен для классов, которые находятся в других пакетах.

Одно из важных свойств объектно-ориентированных называется ***инкапсуляцией***. Это означает способность прятать и защищать элементы класса от доступа из других классов.

В процессе разработки класса скрывайте методы и переменные класса, которые не должны быть видимы во внешнем мире. Если конструктор автомобиля не скроет управление частью внутренних операций, то водитель столкнется с необходимостью взаимодействия с сотнями кнопок, переключателей и приборов.



В следующем разделе вы сможете найти класс `Score`, который скрывает свои свойства в переменных с доступом `private`.

Возвращаемся к массивам

В главе 9 программа `ScoreWriter` создавала массив объектов `String` и сохраняла имена и очки игроков в файл. Настало время узнать, как использовать массивы для хранения любых объектов.

В этот раз для представления счета в игре мы создадим объект. Этот объект будет содержать такие атрибуты, как имя и фамилия игрока, счет и последнюю дату игры.

Ниже представлен класс `Score`. Он содержит специальные методы для чтения (*getter*) и записи (*setter*) каждого из своего атрибутов, который объявлен с модификатором доступа *private*. Наверное, может показаться неочевидным, почему вызывающий класс просто не устанавливает значение атрибута напрямую, например, так:

```
Score.score = 250;
```

Вместо этого он это делает так:

```
Score.setScore(250);
```

А что, если позже мы решим, что наша программа должна проигрывать мелодию тогда, когда игрок достигает счета в 500 очков? Если в классе `Score` есть метод `setScore()`, все что нужно сделать, это изменить только этот метод. Нужно добавить в него код, который проверяет количество очков и при необходимости проигрывает мелодию.

Вызывающий класс продолжит вызывать музыкальную версию метода `setScore()` точно таким же образом. Если вызывающий класс будет устанавливать счет напрямую, то все «музыкальные» изменения нужно будет реализовать в вызывающем классе. А если потребуется использовать класс в двух разных игровых программах? В случае прямого изменения значений атрибутов потребуется реализовать эти изменения в двух вызывающих классах. Но, если бы у вас был setter-метод, то изменения были бы *инкапсулированы* в нем и вызывающие классы менять не нужно было бы.

```
import java.util.Date;
```

```
public class Score {

  private String firstName;
  private String lastName;
  private int score;
  private Date playDate;

  public String getFirstName(){

      return firstName;

  }

  public void setFirstName(String firstName){

      this.firstName = firstName;

  }

  public String getLastName(){

        return lastName;

  }

  public void setLastName(String lastName){

        this.lastName = lastName;

  }

  public int getScore(){

        return score;

  }

  public void setScore(int score){

      this.score=score;

  }

  public Date getPlayDate(){

        return playDate;

  }

  public void setPlayDate(Date playDate){

      this.playDate=playDate;

  }

// Объединяем все атрибуты в строку (String)
// и в конце добавляем символ перевода на новую строку.
// Этот метод удобно использовать, если вызывающий класс
```

```
// хочет за один раз распечатать все значения, например
// System.out.println(myScore.toString());

  public String toString(){

      String scoreString = firstName + " " +
         lastName + " " +  score + " " + playDate +
        System.getProperty("line.separator");

      return scoreString;
  }
}
```

Программа ScoreWriter2 создаст экземпляры объекта Score и назначит значения атрибутам этих экземпляров.

```
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.IOException;
import java.util.Date;

public class ScoreWriter2 {

/** Метод main выполняет следующие действия:

 1. Создает экземпляр массива
 2. Создает объекты Score и заполняет ими массив
 3. Записывает счет игры в файл
*/

   public static void main(String[] args) {

      FileWriter myFile = null;
      BufferedWriter buff = null;

      Date today = new Date();
      Score scores[] = new Score[3];

      // The player #1
      scores[0]=new Score();
      scores[0].setFirstName("Николай");
      scores[0].setLastName("Смирнов");
      scores[0].setScore(250);
      scores[0].setPlayDate(today);

      // The player #2
      scores[1]=new Score();
      scores[1].setFirstName("Анна");
      scores[1].setLastName("Егорова");
      scores[1].setScore(300);
      scores[1].setPlayDate(today);

      // The player #3

      scores[2]=new Score();
      scores[2].setFirstName("Сергей");
```

```
      scores[2].setLastName("Данилов");
      scores[2].setScore(190);
      scores[2].setPlayDate(today);

   try {

      myFile = new FileWriter("c:\\scores2.txt");
      buff = new BufferedWriter(myFile);

      for (int i=0; i < scores.length; i++) {

       // Преобразует каждый счет в объект String
       // и записывает его в scores2.txt
       buff.write(scores[i].toString());
       System.out.println("Запись  " +
                         scores[i].getLastName() );
      }

    System.out.println("Запись файла завершена");

  }catch (IOException e){

     e.printStackTrace();

  } finally {

     try{

        buff.flush();
        buff.close();
        myFile.close();
     }catch(IOException e1){

        e1.printStackTrace();
     }
   }
  }
}
```

Если программа пытается обратиться к элементу массива, который находится за его границами, например, `scores[5].getLastName()`, то Java вызывает исключение `ArrayIndexOutOfBoundsException`.

Класс ArrayList

Пакет `java.util` содержит классы, которые весьма удобно использовать в случае, если программе требуется хранить несколько экземпляров (*коллекцию*) неких объектов в памяти. Вот некоторые из популярных классов коллекций: `ArrayList`, `HashTable`, `HashMap` и `List`. Я покажу вам, как использовать класс `java.util.ArrayList`.

Недостатком обычного массива является то, что вам заранее нужно знать число элементов в массиве. Помните, чтобы создать экземпляр массива, нужно указать число между квадратными скобками:

```
String[] myFriends = new String[5];
```

Класс `ArrayList` не имеет такого ограничения — можно создать экземпляр этой коллекции, не зная о том, сколько объектов будет в нем. Просто добавляйте столько элементов, сколько требуется.

Зачем тогда использовать массивы? Давайте всегда использовать класс `ArrayList`! К сожалению, бесплатный сыр бывает только в мышеловке. За удобство нужно заплатить — `ArrayList` работает несколько медленнее, чем обычный массив. Кроме того, в нем можно хранить только объекты, но нельзя просто так сохранить набор чисел с типом `int` (их нужно заворачивать в классы-обертки или пользоваться так называемым автобокхингом - autoboxing).

Чтобы создать и заполнить объект `ArrayList`, сначала нужно создать его экземпляр. Далее создать экземпляры объектов, которые планируете в нем хранить и добавить их в `ArrayList`, с помощью его метода `add()`. Ниже представлена небольшая программа, которая *заполнит* объект `ArrayList` объектами с типом `String` и выведет содержимое получившейся коллекции.

```
import java.util.ArrayList;

public class ArrayListDemo {

  public static void main(String[] args) {

      // Создаем и заполняем ArrayList
      ArrayList friends = new ArrayList();
      friends.add("Елена");
      friends.add("Анна");
      friends.add("Николай");
      friends.add("Сергей");

      // Сколько в нем друзей?
      int friendsCount = friends.size();

    // Печатаем содержимое ArrayList
    for (int i=0; i<friendsCount; i++){

      System.out.println("Друг №" + i + " это "+friends.get(i));

    }
  }
}
```

Эта программа напечатает следующие строки:

```
Друг №0 это Елена
Друг №1 это Анна
Друг №2 это Николай
Друг №3 это Сергей
```

Метод `get()` *извлекает* из объекта `ArrayList` элемент, который располагается по указанному индексу. Так как вы можете хранить в коллекции любые объекты, метод `get()` возвращает каждый элемент с типом `Object`. Ответственность за приведение этого объекта к правильному типу данных ложится на программу. Мы не обязаны были делать это в предыдущем примере только потому, что хранили в коллекции `friends` объекты с типом `String`.

Java автоматически выполняет преобразование объекта с типом `Object` в объект с типом `String`. Но, если вы решите хранить в `ArrayList` другие объекты, например, экземпляры класса `Fish`, правильный код для добавления и извлечения конкретных объектов `Fish` может выглядеть так, как в программе `FishTank`, которая приводится далее.

Сначала эта программа создает несколько экземпляров класса `Fish`, назначает некое значение цвету, весу и текущей глубине погружения и сохраняет эти значения в объект `ArrayList` с именем `fishTank`. Затем программа получает объекты из коллекции, приводит их тип к типу `Fish` и печатает значения этих объектов.

```
import java.util.ArrayList;

public class FishTank {

  public static void main(String[] args) {

      ArrayList fishTank = new ArrayList();

      Fish theFish;

      Fish aFish = new Fish(20);

      aFish.color = "красную";

      aFish.weight = 2;

      fishTank.add(aFish);

      aFish = new Fish(10);

      aFish.color = "зеленую";

      aFish.weight = 5;

      fishTank.add(aFish);
```

```
      int fishCount = fishTank.size();


      for (int i=0;i<fishCount; i++){

        theFish = (Fish) fishTank.get(i);

        System.out.println("Поймал "+ theFish.color +
          " рыбу с весом " + theFish.weight + " кг. Глубина:"
           + theFish.currentDepth);
      }
    }
}
```

Ниже приводятся выходные данные программы FishTank:

```
Поймал красную рыбу с весом 2.0 кг. Глубина:20
Поймал зеленую рыбу с весом 5.0 кг. Глубина:10
```

Теперь, когда вы прочитали об уровнях доступа в языке Java, можно немного изменить классы Pet и Fish. Такие переменные как age, color, weight и height можно объявлять с модификатором доступа protected, если вы думаете, что кто-то захочет наследовать свои классы из них. А переменная currentDepth должна быть private. Также нужно добавить новые public методы, например, getAge(), чтобы возвращать значение переменной age, и метод setAge(), чтобы устанавливать значение этой переменной.

Программисты с хорошими манерами не разрешают одному классу напрямую изменять свойства другого класса. Класс должен предоставлять методы, которые бы изменяли его внутренние элементы. Вот почему класс Score из предыдущего раздела был спроектирован с private переменными, которые могли быть получены или изменены только с помощью специальных getter- и setter-методов для чтения и записи соответственно.

В этой главе я показал вам различные элементы и приемы в языке Java, которые, на первый взгляд, кажутся несвязанными друг с другом. Однако все эти элементы очень часто используются профессиональными программистами на Java. После выполнения практических упражнений к этой главе, вы будете лучше понимать как эти элементы работают вместе.

Материалы для дополнительного чтения

1. Коллекции Java:

http://download.oracle.com/javase/tutorial/collections/intro/index.html

2. Класс `ArrayList`:

http://download.oracle.com/javase/7/docs/api/java/util/ArrayList.html

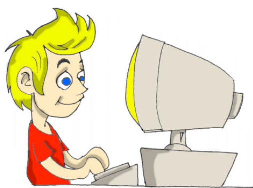
3. Работа с датами:

http://www.vogella.de/articles/JavaDateTimeAPI/article.html

4. Класс `Calendar`:

http://download.oracle.com/javase/7/docs/api/java/util/Calendar.html

Практические упражнения



1. Добавьте перегруженный конструктор по умолчанию (без аргументов) в класс `Fish`. Этот конструктор должен устанавливать значение стартовой позиции равное 10 метрам. Класс `FishMaster` будет создавать экземпляр объекта `Fish` очень просто, вот так:

```
Fish myFish = new Fish();
```

2. Добавьте в класс `Score` конструктор с четырьмя аргументами. Создайте программу `ScoreWriter3`, которая будет заполнять экземпляры класса `Score` не с помощью setter-методов, а в момент создания объекта, например так:

```
Score aScore = new Score("Николай",
```

```
                    "Смирнов", 250, today);
```

Практические упражнения для умников и умниц

Погуглите как использовать класс `HashMap`. Попробуйте создать программу `HashMapDemo`, которая похожа на программу `ArrayListDemo`, но в коллекции хранятся имена и телефоны.

Глава 11. Возвращаемся к графике. Пинг-Понг

В главах 5, 6 и 7 мы использовали некоторые компоненты из AWT и Swing библиотек. Теперь я покажу вам, как можно рисовать и двигать такие фигуры, как овалы, прямоугольники и линии внутри окна. Также, вы узнаете, как обрабатывать события мыши и клавиатуры. Чтобы добавить немного веселья в эти скучные темы, в этой главе мы будем изучать их при создании игры пинг-понг. В игре будут два участника, я называю их *ребенок* и *компьютер*.

Стратегия

Давайте установим правила игры:

- ✓ Игра продолжается, пока один из игроков (ребенок или компьютер) не наберет 21 очко.
- ✓ Ракетка ребенка будет управляться с помощью мыши.
- ✓ Счет игры будет отображаться в нижней части окна.
- ✓ Новая игра начинается при нажатии кнопки *N* на клавиатуре, *Q* – завершает игру и *S* – подает мяч.
- ✓ Подать мяч может только ребенок.
- ✓ Для того, что выиграть очко, мяч должен попасть в область за вертикальной линией ракетки, когда ракетка не блокирует мяч.

- ✓ Когда компьютер отбивает мяч, мяч может двигаться только горизонтально вправо.
- ✓ Если мяч касается ракетки в верхней половине стола, он может двигаться только влево и вверх. Если мяч находился в нижней части стола, он может двигаться только вниз и влево.

Должно быть, вы думаете, что написать такую программу очень сложно. Хитрость в том, чтобы разбить сложную задачу на несколько маленьких и простых задач и сделать каждую из них отдельно.

Эта хитрость называется *аналитическим мышлением* и она помогает не только в программировании, но и в повседневной жизни. Не расстраивайтесь, если вы не можете достичь большой цели, разбейте ее на маленькие и выполняйте по отдельности.

Поэтому первая версия программы будет выполнять только некоторые из правил – программа будет рисовать стол, перемещать ракетку и показывать координаты кликов мыши.

Вместо того, чтобы говорить «Мой компьютер не работает» (большая проблема), попробуйте посмотреть, что именно не работает (найдите проблему поменьше).

1. Подключен ли компьютер к розетке? (да/нет)? *Да.*

2.Когда я запускаю компьютер, отображаются ли иконки на экране (да/нет)? *Да*.

3. Можно ли перемещать мышь по экрану (да/нет)? *Нет*.

4. Правильно ли подключен кабель мыши (да/нет)? *Нет*.

Просто подключите кабель мыши и компьютер опять будет работать! Большая проблема решается всего лишь подключением кабеля мыши.

Код

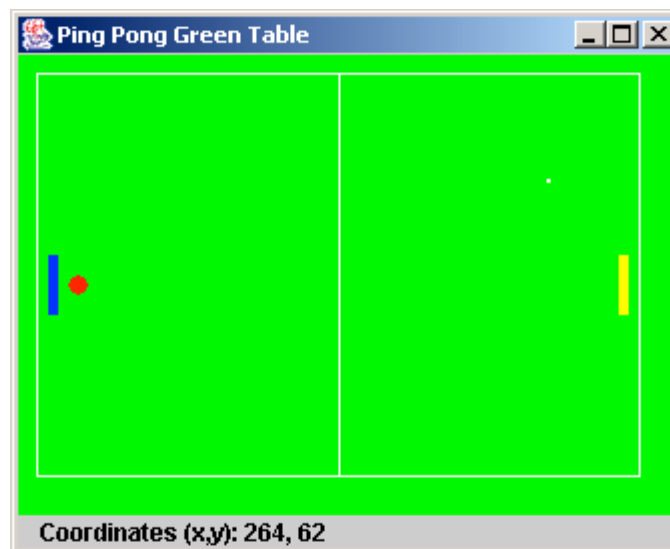
Игра будет состоять из трех классов:

- ✓ Класс `PingPongGreenTable` возьмет на себя визуальную часть. В течение игры он будет отображать стол, ракетки и мяч.

- ✓ Класс `PingPongGameEngine`, который будет считать координаты мяча и ракеток, начинать и завершать игру, подавать мяч. Класс будет передавать текущие координаты компонентов классу `PingPongGreenTable`, который будет перерисовываться в соответствии с данными.

- ✓ Интерфейс `GameConstants` будет объявлять все константы, которые понадобятся в игре, например длину и ширину стола, начальные позиции ракеток и т.д.

Вот как будет выглядеть стол для пинг-понга:



Первая версия этой игры будет делать только три вещи:

- ➢ Отображать зеленый стол для пинг-понга.
- ➢ Отображать координаты указателя мыши, когда вы кликаете.
- ➢ Двигать ракетку ребенка вверх и вниз.

Через две страницы вы сможете увидеть наш класс `PingPongGreenTable`, который наследуется от класса `JPanel` из Swing. Посмотрите на этот код, пока читайте текст ниже.

Так как наша программа должна знать точные координаты указателя мыши, конструктор PingPongGreenTable будет создавать объект класса-обработчика событий PingPongGameEngine.

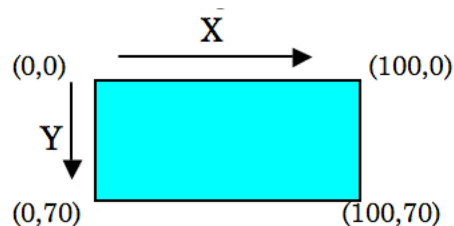
Этот класс будет выполнять некоторые действия, когда ребенок кликает на кнопку мыши или просто двигает ее. Метод addPaneltoFrame() создает текстовый элемент, который будет отображать координаты мыши.

Этот класс не апплет, поэтому взамен метода paint(), используется paintComponent(). Этот метод может быть вызван как JVM, когда нужно перерисовать окно, так и нашей программой, с помощью метода repaint(). Да, вы прочли правильно, метод repaint() внутри вызывает paintComponent() и предоставляет вашему классу доступ к объекту Graphics, чтобы вы смогли рисовать внутри окна. Мы будем вызывать этот метод каждый раз после пересчета координат ракеток или мяча для их правильного отображения.

Чтобы нарисовать ракетку, сначала задайте цвет, а потом заполните им прямоугольник с помощью метода fillRect(). Этот метод должен знать X и Y координаты верхнего левого угла прямоугольника, а так же ширину и высоту в пикселях.

Круг рисуется с помощью метода fillOval(), для этого надо знать координаты центра овала, его высоту и ширину. Когда высота и ширина одинаковы, овал выглядит как круг.

Координата X в окне увеличивается слева направо, координата Y растет сверху вниз. Допустим, ширина вот этого прямоугольника 100 пикселей, а высота – 70:



Координаты X и Y углов прямоугольника показаны в скобках.

Еще один интересный метод – getPreferredSize(). Для того, чтобы установить размеры стола, мы создадим экземпляр класса Dimension из Swing. Виртуальной машине Java нужно знать о размерах окна, поэтому она вызывает метод getPreferredSize() объекта

PingPongGreenTable. Этот метод возвращает объект Dimension, который мы создали в соответствии с размерами нашего стола.

Оба класса PingPongGreenTable и PingPongGameEngine используют некоторые константы, которые не меняются. Например, в классе PingPongGreenTable используется ширина и высота стола, а PingPongGameEngine должен знать, на сколько пикселей двигать мяч – чем меньше это значение, тем более плавным будет движение.

Удобно хранить все константы (переменные final) в интерфейсе. В нашей игре интерфейс называется GameConstants. Если в классе понадобятся эти значения, просто добавьте implements GameConstants к объявлению класса и используйте любые переменные final из этого интерфейса так, как будто они заданы в этом же классе! Поэтому оба класса PingPongGreenTable и PingPongGameEngine реализуют интерфейс GameConstants.

Если вы решите поменять размер стола, мяча или ракетки, единственное место, где это нужно будет сделать – интерфейс GameConstants. Давайте посмотрим на код класса PingPongGreenTable и интерфейс GameConstants.

```
package screens;


import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.BoxLayout;
import javax.swing.JLabel;
import javax.swing.WindowConstants;
import java.awt.Point;
import java.awt.Dimension;
import java.awt.Container;
import java.awt.Graphics;
import java.awt.Color;
import engine.PingPongGameEngine;

/**

Этот класс рисует стол для пинг-понга и отображает координаты
точки,где пользователь кликнул мышью
*/

publicclass PingPongGreenTable extends JPanel

implements GameConstants{

JLabel label;

public Point point = new Point(0,0);
```

```
public int ComputerRacket_X =15;
private int kidRacket_Y =KID_RACKET_Y_START;


Dimension preferredSize= new Dimension(TABLE_WIDTH,TABLE_HEIGHT);

// Этот метод устанавливает размер
// Вызывается виртуальной Java машиной

public Dimension getPreferredSize() {

      return preferredSize;

}

//Конструктор. Создает обработчик событий мыши.
PingPongGreenTable(){

  PingPongGameEngine gameEngine =new PingPongGameEngine(this);

  // Обрабатывает клики мыши для отображения ее координат
  addMouseListener(gameEngine);

  // Обрабатывает движения мыши для передвижения ракеток
  addMouseMotionListener(gameEngine);

}


// Добавить панель с JLabel в окно
void addPaneltoFrame(Container container) {

      container.setLayout(new BoxLayout(container,

                                          BoxLayout.Y_AXIS));

      container.add(this);

      label = new JLabel("Click to see coordinates");
      container.add(label);
}

// Перерисовать окно. Этот метод вызывается виртуальной
// машиной, когда нужно обновить экран или
// вызывается метод repaint() из PingPointGameEngine
publicvoid paintComponent(Graphics g) {

      super.paintComponent(g);
      g.setColor(Color.GREEN);

    // Нарисовать стол
    g.fillRect(0,0,TABLE_WIDTH,TABLE_HEIGHT);
    g.setColor(Color.yellow);

    // Нарисовать правую ракетку
    g.fillRect(KID_RACKET_X_START,kidRacket_Y,5,30);
    g.setColor(Color.blue);
```

```
        // Нарисовать левую ракетку
        g.fillRect(ComputerRacket_X,100,5,30);
        g.setColor(Color.red);

        g.fillOval(25,110,10,10); //Нарисовать мяч
        g.setColor(Color.white);

        g.drawRect(10,10,300,200);
        g.drawLine(160,10,160,210);

        // Отобразить точку как маленький квадрат 2x2 пикселей
        if (point != null) {

           label.setText("Coordinates (x,y): " + point.x +
                                                 ", " + point.y);

           g.fillRect(point.x, point.y, 2, 2);

        }
}

// Установить текущее положение ракетки ребенка
public void setKidRacket_Y(int xCoordinate){

        this.kidRacket_Y = xCoordinate;

    }

// Вернуть текущее положение ракетки ребенка
public int getKidRacket_Y(int xCoordinate){

        return kidRacket_Y;

}

publicstaticvoid main(String[] args) {

        // Создать экземпляр окна
       JFrame f = new JFrame("Ping Pong Green Table");

        // Убедиться, что окно может быть закрыто по нажатию на
        //крестик в углу
        f.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        PingPongGreenTable table = new PingPongGreenTable();

        table.addPaneltoFrame(f.getContentPane());

        // Установить размер окна и сделать его видимым
        f.pack();
        f.setVisible(true);

    }
}
```

Теперь посмотрим на интерфейс GameConstants. Все значения переменных в пикселях. В названиях final переменных используйте заглавные буквы:

```
package screens;

public interface GameConstants {

      public final int TABLE_WIDTH =  320;
      public final int TABLE_HEIGHT = 220;
      public final int KID_RACKET_Y_START = 100;
      public final int KID_RACKET_X_START = 300;
      public final int TABLE_TOP = 12;
      public final int TABLE_BOTTOM = 180;
      public final int RACKET_INCREMENT = 4;
}
```

Запущенная программа не сможет поменять значения этих переменных, потому что они объявлены как final. Но, если вы решите поменять, например, размер стола, достаточно изменить значения TABLE_WIDTH и TABLE_HEIGHT и перекомпилировать интерфейс GameConstants.

Решения в этой игре принимает класс PingPongGameEngine, который реализует 2 интерфейса, связанных с событиями мыши.

MouseListener будет использоваться в методе mousePressed(). На каждое нажатие мыши этот метод будет рисовать маленькую белую точку на столе и отображать ее координаты. Честно говоря, в нашей игре этот код бесполезен, но он покажет простой способ достать из объекта MouseEvent координаты мыши, которые были переданы JVM.

Метод mousePressed() передает координаты нажатой кнопки мыши переменной point. После того, как координаты переданы, этот метод просит виртуальную машину перерисовать стол.

MouseMotionListener отслеживает движение мыши над столом, а метод mouseMoved() будет использоваться для перемещения ракетки ребенка вниз или вверх.

Метод mouseMoved() считает следующую позицию ракетки игрока. Если указатель мыши находится над ракеткой (координата Y мыши меньше, чем координата Y ракетки), то этот метод гарантирует, что ракетка не выйдет за пределы стола.

Когда конструктор PingPongGreenTable создает объект класса PingPongGameEngine, он передает в него ссылку объекта стола (ключевое слово this означает ссылку на область памяти с объектом PingPongGreenTable). Теперь, PingPongGameEngine может «разговаривать» со столом, например, устанавливать новые

координаты мяча или перерисовать стол, когда нужно. Если эта часть не совсем понятна, перечитайте раздел главы 6 о передаче данных между классами.

В нашей игре ракетки перемещаются вертикально по 4 пикселя, как мы задали в интерфейсе GameConstants (класс PingPongGameEngine реализует этот интерфейс). Например, следующая строка отнимает 4 от значения переменной kidRacket_Y:

```
kidRacket_Y -= RACKET_INCREMENT;
```

Например, если координата Y ракетки была 100, после этой строки кода она становится равной 96 и ракетка должна подняться вверх. Тот же результат можно получить таким выражением:

```
kidRacket_Y = kidRacket_Y - RACKET_INCREMENT;
```

Если вы помните, мы говорили о разных способах изменить значение переменной в Главе 3.

Далее – класс PingPongGameEngine.

```
package engine;

import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import screens.*;

public class PingPongGameEngine  implements
       MouseListener, MouseMotionListener, GameConstants{

 PingPongGreenTable table;

 public int kidRacket_Y = KID_RACKET_Y_START;

// Конструктор. Содержит ссылку на объект стола
public PingPongGameEngine(PingPongGreenTable greenTable){

      table = greenTable;

 }

// Обязательные методы из интерфейса MouseListener

 public void mousePressed(MouseEvent e) {

// Взять X и Y координаты указателя мыши
// и установить их "белой точке" на столе

      table.point.x = e.getX();
      table.point.y = e.getY();
```

```
// Внутри вызывает метод paintComponent()и обновляет окно

      table.repaint();

 }

 public void mouseReleased(MouseEvent e) {};
 public void mouseEntered(MouseEvent e) {};
 public void mouseExited(MouseEvent e) {};
 public void mouseClicked(MouseEvent e) {};

// Обязательные методы из интерфейса MouseMotionListener

 public void mouseDragged(MouseEvent e) {}

 public void mouseMoved(MouseEvent e) {

  int mouse_Y = e.getY();

// Если мышь находится выше ракетки ребенка
// и не выходит за пределы стола -
// передвинуть ее вверх, в противном случае - опустить вниз

  if (mouse_Y < kidRacket_Y && kidRacket_Y > TABLE_TOP){

      kidRacket_Y -= RACKET_INCREMENT;

  }else if (kidRacket_Y < TABLE_BOTTOM) {

                  kidRacket_Y += RACKET_INCREMENT;

  }

   // Установить новое положение ракетки
   table.setKidRacket_Y(kidRacket_Y);
   table.repaint();

  }
}
```

Основы многопоточности

До этого все действия в наших программах выполнялись последовательно – одно за другим. Если программа вызывает два метода, второй метод ждет, пока не выполнится первый. Другими словами, каждая из наших программ имеет только один *поток исполнения (a thread)*.

Однако, в реальной жизни мы можем делать несколько вещей одновременно, например, есть, разговаривать по телефону, смотреть телевизор и делать домашнее задание. Чтобы выполнять все эти

действия *параллельно*, мы используем несколько *процессоров*: руки, глаза и рот.

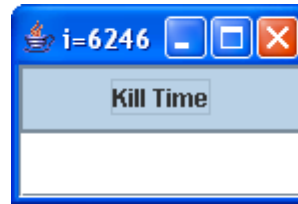
На сегодня, во многих компьютерах тоже два или более процессоров (CPU). Хотя, возможно, в вашем компьютере только один процессор, который считает, посылает команды монитору, диску, удаленным компьютерам и т.д.

Но даже один процессор может выполнять несколько действий сразу, если программа использует *несколько потоков (multiple threads)*. Один Java-класс может запустить несколько *потоков исполнения*, которые будут меняться и получать свою долю процессорного времени.

Хороший пример программы, которая создает несколько потоков – это web-браузер. Вы можете смотреть Интернет-сайты, пока качаются несколько файлов – одна программа запускает два потока исполнения.

В следующей версии нашей пинг-понг игры будет один поток для отрисовки стола. Второй поток будет считать координаты мяча и ракеток, и посылать команды по отрисовке окна первому потоку. Но сначала, я покажу вам две очень простые программы, чтобы вы лучше поняли, зачем нужны потоки.

Каждая из этих программ будет отображать кнопку и текстовое поле.



Когда вы нажмете кнопку *Kill Time*, программа начнет цикл, который увеличивает значение переменной триста тысяч раз. Текущее значение этой переменной будет отображаться в заголовке окна. В классе `NoThreadsSample` только один поток исполнения и вы *не сможете ничего напечатать в текстовом поле, пока цикл не закончится*. Этот цикл забирает все процессорное время, поэтому окно заблокировано.

Скомпилируйте и запустите этот класс, чтобы убедиться, что окно блокируется в течение некоторого времени. Заметьте, что этот класс создает экземпляр класса `JTextField` и передает его в панель контента без создания переменной. Если вы не планируете получать или устанавливать атрибуты этого объекта, вам не нужна такая переменная.

```
import javax.swing.*;
import java.awt.GridLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class NoThreadsSample extends JFrame
                             implements ActionListener{

  // Конструктор
  NoThreadsSample(){

  // Создать окно с кнопкой и текстовым полем

      GridLayout gl =new GridLayout(2,1);
      this.getContentPane().setLayout(gl);

      JButton myButton = new JButton("Kill Time");
      myButton.addActionListener(this);

      this.getContentPane().add(myButton);
      this.getContentPane().add(new JTextField());
 }

// Обработчик нажатия кнопки
public void actionPerformed(ActionEvent e){

// Просто заморозить на некоторое время,
// чтобы показать, что окно заблокировано
   for (int i=0; i<300000;i++){

      this.setTitle("i="+i);
```

```
      }

    }

public static void main(String[] args) {

  // Создать окно
  NoThreadsSample myWindow = new NoThreadsSample();

 // Убедиться, что окно закрывается при нажатии на крестик в углу
 myWindow.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);


  // Установить размеры окна – координаты левого верхнего
  //угла и высоту с шириной
  myWindow.setBounds(0,0,150, 100);

  //Сделать окно видимым
  myWindow.setVisible(true);

  }
}
```

Следующая версия этого маленького окошка будет создавать и запускать отдельный поток для цикла, и главный поток окна позволит печатать в текстовом поле, пока цикл выполняется.

Вы можете создать поток одним из следующих способов:

- ✓ Создать экземпляр Java-класса `Thread` и передать ему объект, который реализует интерфейс `Runnable`. Если ваш класс реализует интерфейс `Runnable`, код будет выглядеть так:

  ```
  Thread worker = new Thread(this);
  ```

  Этот интерфейс требует написать в методе `run()` код, который будет выполняться в отдельном потоке.

  ```
  worker.start();
  ```

- ✓ Создать подкласс класса `Thread` и реализовать метод `run()`. Для того, чтобы запустить поток, надо вызвать метод `start()`.

```
public class MyThread extends Thread{

   public static void main(String[] args) {

         MyThread worker = new MyThread();
         worker.start();

   }

   public void run(){
```

```
    // Здесь будет ваш код

  }
}
```

В классе ThreadsSample я буду использовать первый способ, потому что этот класс уже наследуется от JFrame, а наследовать больше одного класса в Java нельзя.

```
import javax.swing.*;
import java.awt.GridLayout;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class ThreadsSample extends JFrame
                    implements ActionListener, Runnable{

  // Конструктор
  ThreadsSample(){

    // Создать окно с кнопкой и текстовым полем
    GridLayout gl =new GridLayout(2,1);

    this.getContentPane().setLayout(gl);

    JButton myButton = new JButton("Kill Time");
    myButton.addActionListener(this);
    this.getContentPane().add(myButton);

    this.getContentPane().add(new JTextField());
}

public void actionPerformed(ActionEvent e){

  // Создать поток и выполнить “замораживающий” код
  // без блокировки

    Thread worker = new Thread(this);
    worker.start();  // вызывает метод run()

  }

public void run(){

  // Заморозить на некоторое время, чтобы показать, что
  // элементы окна НЕ блокируются

      for (int i=0; i<300000;i++){

         this.setTitle("i="+i);

      }
}
```

```
  public static void main(String[] args) {

      ThreadsSample myWindow = new ThreadsSample();

   //Убедись,что окно закрывается по нажатию на крестик в углу
    myWindow.setDefaultCloseOperation(
                         WindowConstants.EXIT_ON_CLOSE);

  // Установи размеры окна и сделай его видимым

      myWindow.setBounds(0,0,150, 100);
      myWindow.setVisible(true);

  }
}
```

После нажатия на кнопку *Kill Time*, класс `ThreadsSample` запускает новый поток. После этого, поток с циклом и главный поток получают по своей доле процессорного времени. И теперь вы можете печатать в текстовом поле (главный поток), пока другой поток выполняет цикл!

Изучение потоков заслуживают гораздо большего внимания, чем эти несколько страниц, и я рекомендую вам почитать дополнительный материалы по этой теме.

Заканчиваем игру Пинг-Понг

Теперь, после краткого введения в потоки мы готовы поменять классы нашей игры в пинг-понг. Давайте начнем с класса `PingPongGreenTable`. Нам не надо отображать белую точку по клику мыши – это было просто учебное упражнение для отображения координат указателя мыши. Поэтому мы удалим объявление переменной `point` и строки, которые рисуют белую точку из метода `paintComponent()`. Также, в конструкторе больше не нужен `MouseListener`, так как он только показывает координаты точки.

С другой стороны, этот класс должен реагировать на некоторые кнопки клавиатуры (*N* – для начала новой игры, *S* – для подачи мяча и *Q* – для выхода из игры). В этом нам поможет метод `addKeyListener()`.

Для того, чтобы сделать наш код более инкапсулированным, я переместил вызовы `repaint()` из класса `PingPongGameEngine` в класс `PingPongGreenTable`. Теперь, когда понадобится, `PingPongGreenTable` будет перерисовывать себя сам.

Также, я добавил методы для изменения положения мяча, ракетки компьютера и для отображения сообщений.

```
package screens;

import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.BoxLayout;
import javax.swing.JLabel;
import javax.swing.WindowConstants;
import java.awt.Dimension;
import java.awt.Container;
import java.awt.Graphics;
import java.awt.Color;
import engine.PingPongGameEngine;

/**

*Этот класс рисует зеленый стол для пинг-понга,
шар, ракетки, отображает счет

*/

public class PingPongGreenTable extends JPanel
                                 implements GameConstants{

  private JLabel label;
  private int computerRacket_Y = COMPUTER_RACKET_Y_START;
  private int kidRacket_Y = KID_RACKET_Y_START;
  private int ballX = BALL_START_X;
  private int ballY = BALL_START_Y;

  Dimension preferredSize = new
                    Dimension(TABLE_WIDTH,TABLE_HEIGHT);

// Устанавливаем размеры окна.Вызывается виртуальной машиной
public Dimension getPreferredSize() {

    return preferredSize;

  }

 // Конструктор. Создает обработчик событий мыши.
  PingPongGreenTable(){

    PingPongGameEngine gameEngine = new PingPongGameEngine(this);

   // Обрабатываем движения мыши для передвижения ракеток
   addMouseMotionListener(gameEngine);

   // Обрабатываем события клавиатуры
   addKeyListener(gameEngine);

  }

// Добавитм в окно панель с JLabel
void addPaneltoFrame(Container container) {
```

```
   container.setLayout(new BoxLayout(container,BoxLayout.Y_AXIS));
   container.add(this);

   label = new JLabel(
            "Press N for a new game, S to serve or Q to quit");
   container.add(label);

  }

// Перерисовать окно. Этот метод вызывается виртуальной
// машиной, когда нужно обновить экран или
// вызывается метод repaint() из PingPointGameEngine
public void paintComponent(Graphics g) {

    super.paintComponent(g);

   // Нарисовать зеленый стол
   g.setColor(Color.GREEN);
   g.fillRect(0,0,TABLE_WIDTH,TABLE_HEIGHT);

   // Нарисовать правую ракетку
    g.setColor(Color.yellow);
    g.fillRect(KID_RACKET_X, kidRacket_Y,
                                RACKET_WIDTH, RACKET_LENGTH);

   // Нарисовать левую ракетку
    g.setColor(Color.blue);
    g.fillRect(COMPUTER_RACKET_X, computerRacket_Y,
                                 RACKET_WIDTH,RACKET_LENGTH);

   // Нарисовать мяч
    g.setColor(Color.red);
    g.fillOval(ballX,ballY,10,10);

    // Нарисовать белые линии

    g.setColor(Color.white);
    g.drawRect(10,10,300,200);
    g.drawLine(160,10,160,210);

    // Установить фокус на стол, чтобы
    // обработчик клавиатуры мог посылать команды столу
    requestFocus();

  }

// Установить текущее положение ракетки ребенка

public void setKidRacket_Y(int yCoordinate){

   this.kidRacket_Y = yCoordinate;
   repaint();

}


// Вернуть текущее положение ракетки ребенка
public int getKidRacket_Y(){
```

```
    return kidRacket_Y;

 }

// Установить текущее положение ракетки компьютера
public void setComputerRacket_Y(int yCoordinate){

    this.computerRacket_Y = yCoordinate;
    repaint();
}

// Установить игровое сообщение
public void setMessageText(String text){

    label.setText(text);
    repaint();
}

// Установить позицию мяча
 public void setBallPosition(int xPos, int yPos){

    ballX=xPos;
    ballY=yPos;
    repaint();
}

public static void main(String[] args) {

  // Создать экземпляр окна
  JFrame f = new JFrame("Ping Pong Green Table");

  // Убедиться, что окно может быть закрыто по нажатию на
  //крестик в углу
 f.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

  PingPongGreenTable table = new PingPongGreenTable();
  table.addPaneltoFrame(f.getContentPane());

  // Установить размер окна и сделать его видимым
  f.setBounds(0,0,TABLE_WIDTH+5, TABLE_HEIGHT+40);
  f.setVisible(true);

 }

}
```

Я добавил несколько `final` переменных в интерфейс `GameConstants`, и вы должны догадаться по их именам, для чего они нужны.

```
package screens;

/**
*Этот интерфейс содержит все константы,которые используются в игре
*/
public interface GameConstants {

  // Размеры стола
  public final int TABLE_WIDTH =  320;
```

```
  public final int TABLE_HEIGHT = 220;
  public final int TABLE_TOP = 12;
  public final int TABLE_BOTTOM = 180;

  // Шаг перемещения мяча в пикселях
  public final int BALL_INCREMENT = 4;

  // Максимальные и минимальные координаты мяча
  public final int BALL_MIN_X = 1+ BALL_INCREMENT;
  public final int BALL_MIN_Y = 1 + BALL_INCREMENT;
  public final int BALL_MAX_X = TABLE_WIDTH - BALL_INCREMENT;
  public final int BALL_MAX_Y = TABLE_HEIGHT - BALL_INCREMENT;

  // Начальные координаты мяча
  public final int BALL_START_X = TABLE_WIDTH/2;
  public final int BALL_START_Y = TABLE_HEIGHT/2;

  //Размеры, расположения и шаг перемещения ракеток

  public final int KID_RACKET_X = 300;
  public final int KID_RACKET_Y_START = 100;
  public final int COMPUTER_RACKET_X = 15;
  public final int COMPUTER_RACKET_Y_START = 100;
  public final int RACKET_INCREMENT = 2;
  public final int RACKET_LENGTH = 30;
  public final int RACKET_WIDTH = 5;
  public final int WINNING_SCORE = 21;

 // Замедлить быстрые компьютеры – измените это значение,
 // если понадобится
  public final int SLEEP_TIME = 10; //время в миллисекундах
}
```

Ниже я перечислил основные изменения, которые сделал в классе PingPongGameEngine:

- ✓ Удалил интерфейс MouseListener и все его методы, потому что мы больше не обрабатываем клики мыши. Все движения мыши будут обрабатываться MouseMotionListener.

- ✓ Теперь этот класс реализует интерфейс Runnable, а вся логика находится в методе run(). Посмотрите на конструктор – там я создаю и запускаю новый поток. Метод run() обрабатывает правила игры в несколько шагов, все эти шаги запрограммированы внутри условия if(ballServed). Это сокращенный вариант выражения if(ballServed==true).

- ✓ Пожалуйста, обратите внимание на условие, которое устанавливает значение переменной canBounce в первом шаге. В зависимости от этого выражения, значение переменной будет либо true, либо false.

- ✓ Класс реализует интерфейс `KeyListener`, и метод `keyPressed()` проверяет, какая кнопка была нажата для начала/завершения игры, либо подачи мяча. Этот метод позволяет обрабатывать как заглавные, так и маленькие буквы, например `N` и `n`.

- ✓ Я добавил несколько `private` методов: `displayScore()`, `kidServe()` и `isBallOnTheTable()`. Они объявлены приватными, потому что используются только внутри этого класса и другие классы даже не подозревают об их существовании. Это пример *инкапсуляции* в действии.|

- ✓ Некоторые компьютеры настолько быстрые, что контролировать движение мяча становится трудно. Поэтому я замедлил игру с помощью метода `Thread.sleep()`. Статический метод `sleep()` приостановит текущий поток на заданное в конструкторе количество миллисекунд.

- ✓ Чтобы игра стала немного веселее, мяч теперь движется по диагонали после удара ракеткой ребенка. Поэтому меняется не только `X` координата мяча, но и `Y`.

```
package engine;

import java.awt.event.MouseMotionListener;
import java.awt.event.MouseEvent;
import java.awt.event.KeyListener;
import java.awt.event.KeyEvent;
import screens.*;

/**

*Этот класс - обработчик событий мыши и клавиатуры.
* Рассчитывает движение мяча и ракеток,изменение их координат.
*/

public class PingPongGameEngine implementsRunnable,
                 MouseMotionListener, KeyListener, GameConstants{

   private PingPongGreenTable table; // ссылка на стол
   private int kidRacket_Y = KID_RACKET_Y_START;
   private int computerRacket_Y=COMPUTER_RACKET_Y_START;
   private int kidScore;
   private int computerScore
   private int ballX;   // координата X мяча
   private int ballY;   // координата Y мяча
   private boolean movingLeft = true;
   private boolean ballServed = false;
```

```
  //Значение вертикального передвижения мяча в пикселях
  private int verticalSlide;

  // Конструктор. Содержит ссылку на объект стола
  public PingPongGameEngine(PingPongGreenTable greenTable){

   table = greenTable;

   Thread worker = new Thread(this);
   worker.start();

  }

// Обязательные методы из интерфейса MouseMotionListener
// (некоторые из них пустые,но должны быть включены все равно)
public void mouseDragged(MouseEvent e) {

 }

public void mouseMoved(MouseEvent e) {

  int mouse_Y = e.getY();

// Если мышь находится выше ракетки ребенка
// и не выходит за пределы стола – передвинуть ее вверх,
// в противном случае – опустить вниз

   if (mouse_Y<kidRacket_Y && kidRacket_Y>TABLE_TOP){

      kidRacket_Y -= RACKET_INCREMENT;

   }elseif (kidRacket_Y < TABLE_BOTTOM) {

      kidRacket_Y += RACKET_INCREMENT;

   }

   // Установить новое положение ракетки
   table.setKidRacket_Y(kidRacket_Y);

 }

// Обязательные методы из интерфейса KeyListener
public void keyPressed(KeyEvent e){

char key = e.getKeyChar();

if ('n' == key || 'N' == key){

      startNewGame();

   } elseif ('q' == key || 'Q' == key){

      endGame();

   } elseif ('s' == key || 'S' == key){

      kidServe();

   }
```

```
}

public void keyReleased(KeyEvent e){}

public void keyTyped(KeyEvent e){}

// Начать новую игру
public void startNewGame(){

  computerScore=0;
  kidScore=0;
  table.setMessageText("Score Computer: 0  Kid: 0");

  kidServe();
}

// Завершить игру
public void endGame(){

  System.exit(0);

}


// Обязательный метод run() из интерфейса Runnable
public void run(){

  boolean canBounce=false;

  while (true) {

    if(ballServed){ // если мяч движется

   //Шаг 1. Мяч движется влево?
   if ( movingLeft && ballX > BALL_MIN_X){

     canBounce = (ballY >= computerRacket_Y &&
      ballY < (computerRacket_Y + RACKET_LENGTH)?true: false);

     ballX-=BALL_INCREMENT;

    // Добавить смещение вверх или вниз к любым
    // движениям мяча влево или вправо
    ballY-=verticalSlide;

    table.setBallPosition(ballX,ballY);

    // Может отскочить?
    if (ballX <= COMPUTER_RACKET_X  && canBounce){

      movingLeft=false;

    }
  }

  // Шаг 2. Мяч движется вправо?
  if ( !movingLeft && ballX <= BALL_MAX_X){

     canBounce = (ballY >= kidRacket_Y && ballY <
```

```
                (kidRacket_Y + RACKET_LENGTH)?true:false);


        ballX+=BALL_INCREMENT;
        table.setBallPosition(ballX,ballY);

        // Может отскочить?
        if (ballX >= KID_RACKET_X && canBounce){
                       movingLeft=true;

        }
     }

     // Шаг 3. Перемещать ракетку компьютера вверх или вниз,
     // чтобы блокировать мяч
     if (computerRacket_Y < ballY
                && computerRacket_Y < TABLE_BOTTOM){

        computerRacket_Y +=RACKET_INCREMENT;

     }else if (computerRacket_Y > TABLE_TOP){

        computerRacket_Y -=RACKET_INCREMENT;
     }

    table.setComputerRacket_Y(computerRacket_Y);

    // Шаг 4. Приостановить
    try {

        Thread.sleep(SLEEP_TIME);

    } catch (InterruptedException e) {

        e.printStackTrace();

    }

     // Шаг 5. Обновить счет, если мячв зеленой области, но не движется
     if (isBallOnTheTable()){

        if (ballX > BALL_MAX_X ){

          computerScore++;
          displayScore();

        }else if (ballX < BALL_MIN_X){

          kidScore++;
          displayScore();

        }
     }

    } // Конец if ballServed
   } // Конец while
 }// Конец run()
```

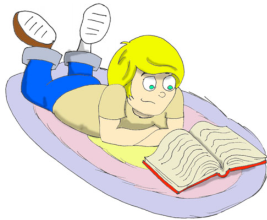
```
// Подать с текущей позиции ракетки ребенка

private void kidServe(){

   ballServed = true;
   ballX = KID_RACKET_X-1;
   ballY=kidRacket_Y;

   if (ballY > TABLE_HEIGHT/2){

      verticalSlide=-1;
   }else{

      verticalSlide=1;
   }

   table.setBallPosition(ballX,ballY);
   table.setKidRacket_Y(kidRacket_Y);
}


private void displayScore(){

   ballServed = false;

   if (computerScore ==WINNING_SCORE){

     table.setMessageText("Computer won! " + computerScore +
                                         ":" + kidScore);

   }else if (kidScore ==WINNING_SCORE){
         table.setMessageText("You won! "+ kidScore +
                                    ":" + computerScore);

   }else{

     table.setMessageText("Computer: "+ computerScore +
                                    " Kid: " + kidScore);

   }
 }

 // Проверить, не пересек ли мяч верхнюю или нижнюю границу стола
 private boolean isBallOnTheTable(){

   if (ballY >= BALL_MIN_Y && ballY <= BALL_MAX_Y){

      return true;
   }else {

      return false;
   }
 }
}
```

Поздравляю! Вы закончили разработку вашей второй игры. Скомпилируйте классы и играйте в нее. После того, как вы разберетесь в коде, попробуйте изменить его – я уверен, у вас есть идеи, как сделать эту игру лучше.

Если хотите продолжить программировать игры, взгляните на Robocode – эта программируемая игра, позволяет изучать Java, создавая роботов:

http://robocode.sourceforge.net/?Open&ca=daw-prod-robocode

Материалы для дополнительного чтения



Пособие по сокетам в Java:

http://www.oracle.com/technetwork/java/socket-140484.html

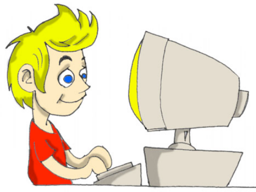
Введение в потоки Java:

http://www-106.ibm.com/developerworks/edu/j-dw-javathread-i.html

Класс `java.awt.Graphics`:

http://download.oracle.com/javase/6/docs/api/java/awt/Graphics.html

Практические задания



1. Класс `PingPongGameEngine` устанавливает координаты белой точки с помощью такого кода:

```
table.point.x = e.getX();
```

Сделайте переменную `point` приватной в классе `PingPongGreenTable` и добавьте `public` метод `setPointCoordinates(int x, int y)`.

Используйте этот метод в классе `PingPongGameEngine`.

2. В нашей игре пинг-понг есть ошибка: после того, как один из игроков выиграл, все еще можно нажать кнопку *S* и игра продолжится. Исправьте эту ошибку.

Практические упражнения для умников и умниц

1. Попробуйте поменять значения `RACKET_INCREMENT` и `BALL_INCREMENT`. Чем больше эти значения, тем быстрее движется мяч и ракетки. Поменяйте код так, чтобы пользователь мог выбирать уровень от 1 до 10. Используйте выбранные значения как коэффициент движения для мяча и ракеток.

2. Когда ракетка ребенка отбивает мяч в верхней части стола, мяч отскакивает по диагонали и быстро падает со стола. Измените программу так, чтобы мяч отскакивал от верхней части стола по диагонали вниз, а от нижней части стола диагонально вверх.

Приложение А. Java архивы - JARs

Очень часто пользователям компьютеров нужно обмениваться файлами. Они могут скопировать файлы на USB flash drive, CD, воспользоваться электронной почтой или просто отправить данные через сеть. Существует специальная программа, которая *сжимает* несколько файлов в один файл-*архив*.

Размер такого архива обычно меньше, чем общий размер всех файлов в отдельности, его быстрее копировать и он занимает меньше места на ваших дисках.

Java поставляется с программой под названием `jar`, которая архивирует несколько Java классов в один файл с расширением `.jar`.

Внутренний формат jar файлов такой же, как и в популярной программе WinZip (мы её использовали во 2-ой главе) .

Следующие три команды иллюстрируют, как можно использовать программу `jar`.

1. Для того, чтобы создать архив jar с файлами с расширением `.class`, откройте черное окно терминала, войдите в папку, где находятся ваши классы и наберите следующую команду:

```
jar cvf myClasses.jar *.class
```

После слова `jar` вы должны указать `опции` для этой команды. В последнем примере *c* – для создания нового архива, *v* – для отображения того, что происходит и *f* означает то, что мы указали имя файла для нового архива.

Теперь вы можете скопировать этот файл на другой диск или отправить по электронной почте вашему другу. Для того, чтобы *распаковать* файлы из архива `myClasses.jar`, наберите такую команду:

```
jar xvf myClasses.jar
```

Все файлы будут распакованы в текущую папку. В этом примере *x* – для извлечения файлов из архива.

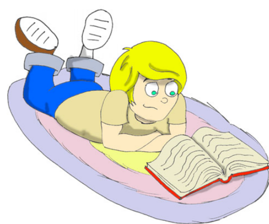
Если вы просто хотите посмотреть содержимое jar-архива без распаковки, воспользуйтесь следующей командой, где *t* – содержание.

```
jar tvf myClasses.jar
```

На самом деле, я предпочитаю использовать программу WinZip для того, чтобы посмотреть, что находится в jar-архиве.

В большинстве случаев Java приложения из реального мира состоят из множества классов, которые находятся в jar-архивах. Хотя и существует много других опций, которые можно использовать с командой `jar`, три примера из этой главы – это главное, что нужно знать для большинства ваших будущих проектов.

Материалы для дополнительного чтения



Jar – Инструмент для java-архивов:
http://download.oracle.com/javase/7/docs/technotes/tools/windows/jar.html

Приложение Б. Советы для работы в Eclipse

В Eclipse существует множество маленьких удобных команд, которые делают программирование на Java немного быстрее. Я перечислил здесь некоторые полезные советы для Eclipse, и я уверен, что вы найдете их еще больше, когда начнете использовать этот инструмент.

- ✓ Если вы видите маленькую звездочку на вкладке с классом – это значит, что класс содержит несохраненные изменения в коде.
- ✓ Выделите имя класса или метода в вашем коде и нажмите клавишу *F3*. Эта команда переместит вас на строку, где объявлен этот класс или метод.
- ✓ Если некоторые строки помечены красными кругами с ошибками, то наведя курсор мыши на кружок, вы увидите текст ошибки.
- ✓ Нажмите *Ctrl-F11*, чтобы запустить последнюю выполненную программу.
- ✓ Поместите курсор после фигурной скобки, и Eclipse выделит другую соответствующую ей закрывающую или открывающую скобку.
- ✓ Для того, чтобы скопировать класс из одного пакета в другой, выберите класс и нажмите *Ctrl-C*. Выберите пакет, в который хотите его скопировать и нажмите *Ctrl-V*.

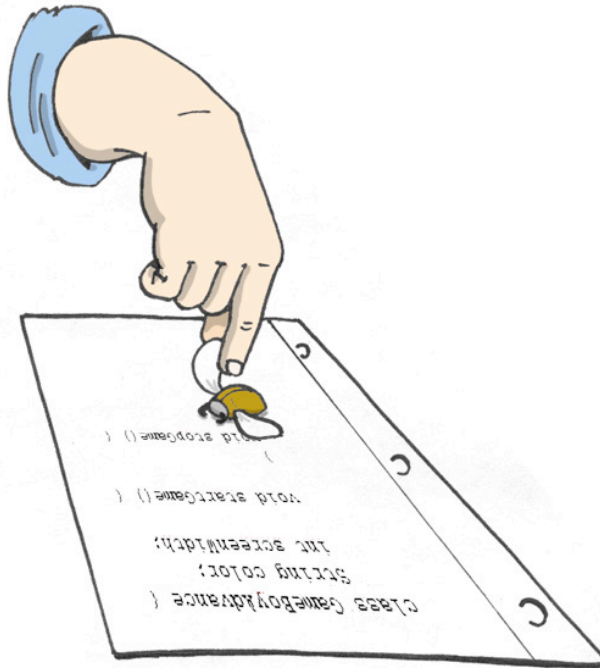
- ✓ Для того, чтобы переименовать класс, переменную и метод, кликните правой кнопкой мыши на нем и выберите *Refactor* и *Rename* из всплывающего меню. Eclipse переименует это имя везде, где оно упоминается.

- ✓ Если в вашем проекте нужны внешние jar-архивы (например, сделанные кем-то другим), кликните правой кнопкой на имени проекта, выберите *Properties*, *Java Build Path* и нажмите кнопку *Add External Jars*.

Отладчик Eclipse

По слухам, около 50 лет назад, когда компьютеры были большими и даже не поместились бы в вашей комнате, вдруг, одна из программ начала выдавать неверные результаты. Эти проблемы были вызваны маленьким жучком (англ. bug), который сидел внутри компьютера где-то в проводах. Когда люди достали этого жука, программа опять стала работать правильно. Начиная с этого момента, *отлаживать программу* (англ. debug) стало означать нахождение причины некорректных результатов программы.

Не путайте логические ошибки с ошибками компиляции. Например, вместо того, чтобы умножить переменную на 2, вы умножите её на 22. Эта опечатка не вызовет никаких ошибок компиляции, но результат будет неверный. Отладчики позволяют шагать в запущенной программе строчка за строчкой с остановками, и вы можете видеть или менять значения всех переменных в любой момент выполнения программы.

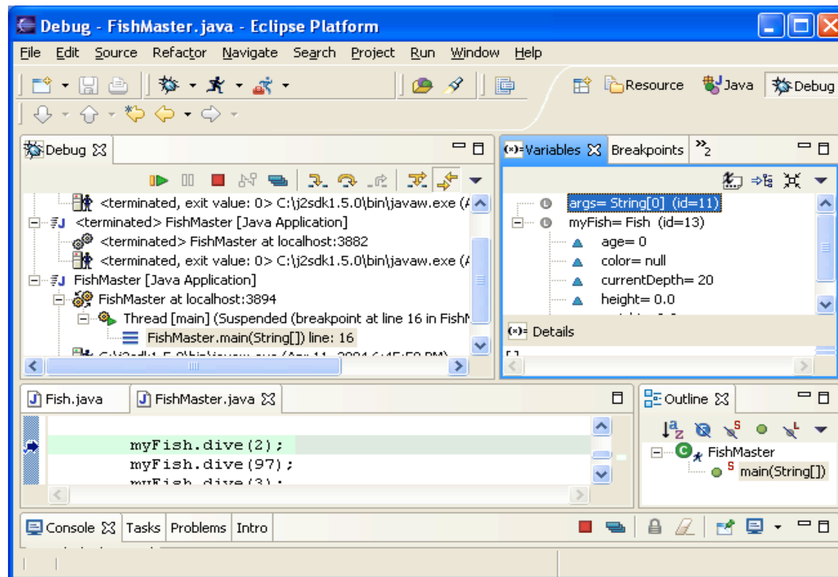


Я покажу, как использовать отладчик Eclipse на примере программы FishMaster из четвертой главы.

Точка остановки (breakpoint) – это строка кода, где вы хотите, чтобы программа остановилась для того, чтобы наблюдать/менять текущие значения переменных, и другую информацию времени выполнения. Для того, чтобы установить точку остановки, просто сделайте двойной щелчок на серой вертикальной полосе слева от линии, где вы хотите остановить программу. Давайте сделаем это в классе FishMaster, на строке `myFish.dive(2)`. Вы увидите круглый маркер на строке с точкой останова. Теперь, выберите в меню *Run, Debug...*. Выберите программу FishMaster и нажмите кнопку *Debug.*

FishMaster запустится *в режиме отладки* и как только программа достигнет строки `myFish.dive(2)`, остановится и будет ждать ваших дальнейших действий.

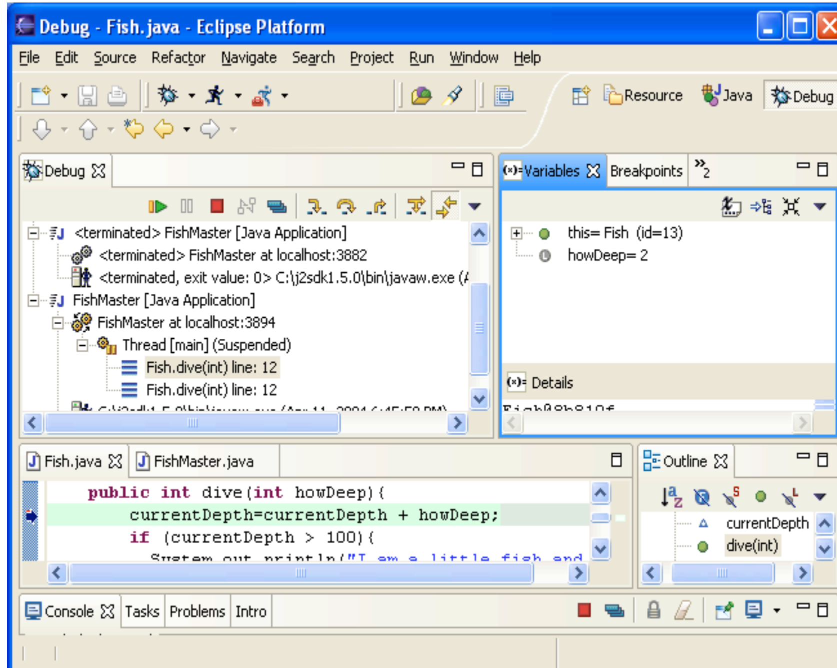
Вы увидите окно отладчика, похожее на это :



В левой нижней части перспективы отладки вы увидите, что строка с точкой остановки подсвечена. Синяя стрелка указывает на строку, которая выполняется. С правой стороны находится окно *Variables*, кликните на маленьком значке плюса у переменной `myFish`. Т.к. переменная указывает на объект `Fish`, вы увидите все элементы этого класса и их текущее состояние, например, `currentDepth=20`.

Стрелки в верхней левой части позволяют продолжить выполнение программы в разных режимах. Первая желтая стрелка означает *войти внутрь* метода. Если вы нажмете эту стрелку (или F5), то попадете внутрь метода dive(). Окно поменяется, и вы увидите значение аргумента `howDeep=2`, как на следующей картинке. Кликните на маленьком плюсе около слова `this` для того, чтобы увидеть текущие значения атрибутов этого класса.

Для того, чтобы поменять значение переменной, кликните правой кнопкой на ней и введите новое значение. Это поможет, если вы не понимаете, почему программа работает неправильно. И если вы хотите поиграть в угадайку – как-бы работала программа, если значение переменной было-бы другим.



Для того, чтобы продолжить выполнение программы по одной строчке, нажимайте следующую стрелку – перешагнуть (или клавишу *F6*).

Если хотите продолжить выполнение программы в быстром режиме, нажмите маленький зеленый треугольник или клавишу *F8*.

Для того, чтобы удалить точку остановки, просто кликните два раза на маленьком круглом маркере, и она исчезнет. Я люблю использовать отладчик, даже если в моей программе нет ошибок – это помогает мне лучше понять, что именно происходит внутри выполняемой программы.

Где ставить точку остановки? Если вы догадываетесь, какой метод может создавать проблемы, поставьте точку остановки перед подозрительной строкой. Если вы не знаете, в чем проблема, просто поставьте ее в первой строке метода `main()` и медленно, пошагово идите по программе.

Приложение В. Как опубликовать Веб-страницу

Интернет-страницы состоят из HTML файлов (с возможными вставками кода на языке JavaScript), изображений, звуковых и видео файлов и т.д. HTML был кратко упомянут в Главе 7, но если вы планируете стать Веб-дизайнером, вам следует уделить больше времени изучению HTML. Начать можно здесь: www.w3schools.com. На самом деле, существует множество веб-сайтов и программ, которые позволяют создавать веб-страницу за несколько минут, даже если вы не знаете, как это делается. Эти программы генерируют HTML, но сам процесс создания будет скрыт от вас. Но, если вы освоили эту книгу, я объявляю вас джуниором - **Младшим Java Программистом** (я не шучу!) и изучение HTML для вас – пустяковое дело.

Для того, чтобы разработать Веб-страницу, обычно, вам достаточно создать один или несколько HTML файлов на диске вашего компьютера, но проблема в том, что ваш компьютер *невидим* для других пользователей Интернета. Поэтому, когда страница готова, вам нужно скопировать (*загрузить*) эти файлы в такое место, где их все могут увидеть. Одно из таких мест – диск на компьютере вашего *Интернет-провайдера*.

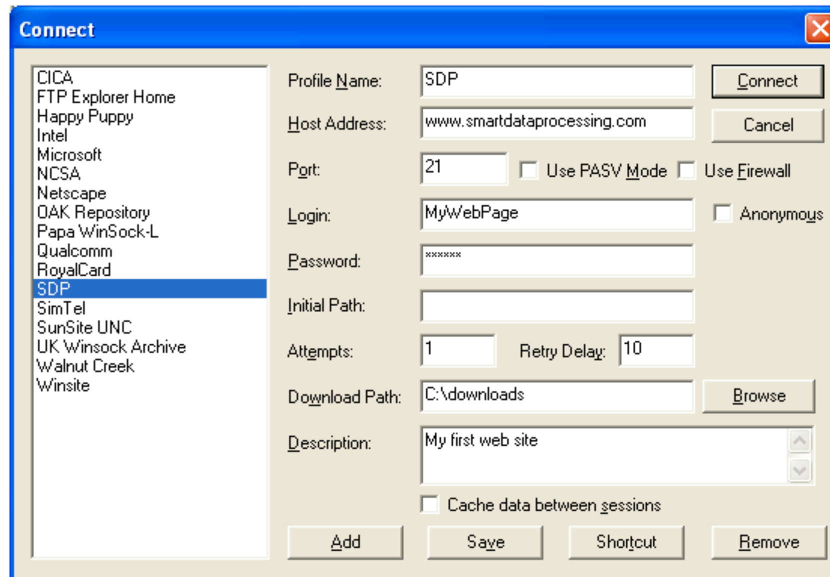
Во-первых, у вас должна быть своя папка на компьютере Интернет-провайдера. Свяжитесь с провайдером по телефону или электронной почте, скажите, что вы создали HTML страничку и хотите ее опубликовать. Обычно они предоставляют следующую информацию:

- ✓ Сетевое имя их компьютера (главный компьютер).
- ✓ Имя папки на их компьютере, где вы можете хранить свои файлы.
- ✓ Интернет-адрес (*URL*) вашей новой странички – вы сможете дать его всем, кто захочет посмотреть вашу страницу.
- ✓ Имя пользователя и пароль, которые вам понадобятся для того, чтобы загрузить новые или изменить старые файлы.

Большинство провайдеров сейчас могут предоставить вам как минимум 100Мб места на их жестких дисках бесплатно, что более чем достаточно для большинства людей.

Еще вам понадобится программа, позволяющая скопировать файлы с вашего компьютера на компьютер провайдера. Копирование файлов с вашего компьютера на компьютер в интернете называется *выкладыванием,* а копирование файлов из интернета на ваш компьютер называется *скачиванием*. Вы можете выкладывать или скачивать файлы с помощью программы *FTP-клиента*.

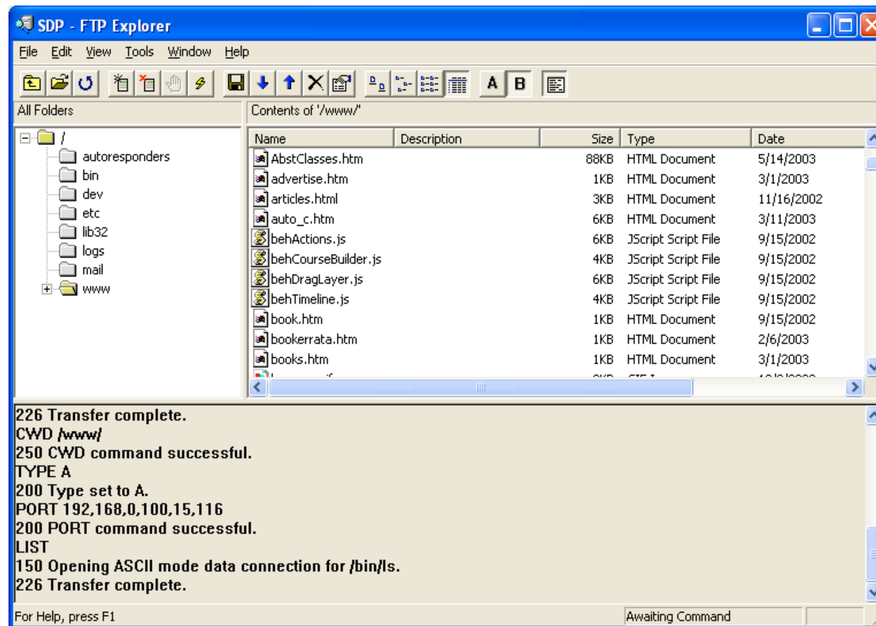
Один из простых в использовании FTP-клиентов – это программа *FTP Explorer*, вы можете скачать его на сайте www.ftpx.com. Установите эту программу и добавьте компьютер вашего Интернет-провайдера в список соединений FTP-клиента – запустите FTP Explorer и первое окно, которое вы увидите, будет окном соединений. Также, вы можете кликнуть на пункт *Connection* в меню *Tools* (в последних версиях этой программы окна и меню выглядят иначе).



Нажмите кнопку *Add* и введите адрес компьютера, имя пользователя и пароль, которые вы получили от вашего Интернет-провайдера. В поле Profile Name просто введите название вашего провайдера. Если вы все сделали правильно, то будет создан новый профиль соединения в списке доступных FTP-серверов.

Нажмите кнопку *Connect* и вы увидите папки на компьютере вашего провайдера. Найдите вашу папку и начинайте выкладывать файлы, как описано ниже.

В панели инструментов находятся две синие стрелки. Стрелка, которая указывает вверх – для выкладывания файлов. Нажмите эту стрелку и вы увидите стандартное окно, где можно войти в папку с вашими HTML файлами. Выберите файлы, которые планируете выложить и нажмите кнопку *Open.* Через несколько секунд вы увидите эти файлы на компьютере вашего провайдера.



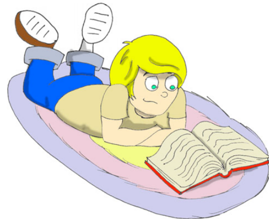
Обратите внимание на нижнюю часть окна и убедитесь, что при выкладывании не было никаких сообщений об ошибках.

Назовите главный файл `index.html`. Тогда URL вашей странички будет короче, и людям не нужно будет печатать имя этого файла. Например, если имя папки на компьютере провайдера www.xyz.com/~David и имя главного файла вашей странички `myMainPage.html`, то адрес вашей странички www.xyz.com/~David/myMainPage.html. Но, если имя главного файла – `index.html`, то адрес вашей странички будет короче – www.xyz.com/~David. Теперь любой, кто знает этот адрес, сможет увидеть ее в Интернете. Если, позже, вы решите изменить эту страницу, просто повторите весь процесс – сделайте изменения на вашем диске. После этого загрузите файлы, и старые файлы заменятся новыми.

Если вы решите стать Веб-дизайнером, следующий язык, который нужно изучить – это JavaScript. Этот язык намного проще, чем Java и позволит вам сделать ваши Веб-страницы ярче и интересней.

Также, вы можете воспользоваться одним из бесплатных сервисов для хранения HTML файлов в интернете, например – Яндекс.Народ (http://narod.yandex.ru/) – прим. переводчика.

Материалы для дополнительного чтения



1. How to make/create you own Web site:
http://www.thesitewizard.com/gettingstarted/startwebsite.shtml

2.The World Wide Web
http://www.w3schools.com/web/web_www.asp

Практические упражнения

Создайте веб-страницу и опубликуйте игру Крестики-Нолики из Главы 7. Для того, чтобы начать, просто загрузите на вашу страницу файлы `TicTacToe.html` и `TicTacToe.class`.

Конец этой книжки

... но у меня есть и другая, правда, на инглише и не для детей. Все-же взгляните, а вдруг понравится? А называется она «Java Tutorial. 24-Hour Trainer». С книжкой идет и DVD, где я записал скринкасты почти ко всем урокам. Ее можно купить на Амазоне – крупнейшем американском интернет-магазине с отличной репутацией. Они доставляют товары во все страны мира. Вот URL этой книженции:

http://www.amazon.com/Java-Programming-24-Hour-Trainer-Yakov/dp/04708896402

Пока-пока!

Индекс

!, 63
&&, 63
||, 62
Access Levels, 176
ActionListener, 100
actionPerformed, 100
Adapters, 114
algorithm, 124
argument, 38
array, 69, 70, 71
ArrayList, 186, 187, 188
arrays, 183
AWT, 76
BorderLayout, 84, 125
BoxLayout, 88
break, 66, 72
Buffered Streams, 155
BufferedInputStream, 155
BufferedOutputStream, 157
callback methods, 121
CardLayout, 90
Casting, 102
catch, 142, 144
checked exceptions, 140
CLASSPATH, 18, 20
conditional if, 63
constructor, 67, 68
continue, 72, 73
date, 169
Debug, 222
do while, 73
Download Eclipse, 23
Eclipse, 23
else if, 64
encapsulation, 182, 183
events, 79
Exception, 140, 143
exceptions, 137
extends, 52, 55
File, 164
FileInputStream, 153
FileOutputStream, 154
FileReader, 160
FileWriter, 160
finally, 145, 146
FlowLayout, 82
frame, 78
FTP, 227
GridBagConstraints, 89
GridBagLayout, 89
GridLayout, 82
HelloWorld, 36
HTML, 116, 117, 119, 122
I/O, 138
IDE, 23
implement, 98
implements, 98
import, 77
instance, 105
instance variables, 67
instanceof, 104
interfaces, 97
jar, 218
javadoc, 59
JDK, 14
JRE, 21, 30
JVM, 13
Layout manager, 78
listeners, 101
Logical Operators, 62
Loops, 71
member variable, 67
method overloading, 171
method signature, 36
new, 67
Object, 54
override, 56
packages, 176, 178
panel, 78

`PATH`, 20
`private`, 179, 180, 189
Program Arguments, 159
`protected`, 179, 182
`public`, 36, 179
run-time errors, 137
scope, 67
`showConfirmDialog`, 100
stack trace, 138
`static`, 36, 67
streams, 152
Swing, 76, 93
`switch`, 65
system variables, 17
this, 68
thread, 201, 204
`throw`, 147
throws, 144
TicTacToe, 117
time, 169
`try`, 141, 144, 146
`try/catch`, 141
`void`, 36, 49
`WindowsListener`, 114
апплеты, 120
аттрибуты, 47
Атрибуты, 40
Веб, 225
Интерфейсы, 98
исходный код, 19
килобайт, 46
Классы, 40
классы-адаптеры, 114
комментарии, 59
конкатенация, 44
Конструктор, 173
Логическое и, 63
логическое или, 62
Логическое не, 63
Метод, 48
Методы, 40
наследование, 52
объект, 43
оператор if, 60
Отладчик, 221
параметр, 48
Переменные, 43
Переопределение методов, 56
случайные числа, 125
слушатель, 97
Слушатель движений мыши, 112
Слушатель клавиш, 112
Слушатель мыши, 112
Слушатель окна, 112
Слушатель фокуса, 112
Слушатель элемента, 112
события, 97
супер-класс, 52
Схемы Размещения, 81
Типы Данных, 42
цикл `while`, 72
цикл for, 71
Чтение текстовых файлов, 160
экземпляр, 105
экземпляр объекта, 43