Центр переподготовки специалистов по математике и информатике
математико-механического факультета
Санкт-Петербургского государственного университета

ЯЗЫК ПРОГРАММИРОВАНИЯ JAVA

Конспект лекций с примерами

от 29.02.2012

Составитель:
преподаватель А.В.Дмитриев

Для студентов вечерней формы обучения

Санкт-Петербург

2008

От автора

Вы, слушатель этого курса, и есть главный его критик и комментатор. Я ценю ваше мнение и хочу знать, что было сделано мной правильно, что можно было сделать лучше и что еще вы хотели бы увидеть в данном курсе. Мне интересно услышать и любые другие замечания, которые вам хотелось бы высказать в мой адрес.
Я жду ваших комментариев и надеюсь на них. Вы можете прислать электронное письмо либо посетить наш сервер и оставить свои замечания там. Дайте мне знать, нравится или нет вам этот курс, а также выскажите свое мнение о том, как сделать курсы более интересными для вас.

Посылая письмо или сообщение, не забудьте указать ваш обратный адрес. Я внимательно ознакомлюсь с вашим мнением и обязательно учту его при подготовке последующих курсов.
Мои координаты:
andrei-dmitriev@yandex.ru

http://in4mix2006.narod.ru

1 Введение

Данное учебное пособие призвано обеспечить слушателей основным материалом для выполнения лабораторных работ в рамках курса по технологиям Java.
За дополнительной информацией рекомендуется обращаться к указанным в тексте и в конце пособия источникам.

1.1 История создания языка

Язык Java создан в 1991 группой Джеймса Гослинга. Первоначальное название – Oak. Переименован в Java, ввиду того, что уже существовал одноименный язык. Причиной создания нового языка послужила необходимость платформонезависимого языка для встраивания в бытовую технику. Первым проектом на Java является система дистанционного управления Star 7. Впоследствии пришло осознание применимости языка для WWW. Одна из первых публикаций от языке Java (Java White Paper) : http://java.sun.com/docs/white/langenv/

1.2 Свойства языка

- Простой
- Объектно-ориентированный
- Распределенный
- Интерпретируемый
- Надежный
- Безопасный
- Архитектурно-нейтральный
- Переносимый
- Высокопроизводительный
- Многопоточный

Первоначальная версия языка базировалась на синтаксисе языка C++. Современная же версия имеет следующие отличия от C++:

- Отсутствие перегрузки операторов.
- Отсутствие множественного наследования.
- Автоматическое согласование типов.
- Отсутствие адресной арифметики.
- Отсутствие деструкторов.

Более подробно:
http://java.sun.com/docs/white/langenv/

1.3 Основные термины и инструментарий разработчика

- Средства разработки существуют для большинства аппаратных платформ.
- Виртуальная машина Java (Java Virtual Machine, JVM) гарантирует единообразие интерфейса с операционной системой.
- Переносимость: «Write once, run everywhere».
- Поставляется с исчерпывающей библиотекой классов JDK (Java Development Kit).
- JRE (Java Runtime Environment) – среда, позволяющая запустить программу, написанную на языке Java.

В состав JDK входит набор утилит для создания Java приложений. Среди них такие как:

- javac – компилятор языка Java.
- java – интерпретатор байт-кода.
- javah - создает заголовочные файлы.
- javadoc - формирует стандартную документацию.
- jar – создание дистрибутивов Java.
- javap – дизассемблер.
- apt – обработчик аннотаций.
- Другие базовые инструменты (appletviewer, jdb, extcheck).

1.4 Среда разработки

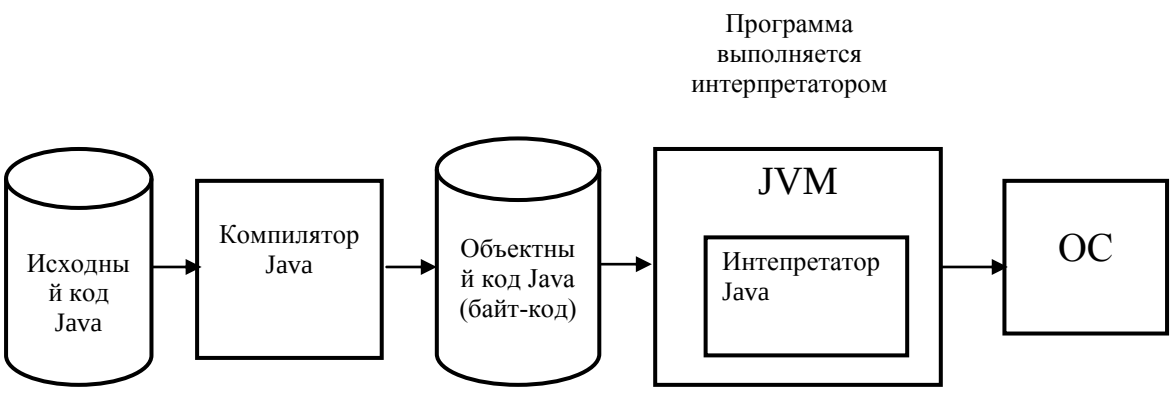
Среда разработки предлагает редактор, компилятор и набор вспомогательных программ, упрощающих разработку ПО.

- NetBeans http://www.netbeans.org/
- Eclipse http://www.eclipse.org/
- JBuilder, CodeGear http://www.borland.com/
- IntelliJ Idea http://www.jetbrains.com/

2 Создание приложения на Java

2.1 Схема разработки приложения

Общая схема разработки и запуска приложения на Java следующая.



2.2 Запуск приложения в командной строке

В текстовом редакторе создайте файл с исходным кодом программы на языке Java: Hello.java
Скомпилируйте исходный код в промежуточный код командой

javac Hello.java

В результате получится файл Hello.class
Запустить приложение командой

java Hello

Если запускаемый класс принадлежит именованному пакету (пространство имен, которому принадлежит класс, package), например, **testpackage,** то скомпилированный класс необходимо поместить в папку, имя которой совпадает с именем указанного в классе пакета (если пакеты вложены друг в друга, то в несколько вложенных каталогов).

Запуск программы командой:

java testpackage.Hello

2.2.1 Простой пример – вывод строки на консоль

Единственный метод класса Hello является точкой входа программы (метод main). В результате работы выводит на консоль строку «Hello, World!».

```
public class Hello{
    public static void main(String []args){
        System.out.println("Hello, World");
    }
}
```

2.2.2 Определение и вызов метода

В данном классе 2 метода и один из них (main) вызывает другой (sayHello), передавая строку (name) в качестве параметра.

```
public class Hello{
    public static void main(String []args){
        sayHello("World");
    }
    public static void sayHello(String name){
        System.out.println("Hello, "+name+"!");
    }
}
```

2.2.3 Передача параметров через командную строку

Поскольку выполнение программы начинается с метода main, в него можно передать различные параметры из консоли, при запуске программы. Здесь первый введенный параметр интерпретируется как имя (args[0]).

```
public class Hello{
    public static void main(String []args){
        sayHello(args[0]);
    }
    public static void sayHello(String name){
        System.out.println("Hello, "+name+"!");
```

```
    }
}
```

2.2.4 Обработка неправильного ввода (отсутствие параметра при запуске)

В предыдущем примере мы предполагали, что программа запускается хотя бы с одним параметром. В ином случае сгенерировалось бы исключение. Здесь мы обрабатываем ситуацию запуска без параметра – при этом выводится «Hello, World!».

```
class Hello{
  public static void main(String []args){
    if (args.length==0){
      sayHello("World");
    }
    else{
      sayHello(args[0]);
    }
  }
  public static void sayHello(String rc){
    System.out.println("Hello, "+rc+"!");
  }
}
```

2.2.5 Перечисление всех параметров и использование вспомогательного класса

Создаем вспомогательный класс (SayHello). Он предоставляет функциональность для главного класса (Hello) – вывод строки на печать (метод printString). Необходимый параметр (имя) передается в конструктор при создании экземпляра класса SayHello.

```
public class Hello{
    public static void main(String []args){
        for (int i=0;i<args.length;++i) {
            FriendlyClass friend =new FriendlyClass (args[i]);
            friend.PrintString();
        }
    }
}

class FriendlyClass{
    String name;
    SayHello(String s){name=s;} //конструктор класса
    void PrintString(){
        System.out.println("Hello, "+name+"!");
    }
}
```

3 Знакомство со средой программирования

3.1 NetBeans IDE

- Бесплатная интегрированная среда разработки.
- Не требует дополнительной настройки.
- Инструменты для разработки и отладки Java SE, ME и EE приложений.
- 100% Java.
- Открытый исходный код.
- Поддерживается сообществом.
- Поддержка контроля версий.

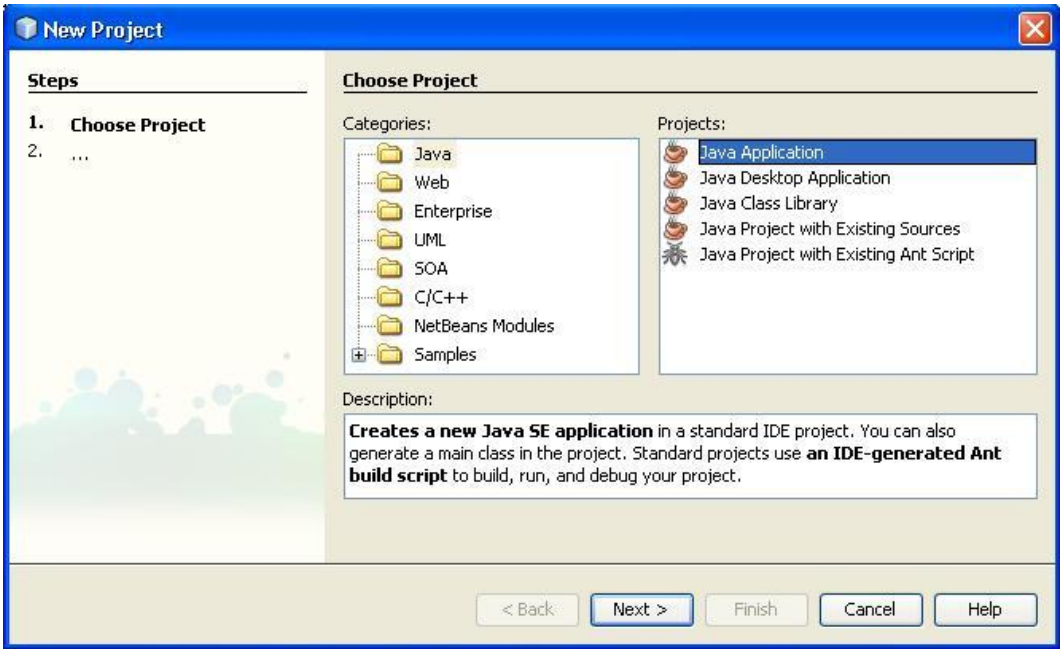
- Модульная архитектура.
- Поддержка рефакторинга.
- Проекты базируются на Ant (инструментарий для управления программными проектами на Java).

3.2 Установка среды

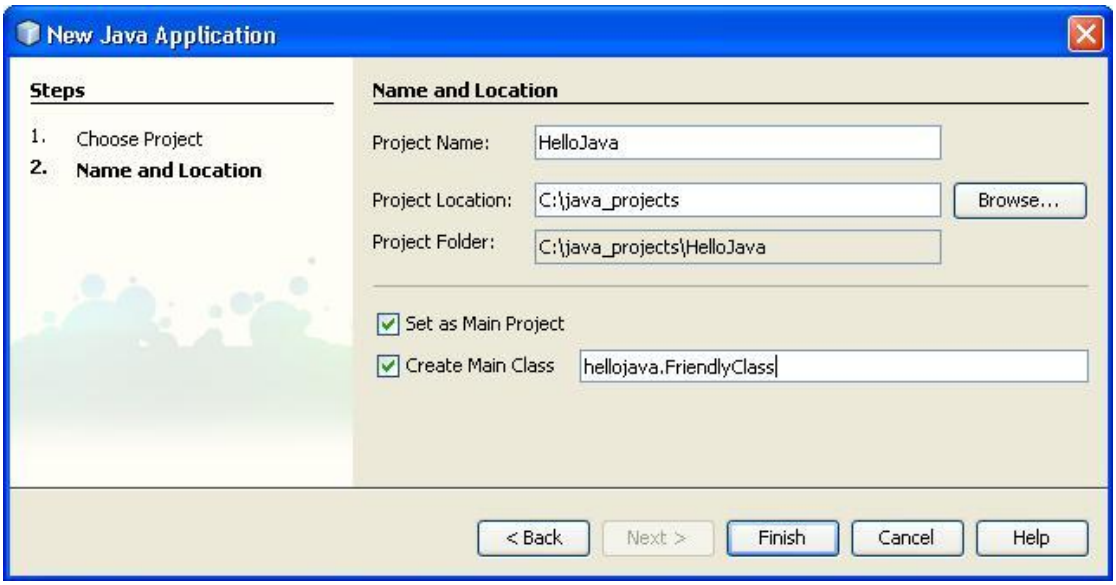
- Дистрибутив NetBeans можно скачать с сайта http://www.netbeans.org/
- Необходима JDK (может поставляться вместе с дистрибутивом NetBeans). Можно скачать с сайта http://java.sun.com/
- Для установки с параметрами по умолчанию достаточно последовательно отвечать на вопросы инсталлятора.

3.3 Лабораторная работа №1: «Разработка приложения HelloJava»

1) Запустите среду разработки.
2) Выберите в меню **File** пункт **New Project**.
3) В появившемся диалоговом окне выберите первый пункт «Java» в левом списке. В списке, появившемся справа, выберите пункт «Java Application». Нажмите **Next**.



4) В поле **Project Name** введите имя проекта (HelloJava) и укажите путь к нему в поле **Project location**. Здесь это «C:\\java_projects», но можно выбрать любой другой.
Оставьте отмеченными флажки **Set as Main Project** и **Create Main Class.**
В текстовом поле рядом с последним флажком введите имя главного класса приложения. Оно должно начинаться с имени пакета (hellojava.FriendlyClass). Можете оставить значение по умолчанию (hellojava.Main).
Нажмите **Finish**.



5) Проект готов! Тело программы пока что пусто.
В левом окне, во вкладке **Projects** вы увидете структуру файлов проекта. Пока в него входит лишь один класс – FriendlyClass.java. Он расположен в пакете hellojava, созданном по умолчанию.

6) В правой части окна замените строку
// TODO code application logic here
следующей строкой
System.out.println("Hello, Java!");
Метод main является точкой входа в программу – именно с него начнется ее выполнение. Пока внутри метода лишь одна строчка. При запуске программы на консоль будет выведен текст «Hello, Java!».

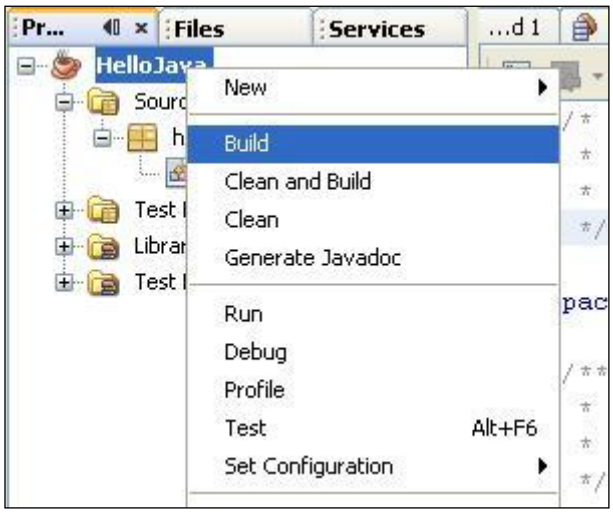
Полный текст программы имеет следующий вид.

```
public class FriendlyClass {
    public static void main(String []args){
        System.out.println("Hello, Java!");
    }
}
```

7) В нижнем левом углу приложения нажмите Output (если данной кнопки нет, выберите в меню Window пункт Output>Output). Появившаяся область экрана – консоль среды разработки. В ней можно видеть результаты различных действий с приложением.
8) **Компиляция проекта.**

Нажмите правой кнопкой мыши на название проекта HelloJava. В появившемся контекстном меню выберите пункт **Build.**

Альтернативный вариант меню **Build**, пункт **Build Main Project** (горячая клавиша F11). Работает лишь если ваш проект выбран главным (стартовым).



На консоли должна появится следующая надпись:

BUILD SUCCESSFUL (total time: ? seconds)

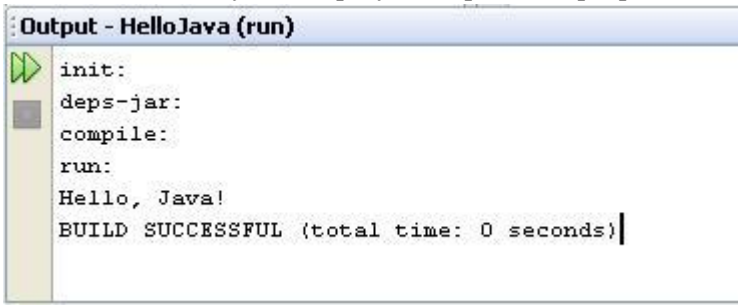
– знак того, что компиляция прошла успешно.

9) Запуск проекта.

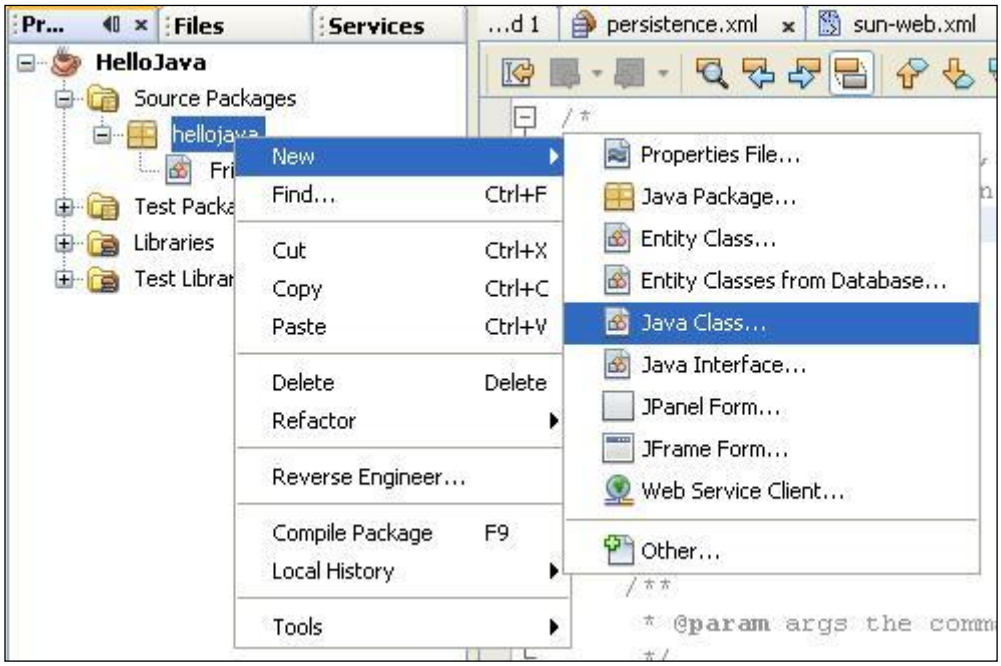
Вновь войдите в контекстное меню для проекта (пункт 8) и выберите пункт Run для того чтобы запустить программу.

Альтернативный вариант меню **Run**, пункт **Run Main Project** (горячая клавиша F6). Работает лишь если ваш проект выбран главным (стартовым).

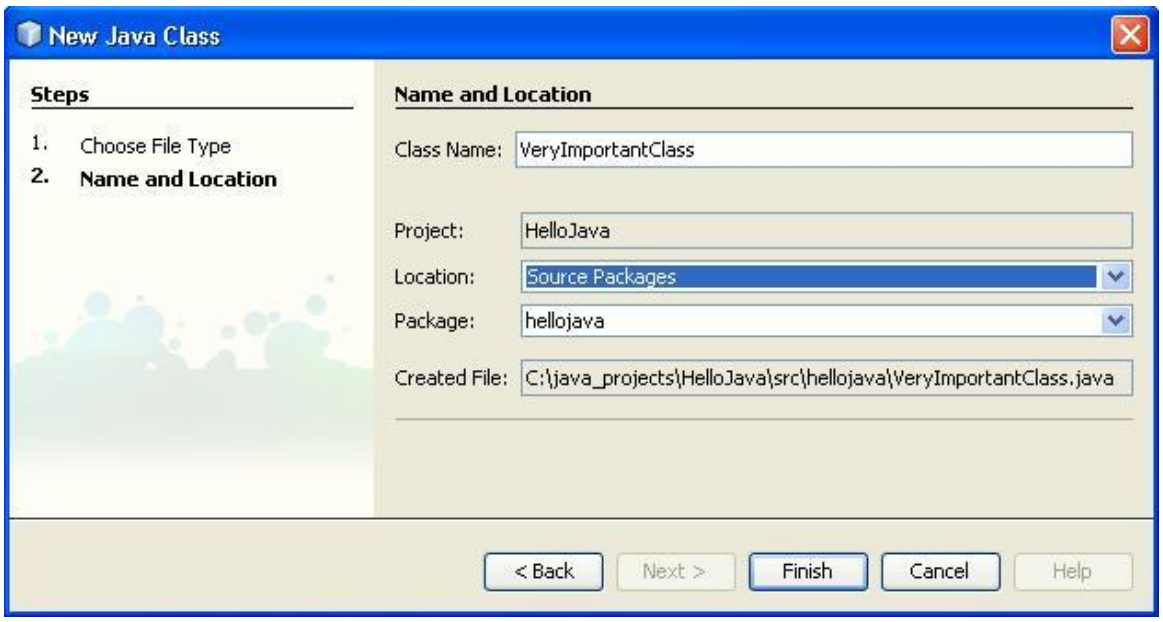
В консоли можно увидеть результат работы программы – строку «Hello, Java!».



10) Добавление нового класса.
Щелкните правой кнопкой мыши по проекту (или пакету hellojava). В появившемся контекстном меню выберите **New > Java Class..**



11) Введите имя нового класса в поле **Class Name**.
В поле **Location** оставьте пункт **Source Packages**.
В поле **Package** выберите имя пакета (hellojava).
Нажмите **Finish**.



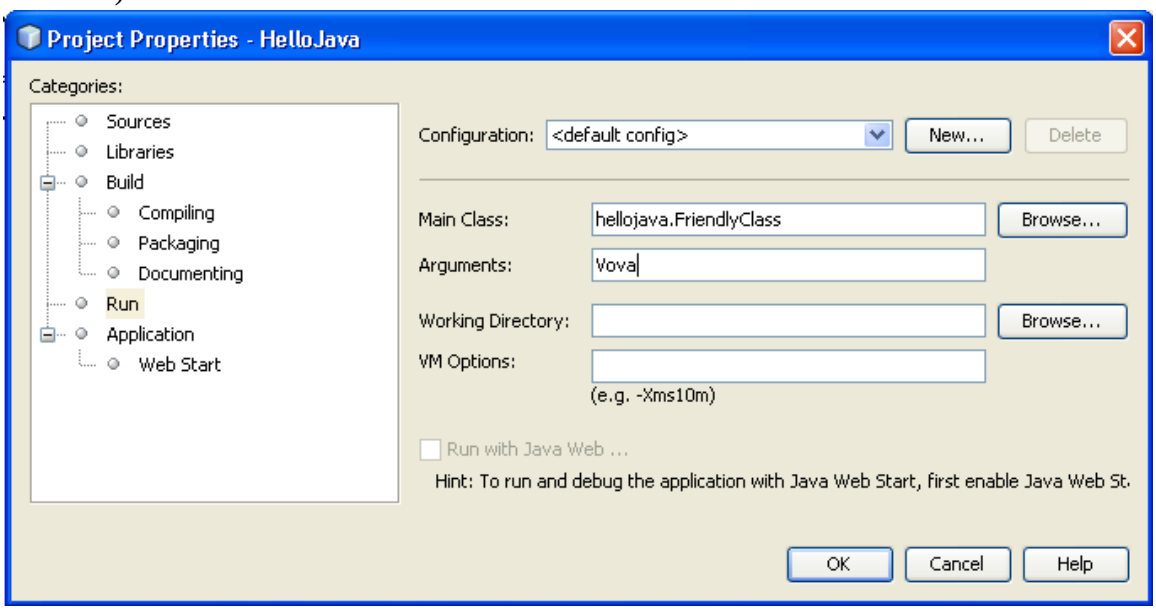
12) Немного об интерактивности
Созданное нами ранее приложение отлично работает, но оно всегда ведет себя одинаково, что бы ни делал пользователь. Добавим в него немного интерактивности.
1) Обратите внимание на параметры главного метода:
main(String []args)

Параметры можно задать при запуске приложения, разделяя их пробелами (их может быть произвольное число). Впрочем, в этом нет никакого смысла, пока мы не добавим в метод код, обрабатывающий вводимые параметры.

2) Перепишем код следующим образом:

```
public class FriendlyClass{
   public static void main(String []args){
      System.out.println("Hello, "+args[0]+"!");
   }
}
```

3) Теперь, передав в качестве параметра запуска свое имя, мы получим индивидуальное приветствие. Сделаем это, не выходя из среды разработки.
Нажмите правой клавишей мыши на названии проекта HelloJava. В появившемся контекстном меню выберите последний пункт **Properties**. Откроется диалоговое окно свойств проекта (в нем можно задать много полезных настроек!).
Выберите пункт **Run** из списка (слева).
Введите параметры командной строки в поле **Arguments** (можно ваше имя, например, Vova).



4) Запустите программу

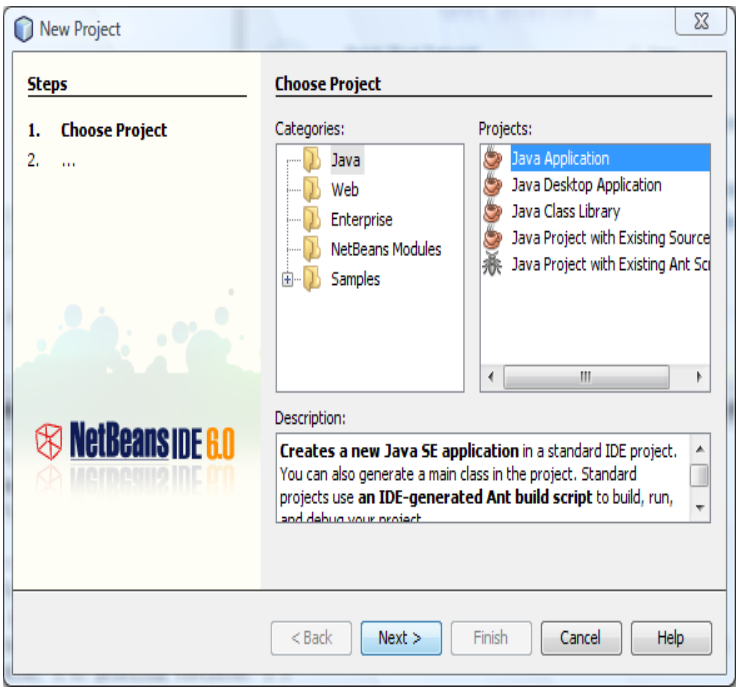
5) На консоль будет выведено личное приветствие «Hello, Vova!».

3.4 Возможности NetBeans

- Текстовый редактор
- Дизайнер графического интерфейса
- UML
- Системы контроля версий
- Отладчик
- И др.

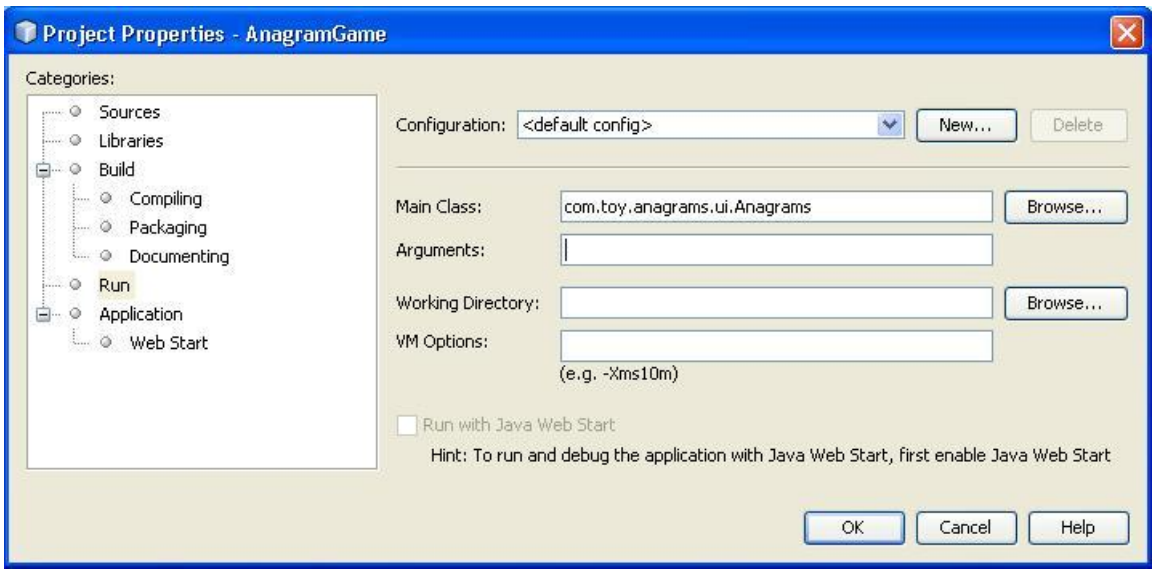
3.5 Выбор типа проекта

General (Java)	Обычные приложения и библиотеки на Java.
Web	Вэб-приложения. Среди шаблонов:JSF и Struts.
Enterprise	EJB и вэб-сервисы.
Mobile	Приложения для портативных устройств.
NetBeans Plug-in Modules	Разработка подключаемых модулей для среды разработки.
Samples	Примеры приложений.



Каждый проект может быть создан из существующих исходных кодов или на основе готового сценария Ant.

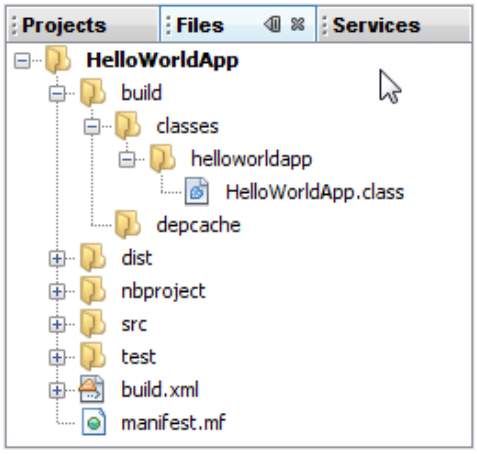
3.6 Настройка окружения



Можно конфигурировать:

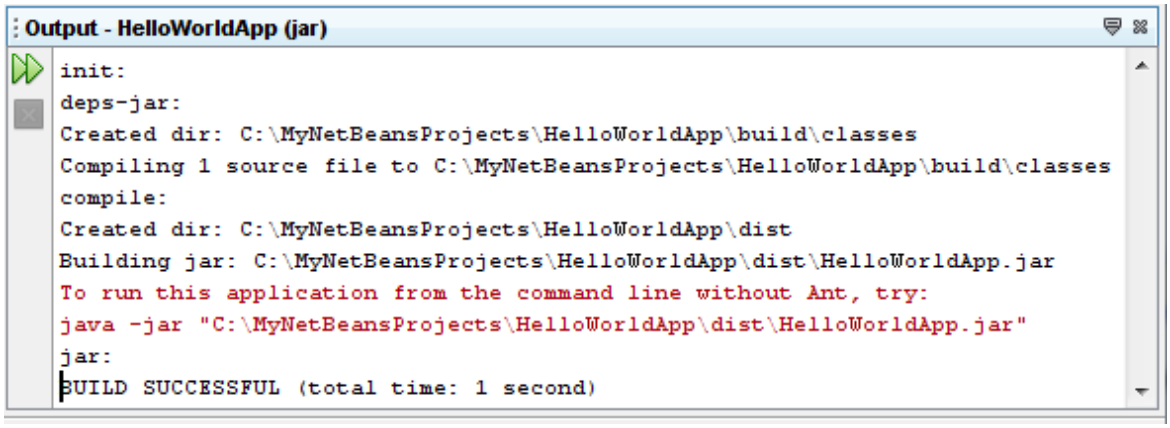
- Используемый проектом JDK.
- Переменные окружения.
- Зависимости между проектами.
- Используемые библиотеки.
- Параметры виртуальной машины и приложения.

3.7 Файловая структура



- При создании файла можно использовать шаблоны документов.
- Каждому типу файла соответствует редактор с соответствующими возможностями.
- Файлы .java можно организовывать в пакеты.
- Реализована удобная навигация по файлам проектов.

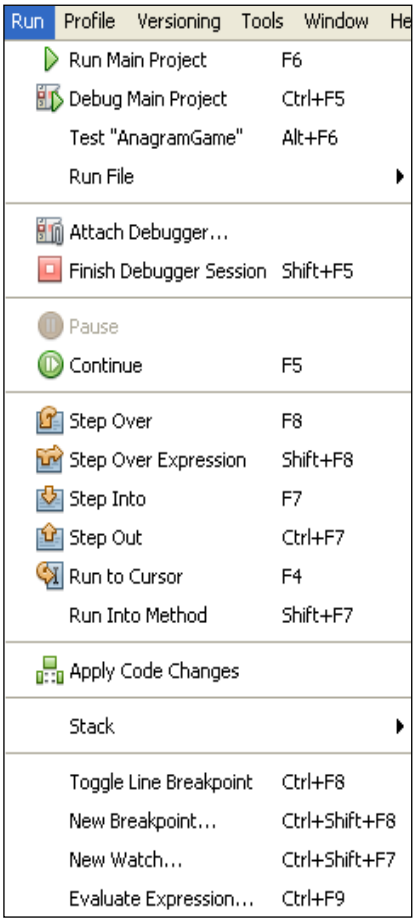
3.8 Компиляция

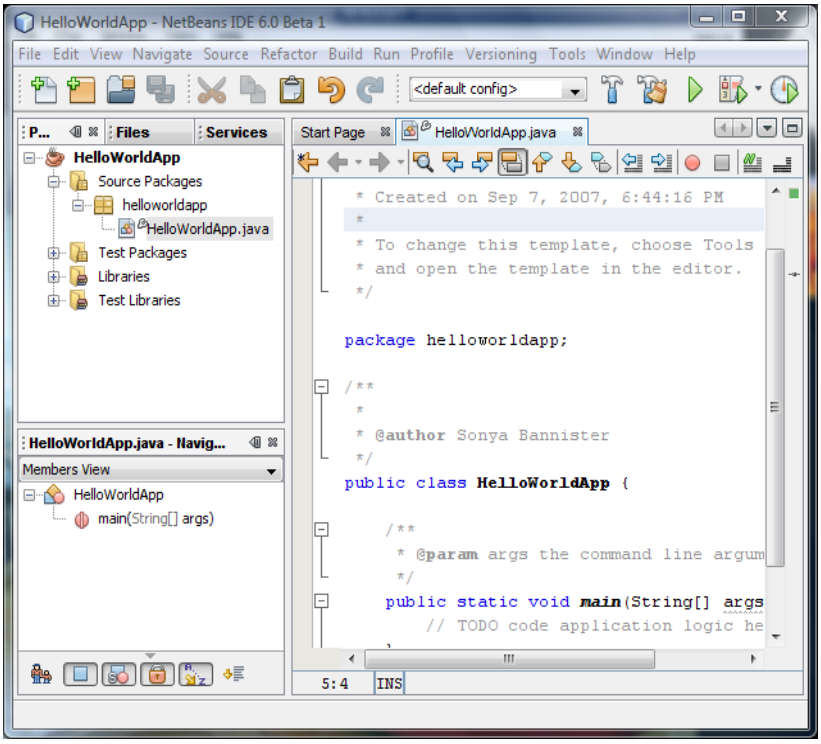


- Ошибки компиляции подсвечены как гиперссылки (на место ошибки в исходном коде) в окне для вывода.
- Файлы проекта можно упаковать в дистрибутив (JAR, WAR) в указанный каталог (по умолчанию: projectdir/dist).
- Для собственных нужд можно редактировать файл build.xml.

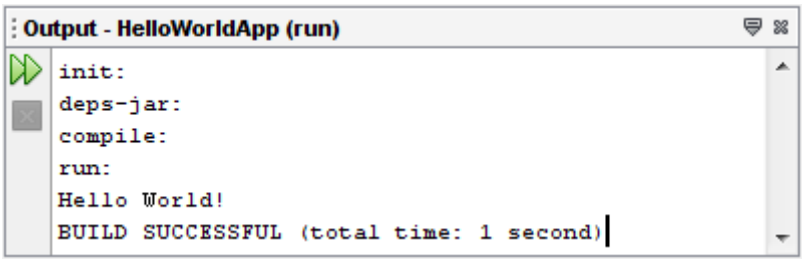
3.9 Запуск

- Для запуска можно использовать различные конфигурации.



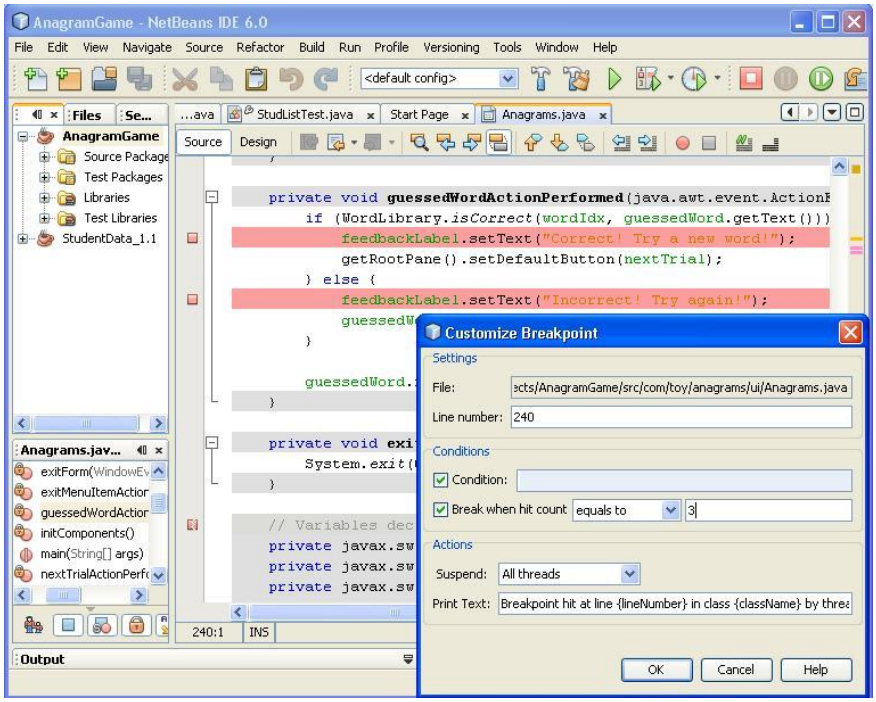


- Результат работы приложения отображается в окне.

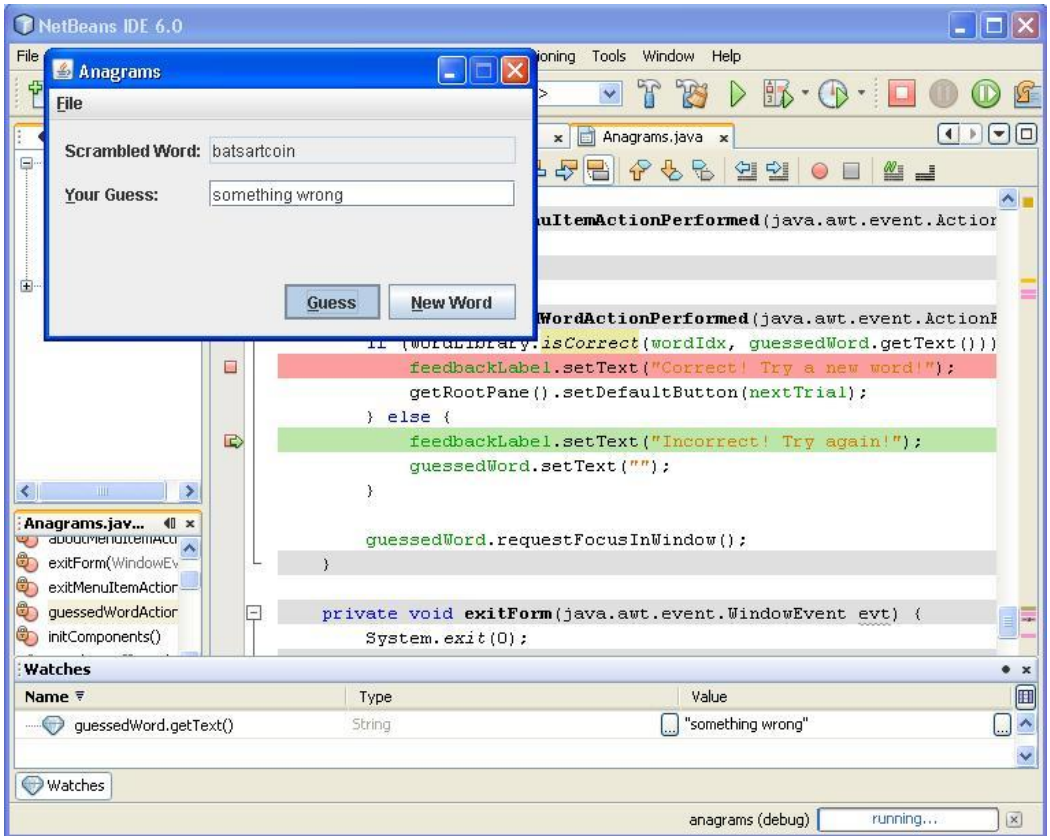


3.10 Отладка

- Установка точек прерывания исполнения.
- Выполнение программы по шагам.
- Отслеживание значений переменных.



Отладка (прерывание выполнения)



Отладка (отслеживание значений)

Примеры проектов

Дистрибутив среды включает примеры типовых приложений, оформленных в виде проекта.

Anagram Game	Простое игровое приложение, использующее такие компоненты Swing, как JFrame, JLabel, и JTextField. Также содержит тестовый класс JUnit.
GUI Form Examples	Демонстрирует возможности работы NetBeans GUI Builder с тремя типичными схемами размещения (layouts).
Document Editor	Простой текстовый редактор; демонстрирует использование действий, карт ресурсов и других функциональных возможностей *Swing Application Framework.*
Mars Rover Viewer	Простой просмотрщик изображений; демонстрирует использование действий, заданий в фоновом потоке, карт ресурсов и т.д.
Client Editor	Простой редактор клиентских данных; демонстрирует использование связывания Bean компонентов. Включает примеры конвертации и валидации.

Ссылки

- Сайт NetBeans:
  - > http://netbeans.org/
- Онлайн-курсы:
  - > http://javapassion.com/
- NetBeans IDE Field Guide by Patrick Keegan, Ludovic Champenois, etc.

4 Основы языка Java

4.1 Основы: Программа DragonWorld

```
package heroes;

public class HelloDragonWorld {
    public static void main(String []args){
        System.out.println("Hello DragonWorld!");
     }
}
```

4.2 Основы: пакет

- Пакет – это совокупность классов и подпакетов, объединенных общим именем.

```
package mydragons;
public class Dragon{//реализация }
//использование
import mydragons.Dragon;
Dragon red = new Dragon(Dragon.RED);
Dragon black = new mydragons.Dragon(Dragon.BLACK);
```

4.3 Основы: класс

1. Класс – это базовая сущность ООП, обладающая определенными свойствами.
2. Любая программа на языке Java представляет собой класс.

```
package animals.slowanimals;

public class Reptile {
   public void eat(Bird b){
       b.wasEaten = true;
```

```
    }
}
```

4.4 Основы: поле

Поле – это именованное свойство класса или объекта.
Поле может относиться как к каждому объекту, так и к классу в целом.

```
package animals.slowanimals;

public class Reptile {
   private int length;
}
```

4.5 Основы: объект

- Объект – это переменная, типом которой является соответствующий класс.
- Объект также называют экземпляром класса.

```
package animals.slowanimals;
//класс:
public class Reptile {
   private int length;
}
…
//объект:
Reptile gecko = new Reptile();
```

4.6 Основы: метод

Метод – это программная функция, относящаяся к определенному объекту или классу. Области, откуда метод может быть доступен, определяются модификаторами метода.

```
package animals.slowanimals;

public class Reptile {
   private int length;
   public void eat(Bird b){
       length++;
   }
}
```

4.7 Основы: наследование

Класс может заимствовать методы другого класса. Язык Java поддерживает операцию наследования:

```
// наследование производится с помощью
// ключевого слова extends
public class Dragon extends Reptile {
    //внутреннее поле класса
    private String magic = "fire";
    public String getMagic(){
         //возврат результата
         return fire;
    }
}
```

4.8 Область видимости переменной

Каждая переменная может быть использована только в ее области видимости.

```
int i = 10;
System.out.println(i); //область видимости ограничится ближайшими
фигурными скобками.

for (int j = 0; j < 100; j++){
    System.out.println(j);// j видна внутри
                          //блока for
}
System.out.println(i);//i видна после блока
                      //for
System.out.println(j); //j: переменная вне
               //области видимости - ошибка
```

4.9 Модификаторы

1. Модификаторы доступа являются реализацией принципа инкапсуляции в языке Java.
2. Изменяя модификаторы, можно контролировать область видимости
   - Полей
   - Методов
   - классов

4.9.1 Модификаторы полей

- Отсутствие модификатора – поле доступно только из текущего пакета.
- static – поле принадлежит структуре класса. Одно значение присуще всем экземплярам.
- final – поле не может быть изменено.
- transient – не участвует в процессе сериализации по умолчанию.
- private – поле не может быть использовано нигде кроме данного класса или его экземпляра.
- protected – поле не может быть использовано нигде кроме данного класса и всех его наследников.
- volatile – значение этого поля будет обновляться каждый раз при обращении к нему.

4.9.2 Модификаторы методов

- Отсутствие модификатора – доступен только из данного пакета.
- public – метод доступен из любого пакета (публичный API).
- final – не может быть переопределен в наследнике.
- static – метод принадлежит классу.
- abstract – метод не имеет реализации.
- synchronized – запрещено одновременное выполнение метода на разных потоках.
- native – метод имеет реализацию на языке C.
- private – метод не может быть использован ниоткуда кроме данного класса (его объекта).
- protected – метод не может быть использован ниоткуда кроме данного класса (его объекта) и всех его наследников (их объектов).

4.9.3 Модификаторы классов

- Отсутствие модификатора – доступен в текущем пакете.
- public – класс доступен из любого пакета (публичный API).

- final – не может расширяться методом наследования от него.
- abstract – класс является абстрактным, нельзя создать объект этого класса.
- static – допустимо только для вложенных классов.

4.10 Конструктор

- Конструктор – это метод, создающий экземпляр класса.
- Не имеет заданного возвращаемого значения.
- Имеет то же имя, что и класс.
- В классе всегда присутствует конструктор по умолчанию, если явно конструктор не задан.

```
public class Dragon{
  private String color = gold;
     public Dragon(String newColor){
          color = newColor;
     }
}
```

4.11 Вызов метода

1. Вызов метода – это обращение к члену класса по его имени.
2. Результат вызова метода – выполненные операторы и возвращаемое значение (если указано).

```
public class Dragon{
    private String name = "Kesha";
    public String sayName (){
          return "I am"+name;
    }
}
Dragon dragon = new Dragon(); //конструктор
System.out.println(dragon.sayName()); //метод
```

4.12 Виртуальный метод

- Метод класса может быть переопределён в классах-наследниках так, что конкретная реализация метода для вызова будет определяться во время исполнения.

- Программисту необязательно знать точный тип объекта для работы с ним через виртуальные методы: достаточно лишь знать, что объект принадлежит классу или наследнику класса, в котором метод объявлен.

- Метод, который, будучи описан в потомках, замещает собой соответствующий метод везде, даже в методах, описанных для предка, если он вызывается для потомка.

4.13 Повторное использование имен

4.13.1 Переопределение

Методы предка и наследника могут быть одноименными.

```
class Reptile{
    public void move(){/*ползти*/ }
}

class Dragon extends Reptile{
    public void move(){/*лететь*/}
```

```
}
Dragon d = new Dragon();
d.move(); //обращение к методу экземпляра Dragon
Reptile r = new Reptile();
r.move(); //обращение к методу экземпляра Reptile
```

4.13.2 Сокрытие

Статические методы принадлежат классу.

```
class Reptile{
    public static void move(){}
}
class Dragon extends Reptile{
    public static void move(){}
}
Dragon d = new Dragon();
Reptile r = new Reptile();
d.move(); //обращение к методу Dragon
r.move(); //обращение к методу Reptile
//Рекомендуется использовать вызовы класса:
Reptile.move();
Dragon.move();

Reptile r1 = new Dragon(); //обращение к методу Reptile
```

4.13.3 Перегрузка

Методы выполняют схожую функцию над разными типами данных.

```
class HungryDragon {
    public void eat(int foodWeight){…}
    public void eat(String foodWeight){
    //разбор строки на значимое целое число
        eat(Integer.parseInt(foodWeight));
    }
}
HungryDragon hd = new HungryDragon();
hd.eat(10);
hd.eat("10");
```

4.13.4 Затенение

1) Локальная переменная делает одноименную глобальную переменную невидимой в локальной области.
2) Так делать не рекомендуется.

```
public class Dragon {
    static String type = "Just Dragon";
    public static void main(String [] s){
        String type = "Black Dragon";
        //выведет "Black Dragon"
        System.out.println(type);
    }
}
```

4.13.5 Перекрытие

- Использование имен существующих методов и полей вносит в программу путаницу.
- Использование существующих имен классов недопустимо.

```
public class BadExample {
    static String System;
    public static void main(String [] s){
       System.out.println("A string");
    }
}
```

4.14 Правила оформления

1) Основная цель хорошего оформления программы – она должна выглядеть понятной.
2) Среди требований можно отметить:
   - Отступы.
   - Мнемоничность имен.
   - Разделение операторов.
   - Использование фигурных скобок.
   - И т.д.
3) Рекомендации от компании Sun Microsystems:
http://java.sun.com/docs/codeconv/

4.15 Передача параметров

Метод класса может получать до 255 параметров. Фактический параметр считается локальной переменной метода.

```
class Dragon {
    public void eat(Object obj){}
    public void fly(String direction){}
}

Dragon d = new Dragon();
d.eat(new Girl());
d.fly(Direction.WEST);
```

4.16 Поле this

Каждый объект имеет ссылку на самого себя, которая может использоваться для формирования ссылки на перегруженный конструктор и на поля объекта.

```
class Dragon {
    private int weight;
    public Dragon(int weight){
       //затенение
       this.weight = weight;
    }
}
```

4.17 Поле super

Каждый объект имеет ссылку на объект-предок, которая позволяет организовать восходящие вызовы конструкторов.

```
class Dragon extends Reptile{
    int flyingSpeed;
```

```
    public void attack(Object obj){
//обращение к предку за выполнением
//базовых действий
    super.attack(obj);
    burn(obj); //метод класса Dragon
}
   // Dragon умеет атаковать, как Reptile,
   // а заодно сжигать жертву
    public void burn(Object obj){ //сжечь объект
   }
}
```

4.18 Статический блок

Класс может иметь в себе участок кода, выполняющийся при инициализации класса.

```
class Dragon {
//статический блок выполняется до того, как создался первый
экземпляр класса
  static {
        System.out.println("Dragons are alive!");
  }
  //конструктор может не выполниться ни разу
  //В то время как статический инициализатор
  //выполнится при загрузке
  public Dragon(){
         System.out.println("New dragon has born");
  }
}
```

4.19 Порядок инициализации

Инициализация членов класса и выполнение статического инициализатора происходит в порядке их описания в классе.

```
class Dragon {
 static int dragonCount = 10;
 static {
//ошибка - переменная dragonEnemy еще не проинициализирована
        System.out.println(dragonEnemy);
//OK - переменная dragonCount уже проинициализирована
        System.out.println(dragonCount);
  }
  static String dragonEnemy = "Phoenix";
}
```

4.20 Константы

Константа - это именованное значение, неизменяемое средствами языка Java.

```
  class Dragon {
   final static int headCount = 1;
}
Dragon.headCount  = 3;//ошибка – попытка присвоить значение
константе. Дракон – не Змей Горыныч!
   class Gorinich {
    //MutableFloat – это класс-хранилище дробного числа и
позволяющий изменять его
   final static MutableFloat headCount = new MutableFloat(1);
}
```

```
Gorinich.headVount.setValue(3);//OK, т.к. значение указателя не
меняется, меняется только содержимое
```

4.21 Константы в статическом блоке

Инициализацию константы можно отложить, но только до времени выполнения статического блока класса. Допустимо произвести только одно присваивание константе.

```
class Dragon {
 final static int headCount;//нет
     //инициализации
   static {
        //OK – 1-я инициализация
        headCount = 1;
        //Ошибка – константа уже присвоена
        headCount = 3;
   }
}
```

4.22 Абстрактный класс

Класс является абстрактным, если имеет модификатор abstract.
Класс должен быть помечен этим модификатором, если у него хоть один абстрактный метод (помечен словом **abstract** и не имеет реализации).

```
abstract class FlyingThing {
  protected String name;
  abstract public void fly();
  public String  getName(){
    return name;
  }
}
//ошибка,абстрактный класс не  может  иметь реализаций
FlyingThing aThing = new FlyingThing();
```

4.23 Наследование от абстрактного класса

Как правило, абстрактный класс служит для создания базы дерева наследования классов.

```
class Dragon extends FlyingSomething{
  public fly(){
    //реализуем полет куда-нибудь
    flySomewhere();
  }
}
//OK – создавать экземпляры можно
Dragon d = new Dragon();
//OK – создание ссылки на абстрактный класс и инициализация
конкретным классом
FlyingSomething fs = new Dragon();
```

4.24 Реализация интерфейса

Интерфейс – это сущность, предназначен для формирования структуры реализующего его класса или для наследования другим интерфейсом.

```
 public interface Flying{
 // класс, реализующий данный интерфейс, // должен предоставить
 // реализацию для этого метода
      int speed();
 }
```

Класс может реализовывать множество интерфейсов. Реализующий класс должен реализовать все методы интерфейса. Интерфейсы могу наследоваться друг от друга. Реализация позволяет снабдить класс дополнительными свойствами.

```
public class Dragon implements Flying {
     protected int speed;
     public int speed(){
  return speed;
     }
}
public class RedDragon extends Dragon{
     public int speed(){
        return 2*speed;
     }
     public long distance(){…}
     public long burn(Object obj){…}
}
```

5 Введение в ООП

5.1 Инкапсуляция

Инкапсуляция (сокрытие данных) - объединение в одной сущности данных и методов работы с ними.

- Состояние объекта определяется значениями его полей.
- Сокрытие данных осуществляется с помощью установки модификаторов, влияющих на видимость членов класса.
- В языке Java используется несколько уровней сокрытия.

5.2 Наследование

Наследование – возможность класса-наследника приобретать признаки класса-предка.

```
abstract class Animal {
    abstract void talk();
}
class Frog extends Animal {
     void jump() { //прыгать}
}
class Bird extends Animal {
    void fly() {//летать }
}
```

Класс не может приобретать свойства нескольких других классов через механизм наследования:

```
//ошибка, разрешено наследоваться только от одного класса.
class FlyingFrog extends Frog, Bird{…}
```

5.3 Полиморфизм

Полиморфизм – способность наследников по-другому реализовывать возможности предков. Объект способен проявлять признаки своего предка.

Экземпляр класса может выступать как экземпляр любого класса-предка данного класса.

```
class Animal {
  void talk(){…};
}
class Dog extends Animal {
   void talk() { System.out.println("Woof!"); }
}
class Cat extends Animal {
   void talk() { System.out.println("Meow!"); }
}
```

Экземпляр класса Dog или Cat одновременно является экземпляром класса Animal:

```
Animal myDog = new Dog();
 Animal myCat = new Cat();
```

5.4 Ромбическое наследование

Для передачи свойств сразу от нескольких сущностей был введен новый тип данных – интерфейс. Использую дополнительную сущность, можно добиться того же результата как и при множественном наследовании:

```
interface Flying {
    void fly();
}
class Frog {
     void jump() { //прыгать}
}
class Bird implements Flying{
    void fly() {//летать }
}
class FlyingFrog extends Frog implements Flying{…} //разрешено
```

5.5 Примеры

5.5.1 Наследование и инкапсуляция

Класс Pixel наследует от класса Point. При этом он расширяет базовый класс, инкапсулируя поле color (цвет), а также переопределяет метод предка clear, добавляя обнуление данного поля.

```
public class Pixel extends Point {
    Color color;
    public void clear() {
        super.clear();
        color = null;
    }
}
```

5.5.2 Полиморфизм (точка и пиксель)

В классе-предке (Point) и классе-наследнике (Pixel) определен метод show. При вызове данного метода выполняется разный код – в зависимомти от того, к объекту какого класса идет обращение.

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

```
        Point list[] = new Point[3];
        list[0] = new Point(0, 0);
        list[2] = new Point(2, 2);
        list[1] = new Pixel(1, 1);
        for (int i = 0; i < list.length; i++) {
            list[i].show();
        }
    }
}

class Point {
    int x, y;
    Point(int x, int y){
        this.x = x;
        this.y = y;
    }
    void show(){
        System.out.println("I'm a Point["+x+","+y+"]");
    }
}

class Pixel extends Point{
    int color;
    final static int defaultColor = 0;
    Pixel(int x, int y){
        super(x, y);
        this.color = defaultColor;
    }

    Pixel(int x, int y, int c){
        super(x, y);
        this.color = c;
    }
    void show(){
        System.out.println("I'm a Pixel["+x+","+y+"]"+color);
    }
}
```

5.5.3 Интерфейсы

В интерфейсе KeyboardListener определены 2 метода, принимающие код нажатой клавиши. Класс CustomComponent реализует интерфейс – а значит, обязан раелизовать оба метода интерфейса.

```
interface KeyboardListener {
    public void altKeyPressed(int keyCode);
    public char getChar(int keyCode);
}
class  CustomComponent implements KeyboardListener{
    public void altKeyPressed(int keyCode){
        // обработка нажатия клавиши
    }
    public char getChar(int keyCode){
        //преобразование кода клавиши в символ
    }
}
```

```
CustomComponent customComp = new CustomComponent();
customComp.altKeyPressed(27);
customComp.getChar(KeyEvent.VK_Z);
```

5.5.4 Использование интерфейса в качестве типа

В сигнатуре метода можно указать интерфейс в качестве типа параметра. Тогда при обращении к методу параметром может выступать объект любого класса, реализующего данный интерфейс.

```
void handleKeyboard(KeyboardListener kbl, int code) {
   System.out.println(kbl.getChar(code));
}
```

6 Исключения

6.1 Определение

- Исключение в Java — это объект, который описывает исключительное состояние, возникшее в каком-либо участке программного кода.
- Когда возникает исключительное состояние, создается объект класса Exception.
- Этот объект пересылается в метод, обрабатывающий данный тип исключительной ситуации.
- По «следам» стека программы можно найти данный метод – и причину ошибки.

6.2 Возникновение исключения

Совершаем преднамеренную ошибку – делим на ноль.

```
class SimpleMistake {
      public static void main(String args[]) {
            int d = 0;
            int a = 42 / d;
      }
 }
```

- Тип исключения указывает на причину его возникновения.
- Стек вызовов позволяет отследить путь, по которому был достигнут проблемный код.
- Стандартный обработчик выдает номер строки кода, в котором произошло исключение.

6.3 Обработка исключения

```
try{
      doSomethingDangerous(); //опасный метод
   }
catch (CatchedExceptionType e) {
      treatDanger(); //обработка исключения
   }
//CatchedExceptionType – класс, к которому
      принадлежит исключение e
```

Ловим свою ошибку и выводим информацию на консоль.

```
class SimpleMistake{
      public static void main(String args[]){
            try{
           int d = 0; //выполнится
           int a = 42 / d;

           int z = a + d; //не выполнится
                  }
            catch (ArithmeticException e) {
                  System.out.println(«Деление на ноль");
            }
      }
}
```

6.4 Виды исключений

- Проверяемое
  - FileNotFoundException, IOException, …
  - После такой ситуации зачастую требуется восстановление состояния программы.
  - Обязательны для описания при определении метода.
- Ошибка
  - класс Error и его наследники.
- Исключение времени исполнения
  - Оно же непроверяемое.
  - RuntimeException и все наследники.
  - Восстановление после таких ситуаций обычно не производится.

6.5 Примеры исключений

ArithmeticException	ошибка при вычислениях – например, деление на 0
ArrayIndexOutOfBoundsException	выход за пределы массива
FileNotFoundException	если не обнаружен запрошенный файл
IOException	любое исключение в системе ввода/вывода; включает предыдущее
OutOfMemoryError	реакция на нехватку памяти
VirtualMachineError	ошибка внутри вируальной машины Java
AWTError	ошибка при работе графического интерфейса

6.6 Требования к коду

- Если метод вызывает проверяемое исключение, то он должен :
  - > либо обработать его
  - > либо передать исключение выше по стеку вызова

- Неудовлетворяющий этому правилу код не компилируется.

6.7 Каскад обработчиков

Иногда одного обработчика недостаточно – создаем несколько, на разные типы исключений.

В данном примере можно поймать сразу 2 исключения разного рода:

- Деление на ноль (при запуска программы без параметров),
- Выход за пределы массива (обращение к несуществующему 42-му элементу).

```
class MultiCatch {
    public static void main(String args[]) {
        try {
            int a = args.length;
            System.out.println("a = " + a);
            int b = 42 / a;
            int c[] = { 1 };
            c[42] = 99;
        } catch (ArithmeticException e) {
            System.out.println("div by 0: " + e);
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("array index out of bound: " + e);
        }
   }
}
```

6.8 Вложенные блоки

Также можно вкладывать один блок try-catch в другой. Здесь в методе main вызывается метод procedure и обрабатывается ситуация деления на ноль. Попытка выхода за пределы массива обрабатывается уже внутри метода procedure() – во вложенном блоке try-catch.

```
class MultiNest {
    static void procedure() {
        try {
            int c[] = { 1 };
            c[42] = 99;
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("array index out: " + e);
        }
}

    public static void main(String args[]) {
        try {
            int a = args.length();
            System.out.println("a = " + a);
            int b = 42 / a;
            procedure();
        } catch (ArithmeticException e) {
            System.out.println("div by 0: " + e);
        }
     }
 }
```

Этот метод сам обрабатывает свое же исключение, поэтому наружу исключение не передается.

```
public static void doCalculation(){
  try {
        riskyCode();
  } catch(ArrayIndexOutOfBoundsException e) {
        tryToHandle();
   }
}
```

6.9 Явно инициированное исключение

Новое исключение создается посредством вызова конструктора. Конструктор принимает строку, описывающую причину исключения. Генерирование исключения происходит с помощью оператора throw.

```
class ThrowDemo {
   void riskyMethod(int value) {
     try {
        //какие-то действия
        if (value == 1){
          //бросаем исключение
           throw new IllegalArgumentException("Can't be 1");
        }
     }catch (IllegalArgumentException e) {
         prepareToClose();
        //передача исключения выше
         throw e;
     }
 }
```

6.10 Описание исключений

После имени метода указывается тип (типы) возможных исключений, которые метод может сгенерировать: throws.

```
class ThrowsDemo {
      static void riskyMethod() throws IllegalAccessException {
     //do something
     if (condition){
           throw new IllegalAccessException("fake");
      }
      }
      public static void main(String args[]){
           riskyMethod();
      }
}
```

6.11 Блок finally

Используя данный блок, добиваемся того, что некий набор действий выполнится независимо от того, сгенерировано исключение или нет.

```
try {
   //какие-то действия
      doSomething();
   //иногда бросает исключение
      doSomethingRisky();
}catch (NumberFormatException e) {
    handleState(); //обрабатываем исключение
}finally {
     //действия, которые нужно выполнить независимо
```

```
        // от того, были ли сгенерировано исключение или нет
        doFinalStuff();
      }


class FinallyDemo {
    static void exceptionDemoMethodA() {
        try {
            System.out.println("inside exceptionDemoMethodA");
            throw new RuntimeException("demo");
        }  finally {
            System.out.println("exceptionDemoMethodA's finally");
        }
    }
    static void exceptionDemoMethodB() {
        try {
           System.out.println("inside exceptionDemoMethodB");
           return;
        } finally {
           System.out.println("exceptionDemoMethodB's finally");
   }
}
    public static void main(String args[]) {
        try {
         exceptionDemoMethodB();
         }catch (Exception e) {}
        exceptionDemoMethodB();
   }
}
```

6.12 Пользовательские классы-исключения

Создаем свой класс исключений на основе класса Exception.

```
class MyException extends Exception {
    private int detail;
    MyException(int a) {     detail = a:   }
    public String toString() {    return "MyException[" + detail + "]";   }
}

class ExceptionDemo {
    static void compute(int a) throws MyException {
        System.out.println("called computer + a + ").");
                if (a > 10)
                    throw new MyException(a);
        System.out.println("normal exit.");
    }
```

Используется обычным способом.

```
    public static void main(String args[]) {
        try {
            compute(1);
            compute(20);
        } catch (MyException e) {
            System.out.println("caught" + e);
        }
   }
}
```

7 Работа со строками

7.1 Пример метода toString()

Каждый класс реализует метод toString (поскольку это метод класса Object). Реализация определяет строковое представление объекта класса.
Можно переопределить данный метод и задать собственное, «говорящее» строковое представление – в данном случае выводятся на печать координаты точки.

```
class Point {
    int x, y;
    Point(int x, int y) {     this.x = x;     this.y = y;   }
    public String toString() {
        return "Point[" + x + ", " + y + "]";
    }
}
class ToStringDemo {
    public static void main(String args[]) {
        Point p = new Point(10, 20);
        System.out.println("p = " + p);
    }
 }
```

7.2 Поэлементный доступ

Существует ряд методов, позволяющих обращаться к элементам строки – символам (char).
В примере сначала в цикле выводятся символы строки s (обращение к ним идет при помощи метода charAt). Затем выводится подстрока строки s, для которой указаны индексы начального и конечного символов (end и start).

```
class GetCharsDemo {
    public static void main(String args[]) {
        String s = "This is a demo of the getChars method.";
        int start = 10;
        int end = 14;
        for (int i=0; i< s.length(); i++){
            System.out.println(s.charAt(i));
        }
        char buf[] = new char[end - start];
        s.getChars(start, end, buf, 0);
        System.out.println(buf);
    }
 }
```

7.3 Сравнение строк

Строки можно сравнивать при помощи метода equals – он вернет true если строки одинаковы. Если регистр букв не должен учитываться при сравнении, следует использовать equalsIgnoreCase.

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

```
        String s1 = "Hello";
        String s2 = "Hello";
        String s3 = "Good-bye";
        String s4 = "HELLO";
        System.out.println(s1 + " equals " + s2 + " -> " + s1.equals(s2));
        System.out.println(s1 + " equals " + s3 + " -> " + s1.equals(s3));
        System.out.println(s1 + " equals " + s4 + " -> " + s1.equals(s4));
        System.out.println(s1 + " equalsIgnoreCase " + s4 + " -> " +
                           s1.equalsIgnoreCase(s4));
    }
 }
```

7.4 Оператор равенства

Следует быть осторожным при использовании оператора равенства в случае строк (поскольку строки являются ссылочными типами). В приведенном примере результат первого сравнения – true, а второго – false.

```
class EqualsNotEqualTo {
    public static void main(String args[]) {
        String s1 = "Hello";
        String s2 = new String(s1);
        System.out.println(s1 + " equals " + s2 + " -> " + s1.equals(s2));
        System.out.println(s1 + " == " + s2 + ", -> " + (s1 == s2));
    }
}
```

7.5 Сравнение и сортировка строк

Здесь массив строк сортируется в алфавитном порядке. Строки сравниваются при помощи метода compareTo(). При необходимости меняются местами (сортировка «пузырьком»).

```
class SortString {
  static String arr[] = {"Now", "is", "the", "time", "for", "all",
                         "good", "men", "to", "come", "to", "the",
                         "aid", "of", "their", "country" };
  public static void main(String args[]) {
    for (int j = 0; j < arr.length; j++) {
      for (int i = j + 1; i < arr.length; i++) {
          if (arr[i].compareTo(arr[j]) < 0) {
              String t = arr[j];
              arr[j] = arr[i];
              arr[i] = t;
          }
      }
      System.out.println(arr[j]);
      }
   }
}
```

7.6 StringBuffer

Строка – неизменяемый объект, соответственно при каждой операции вставки, конкатенации и т.п. создаются новые объекты. Для интенсивной работы со строками более подходят классы StringBuffer и StringBuilder.

```
class StringBufferDemo {
```

```
    public static void main(String args[]) {
        StringBuffer sb = new StringBuffer("Hello");
        System.out.println("buffer = " + sb);
        System.out.println("length = " + sb.length());
        System.out.println("capacity = " + sb.capacity());
    }
}
```

8 Многопоточные приложения

Поток - последовательность выполняемых операций внутри программы. Потоки позволяют исполнение нескольких действий одновременно. Поддержка многопоточного программирования встроена в язык Java.

Поток может находиться в одном из следующих состояний.

- New – создан, но не запущен.
- Running – активно использует ресурс CPU.
- Blocked – ожидает ресурсов или событий.
- Resumed – готов к работе после остановки типа block или suspend.
  - > Поток помещается в очередь, ожидающую доступа к ресурсам ЦП.
- Suspended – поток перестал использовать ЦП.
  - > Самостоятельно или был принудительно снят с исполнения.

8.1 Класс java.lang.Thread

Данный класс предоставляет абстракцию, позволяющую выполнять инструкции языка Java в отдельном потоке. Имеет следующие конструкторы.

Thread()	Создает новый класс.
Thread(String name)	Создает новый именованный класс-поток.
Thread(Runnable target)	Создает новый класс-поток на основе экземпляра Runnable, чей метод run() будет исполняться в отдельном потоке.
Thread(Runnable target, String name)	Создает новый именованный класс-поток на основе Runnable.

Среди методов класса можно выделить:

public static Thread currentThread()	Возвращает ссылку на активный в настоящее время поток.
public void interrupt()	Прерывает исполнение потока.
public final boolean isAlive()	Определяет, работает ли поток.
public void run()	Метод запускается в отдельном потоке.
public void start()	Запускает выполнение метода run() в отдельном потоке.
public void join(Thread other)	Приостанавливает исполнение текущего потока до тех пор пока поток other не завершится.
public void stop()	Завершает выполнение потока.

public void resume()	Восстанавливает выполнение потока.
public void sleep(long delay)	Приостанавливает выполнение потока на delay миллисекунд.
public void yield()	Метод пробует отказаться от выполнения на ЦП в пользу других потоков.

Класс Thread имеет несколько публичных полей, отражающих относительные приоритеты исполнения: **MAX_PRIORITY, MIN_PRIORITY и NORM_PRIORITY.**

Создать новый поток можно одним из следующих способов:

- Наследование от класса Thread

Наследник класса Thread переопределяет метод run(). Вызов метода start() у экземпляра подкласса запускает выполнение потока. Операторы, помещенные в методе run() выполняются виртуальной машиной в новом потоке.

- Реализация интерфейса **java.lang.Runnable**

Интерфейс Runnable должен быть реализован классом, который должен выполняться в отдельном потоке. Для этого класс должен реализовывать всего один метод - run().. Экземпляр класса должен быть передан конструктору класса Thread в качестве аргумента. Запуск потока осуществляется методом start() класса Thread.

8.2 Пример многопоточного приложения на основе класса Thread

Следующий пример выводит на консоль сообщения от двух потоков исполнения. Можно отметить, что потоки выполняются одновременно.

```
public class ConcurrentDemo{
    public static void main(String []args){
        new ConcurrentThread1().start();
        new ConcurrentThread2().start();
    }
}
class ConcurrentThread1 extends Thread{
    public void run(){
        for (int i=0;i<=10000;i++) System.out.println("#1: "+i);
    }
}
class ConcurrentThread2 extends Thread{
    public void run(){
        for (int i=0;i<=10000;i++) System.out.println("#2: "+i);
    }
}
```

8.3 Пример создания потока методом реализации интерфейса Runnable

```
class SimpleCombat implements Runnable{
    public void run(){
       //конкретные действия поединка
       takePosition();
       fire();
    }
}
SimpleCombat combat = new SimpleCombat();
```

```
new Thread(combat).start();
```

8.4 Объект синхронизации

Метод или блок кода, выполнение которого не должно прерываться исполнением того же самого кода, но на другом потоке, можно помечать модификатором synchronized.
Для задания синхронизации между методами нужен некий объект. Обычно таким объектом является разделяемый ресурс. Синхронизованный метод становится «владельцем» монитора объекта. Объектом синхронизации может быть ссылочный тип (объект) или класс.

8.4.1 Синхронизация доступа к состоянию объекта

Методы, обращающиеся к состоянию объекта, синхронизованы для того чтобы исключить одновременное изменение состояния.

```
public class SeniorStable {
    private int horseCount = 10;
    public void synchonized returnHorse(){
        horseCount++;
    }
    public void synchonized takeHorse(){
        horseCount--;
    }
    public int synchonized getHorseCount(){
            return horseCount;
     }
}
```

8.5 Межпотоковое взаимодействие

Следующие методы класса Object используются для реализации взаимодействия потоков между собой.

public final void wait() throws InterruptedException	Заставляет поток ожидать когда какой-то другой поток вызовет метод notify() или notifyAll() для данного объекта.
public final void notify()	Пробуждает поток, который вызвал метод wait() для этого же объекта.
public final void notifyAll()	Пробуждает все потоки, который вызвали метод wait() для этого же объекта.

8.6 Группы потоков

Потоки могут объединяться в группы с целью управления большим количеством потоков одновременно. Потоку разрешено получать информацию о своей группе.
Запрещено о других группах и о более крупных группах (являющихся надгруппами).

Если поток или группа потоков не обрабатывает исключение, то оно обрабатывается JVM. Для установки собственного обработчика можно использовать метод `uncaughtException(Thread thread, Throwable throwable)` класса `java.lang.ThreadGroup`.

Следующий пример демонстрирует создание группы потоков.

```
public class thGroup{
    public static void main(Strings []args){
        ThreadGroup mainGr=new ThreadGroup("main group");
        Thread thread1=new Thread(mainGr, "thread1");
        ThreadGroup minorGr=new ThreadGroup(mainGr, "minor
group");
    }
}
```

8.7 Исполнение по расписанию

В следующем примере сообщение выдается через несколько секунд после инициализации приложения.

```
public class TimerReminder {

    Timer timer;

    public TimerReminder(int seconds) {
        timer = new Timer();
        timer.schedule(new RemindTask(), seconds*1000);
    }

    class RemindTask extends TimerTask {
        public void run() {
            System.out.format("Time's up!%n");
            timer.cancel(); //Terminate the timer thread
        }
    }

    public static void main(String args[]) {
        System.out.format("About to schedule task.%n");
        new TimerReminder(5);
        System.out.format("Task scheduled.%n");
    }
}
```

Дополнительная литература:

- Doug Lea, Concurrent Programming in Java

8.8 Аплет «бегущая строка»

Следующий пример демонстрирует работу потока, обновляющего состояние графического интерфейса аплета через равные интервалы времени.

```
import java.applet.*;
import java.awt.*;

public class ScrollText extends Applet implements Runnable{
    Thread workThread=null;
    String str;
    int xpos;
```

```
    int ypos;
    int leftedgepos;
    int pause;

    public void paint(Graphics g){
        if (--xpos<leftedgepos)
            xpos=size().width;
        g.drawString(str,xpos,ypos);
    }
    public void init(){
        str=getParam("Greet","Hello!");
        xpos=getParam("X",100);
        ypos=getParam("Y",50);
        leftedgepos=getParam("LEFTPOS",-100);
        pause=getParam("PAUSE",100);
        setFont(new Font("Helvetica", Font.BOLD,24));
    }

    public String getParam(String p, String d){
        String s=getParameter(p);
        return s==null? d:s;
    }
    public int getParam(String p, int d){
        try {
            String s=getParameter(p);
            if (s!=null)
                d=Integer.parseInt(s);
        }catch (NumberFormatException e){ }
        return d;
    }

    public void start(){
        workThread=new Thread(this);
        workThread.start();
    }
    public void run(){
        while (true){
            repaint();
            try {
                Thread.sleep(pause);
            }catch (InterruptedException e) { }
        }
    }
    public void stop(){
        if (workThread!=null){
            workThread.stop();
        }
    }
}
```

8.9 Пример: часы реального времени

Пример показывает страничку, содержащую электронные часы. Для отсчета секунд используется отдельный поток.

```
//<applet code= "ClockOnPage.class" width =300 height=300>
//</applet>
public class ClockOnPage extends Applet{
    public void init(){
        add(new SimpleClockDisplay());
        setSize(200,75);
    }
    public void start(){}
    public void stop(){}
}

class Pulse implements Runnable{
    private ClockDisplay clock;
    public Pulse(ClockDisplay c){
        clock = c;
    }
    public void init(){
        clock.updateTime();
        new Thread(this).start();
    }
    private synchronized void tick(){
        try{
            wait(1000);
            clock.updateTime();
        }catch(InterruptedException e){}
    }
    public void run(){
        while(true){ tick();}
    }
}

interface ClockDisplay{
    public void updateTime();
}

class SimpleClockDisplay extends Label implements ClockDisplay{
    Pulse p;
    public SimpleClockDisplay(){
        setFont(new Font("Helvetica",Font.BOLD,20));
        setAlignment(CENTER);
        setForeground(Color.yellow);
        setBackground(Color.black);
        p=new Pulse(this);
        p.init();
    }
    public void updateTime(){
        Date date=new Date();
        int h=date.getHours();
        int m=date.getMinutes();
        int s=date.getSeconds();
```

```
        String hours,minutes,seconds,am_pm;

        if (h==0){
            hours=new String("12");
            am_pm=new String("am");
        }else if (h>12) {
            hours=new String(""+(h-12));
            am_pm=new String("pm");
        }else {
            hours=new String(""+h);
            am_pm=new String("am");
        }
        if (m<10) {
            minutes=new String("0"+m);
        }else{
            minutes=new String(""+m);
        }

        if (s<10) {
            seconds=new String("0"+s);
        }else{
            seconds=new String(""+s);
        }
        setText(“”+hours+":"+minutes+":"+seconds+am_pm);
    }
}
```

9 Обзор пакетов и классов JDK

В состав библиотек JDK входит множество классов, имеющих часто используемую функциональность. Глава состоит из обзора некоторых из них.

9.1 Разбор строки

Класс StringTokenizer предназначен для разбора строки на слова по определенному программистом критерию. В примере строка разбивается на слова, использую разделители ‘=’ и ‘:’.

```
class StringParser {
    static String in = "title=Core PC Library:" + "author=Piter
Norton:" + "isbn=0-07-882199-1:"+ "ean=9 780078 821998:" +
"email=p.norton@starwave.com";
    public static void main(String args[]) {
        StringTokenizer st = new StringTokenizer(in, "=:");
        while (st.hasMoreTokens()) {
            String key = st.nextToken();
            String val = st.nextToken();
            System.out.println(key + "........." + val);
        }
    } }
```

9.2 Класс Runtime

Предоставляет доступ к контексту исполнения данной программы в рамках текущей виртуальной машины.

getRuntime()	Возвращает текущий класс среды исполнения.
exit(int code)	Выход виртуальной машины.
load(String libFileName)	Загружает системную библиотеку по имени файла.
loadLibrary(String lib)	Загружает системную библиотеку по имени библиотеки.
addShutdownHook(), removeShutdownHook()	Установка действий по завершению работы JVM, снятие действий по завершению работы JVM.
exec(String command)	Выполнение системной команды.
runFinalization()	Запрос на финализацию всех доступных объектов.
freeMemory(), maxMemory()	Количество свободной памяти внутри JVM, количество максимально доступной для JVM.

```
class ExecDemo {
    public static void main(String args[]) {
        Process process = null;
        String cmd[] = { "notepad", "c:\\autoexec.bat" };
        try {
            process = Runtime.getRuntime().exec(cmd);
        } catch (Exception e) {
            System.out.println("Error executing " +e + cmd[0]);
        }
    }
}
```

9.3 Класс System

Представляет собой точку доступа к системным ресурсам компьютера.

System.setOut, System.setIn, System.setErr	Перенаправление потоков вывода, ввода и вывода ошибок.
arraycopy(Object, int, Object, int, int)	Копирование массива.
currentTimeMillis()	Текущее время в миллисекундах.
static String getProperty(String name)	Возвращает системное свойство по его имени.
setProperty(*)	Устанавливает системное свойство.
getSecurityManger()	Возвращает менеджер безопасности, если установлен.
setSecurityManger(*)	Устанавливает менеджер безопасности.
static string getEnv(String name)	Возвращает свойство окружения по его имени.
gc()	Запрашивает сборку мусора.

getProperty(String name) getProperties()	Методы, предназначенные для работы с системными свойствами. Примеры системных свойств: • java.class.path • java.runtime.name • user.dir • java.version • sun.boot.class.path
static String getenv(String name)	Метод, предназначенный для чтения переменных окружения. Примеры переменных окружения : • HOMEDRIVE=C: • PROCESSOR_REVISION=0801 • USERDOMAIN=CMSDOMAIN • PATH=C:\DOCUME…
loadLibrary()	Загружает системную библиотеку.

9.4 Класс java.util.Calendar

Предоставляет возможность исчисления дат в различных календарях.

Следующий пример показывает какие действия можно выполнять над датами с помощью класса Calendar.

```
// создание объекта
 Calendar cal = Calendar.getInstance();
 // текущий день в году
 int today = cal.get ( Calendar.DAY_OF_YEAR);
 // смена даты на вчерашний день
 cal.add ( Calendar.DAY_OF_YEAR, -1 ) ;
 // вычисление номера вчерашнего дня
 int yesterday = cal.get ( Calendar.DAY_OF_YEAR ) ;
 // перевод времени на неделю назад
 cal.add ( Calendar.DAY_OF_YEAR, -6 ) ;
 int lastWeek = cal.get ( Calendar.DAY_OF_YEAR);
 //количество дней в году
 cal.getActualMaximum ( Calendar.DAY_OF_MONTH )  ) ;
```

9.5 Классы-оболочки для примитивных типов (java.lang.Integer)

(Wrappers) – это классы, находящиеся в пакете java.lang, включающие в себя соответствующее поле примитивного типа. Существуют следующие классы-оболочки: Integer, Long, Short, Byte, Double, Float, Boolean, Character. Каждый из этих классов предоставляет интерфейс к значению определенного типа. Интерфейс численных типов схож со следующим (класс Integer).

MAX_VALUE, MIN_VALUE	Максимально и минимальное допустимые значения.
public static Integer valueOf(int i)	Метод-фабрика для создания класса из примитивного значения
public static int parseInt(String s)	Метод для преобразования строки в примитивный тип.
public static Integer valueOf(String s) throws NumberFormatException	Метод-фабрика для создания класса из строки.

public int intValue()	Возврат примитивного значения данного класса.
public static Integer getInteger(String propertyName)	Считывание системного свойства, задаваемого строкой, и преобразование в целое число.
public static int signum(int i)	Знак переданного числа.
public long longValue()	Возврат примитивного значения данного класса в виде long.
public static String toBinaryString(int i)	Создание битового представления (в виде строки) переданного
public static String toHexString(int i)	Создание шестнадцатиричного представления (в виде строки) переданного числа.

Начиная с JDK5.0 возможно неявное преобразование примитивного типа в его класс-оболочку и наоборот.

9.6 Класс Property

Является хранилищем пар ключ-значение, привязанных к системным свойствам.

```
import java.util.Properties;
class PropDemo {
   public static void main(String args[]) {
      Properties sysProperties = System.getProperties();
      System.out.println(sysProperties.getProperty("os.name"));
      System.out.println(sysProperties.getProperty("os.arch"));
      System.out.println(sysProperties.getProperty("java.class.path"));
   }
}
```

Дополнительные источники:

- http://java.sun.com/javase/6/docs/api/

10 Графический пользовательский интерфейс

10.1 Простой пользовательский интерфейс на базе аплета

Набор JDK предлагает набор классов и программных интерфейсов для построения графических интерфейсов, независимых от платформы.

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

public class SimpleButton extends JFrame {
   private JButton jb = new JButton("Press Me");
   public SimpleButton(){
         setLayout(new FlowLayout());
         getContentPane().add(jb);
         setSize(200, 200);
         setVisible(true);
   }
   public static void main(String s[]){
         new SimpleButton();
   }
}
```

10.2 Интерфейс на основе JFrame

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

public class FrameWithLabel {
   public static void main(String []args){
      JButton jb = new JButton("Press Me");
      JLabel jl = new JLabel("A Mark");
      JFrame jFrame = new JFrame("Simple JFrame");
         jFrame.setLayout(new FlowLayout());
      jFrame.setSize(200, 200);
      jFrame. add(jb);
      jFrame. add(jl);
      jFrame.setVisible(true);
   }
}
```

10.3 Модель событий

Каждое действие пользователя транслируется в объекты языка Java, в так называемые сообщения (events). Обработка сообщений происходит в методах классов-слушателей(listeners).

Пример реализует реакцию на нажатие кнопки.

```
public class SimpleButton extends JFrame{
    JButton jb1 = new JButton("Button #1");
    JButton jb2 = new JButton("Button #2");
    JTextField jText = new JTextField("Nothing here");

    class ButtonListener implements ActionListener{
        public void actionPerformed(ActionEvent e){
            String s = ((JButton)e.getSource()).getText();
            jText.setText(s);
        }
    }
    public SimpleButton(){
        Container cp = getContentPane();
        cp.setLayout(new FlowLayout());
        jb1.addActionListener(new ButtonListener());
        jb2.addActionListener(new ButtonListener());
        cp.add(jText);
        cp.add(jb1);
        cp.add(jb2);
    }
    public static void main(String []args){
        JFrame jFrame = new SimpleButton();
        jFrame.setSize(200, 200);
        jFrame.setVisible(true);
    }
}
```

TODO: добавить туториал по созданию интерфейса в матисе

Доп. информация:

- **http://www.netbeans.org/kb/articles/matisse.html**
- **http://java.sun.com/docs/books/tutorial/uiswing/components/index.html**

11 Коллекции объектов

Возможность использовать уже готовые структуры данных упрощает разработку приложений, ускоряет работу программы и ее надежность. Улучшается взаимодействие между несвязными программными интерфейсами. Коллекции проще изучить ввиду общности интерфейса. Также увеличиваются шансы на переиспользование кода.

11.1 Список

```
List duckFamily = new ArrayList();
//добавляем 10 уток в список
for (int i = 0; i < 10; i++){
       duckFamily.add(new Duck());
}
for (int i = 0; i < duckFamily.size(); i++){
 System.out.println((Duck)duckFamily.get(i));
} //достаем всех уток из списка
//но если в нем случайно оказалась кошка...
duckFamily.add(new Cat());
//получим ошибку приведения типа!
```

11.2 Множество

```
Set animalTypes = new HashSet();
 animalTypes.add("cat");
 animalTypes.add("pig");
 animalTypes.add("cat");
 Iterator it = animalTypes.iterator();
 while (it.hasNext()){
        System.out.print("" + it.next()+ " : ");
 }
```

11.3 Ассоциативный массив

```
class HashDemo {
    public static void main(String args[]) {
        Hashtable ht = new Hashtable();
        ht.put("title", "Java in a nutshell");
        ht.put("author", "Alex Lesley");
        ht.put("email", "lesley@xxx.com");
        ht.put("age", new Integer(1999));
        show(ht);
    }
    static void show(Dictionary d) {
        System.out.println("Title: " + d.get("title"));
        System.out.println("Author: " + d.get("author"));
        System.out.println("Email: " + d.get("email"));
        System.out.println("Age: " + d.get("age"));
    } }
```

11.4 Stack

```
public class StackDemo {
    static void showpush(Stack st, int a) {
        st.push(new Integer(a));
        System.out.println("push(" + a + ")");
        System.out.println("stack: " + st);
    }
    static void showpop(Stack st) {
        System.out.print("pop -> ");
        Integer a = (Integer) st.pop();
        System.out.println(a);
        System.out.println("stack: " + st);
    }
    public static void main(String args[]) {
        Stack st = new Stack();
        System.out.println("stack: " + st);
        showpush(st, 42);
        showpush(st, 66);
        showpush(st, 99);
        showpop(st);
        showpop(st);
        showpop(st);
        try {
            showpop(st);
        } catch (EmptyStackException e) {
            System.out.println("empty stack");
        }
    }}
```

11.5 Vector

```
public class VectorDemo{
    public static void main(String[] args) {
        Vector comps = new Vector();
        comps.addElement("1");
        comps.addElement("2");
```

```
        comps.add(0, "3");
        comps.insertElementAt("4", 1);
        System.out.println("Capacity is: "+comps.capacity());
        Iterator iter = comps.iterator();
        while (iter.hasNext()){
            System.out.println(iter.next());
        }
    }}
```

Дополнительные источники:

- http://java.sun.com/docs/books/tutorial/collections/index.html
- John Zukowski “Java Collections“

12 Потоки данных

12.1 Содержимое каталога файловой системы

```
import java.io.*;
class ListFiles {
    public static void main(String[] args) {
        File aFile = new File("d:\\directory1\\directory2");
        for (int i = 0; i<aFile.list().length;i++){
            System.out.println(">>"+aFile.list()[i]);
        }
    }
}
```

12.2 Копирование файла

```
import java.io.*;
class CopyFile {
    public static void main(String[] args) {
        try {
            File inFile = new File("infile.dat");
            File outFile = new File("outfile.dat");
            FileInputStream inStream = new
FileInputStream(inFile);
            FileOutputStream outStream = new
FileOutputStream(outFile);
            int c;
            while ((c = inStream.read()) != -1) {
outStream.write(c); }
            inStream.close();
            outStream.close();
        } catch (FileNotFoundException ex) {
            System.out.println("File not found "+ex);
        } catch (IOException ex) {
            System.out.println("IO error  "+ex);
         }
      }
}
```

12.3 Ввод с клавиатуры

TODO добавить как осуществить ввод данных в среде: ввод через параметры ком. Строки и через in.read().

```
import java.io.*;

public class SystemIn{
    public static void main(String []args) throws IOException{
        InputStreamReader helpReader = new
                          InputStreamReader(System.in);
        BufferedReader in = new BufferedReader(helpReader);
        String s;
        while ((s=in.readLine()).length() != 0){
            System.out.println(">>"+s);
        }
    }
}
```

13 Пакет java.net

13.1 Интернет-адрес

```
import java.net.*;
//InetAddress SW[]=InetAddress.getAllByName("www.starwave.com");

public class GetIp{
    public static void main(String args[]){
        InetAddress address=null;
        InetAddress localAddress = null;

        try{
           address=InetAddress.getByName(args[0]);
           localAddress=InetAddress.getLocalHost();
        } catch(UnknownHostException e){}
           System.out.println("host: "+address);
           System.out.println("localhost: "+address);
        }
}
```

13.2 Составные части адреса

```
import java.net.URL;
class myURL {
    public static void main(String args[]) throws Exception {
        URL hp = new URL("http://java.sun.com");
        System.out.println("Protocol: " + hp.getProtocol());
        System.out.printin("Port: " + hp.getPort());
        System.out.println("Host: " + hp.getHost());
        System.out.println("Ext: " + hp.toExternalForm());
    } }
```

13.3 Пример соединения с удаленным сервером

```
import java.net.*;
import java.io.*;
class TimeHost {
    public static void main(String args[]) throws Exception {
        int c;
        Socket s = new Socket("timehost.starwave.com",880);
        InputStream in = s.getInputStream();
        while ((c = in.read()) != -1) {
            System.out.print( (char) c);
        }
        s.close(); }
}
```

13.4 Серверный класс

Два следующих класса реализуют сетевое взаимодействие посредством сокетов.

```
class MyServer implements Runnable{
    public void run(){
        ServerSocket sv = null;
        Socket svSocket = null;
        OutputStream os = null;
        try{
            sv=new ServerSocket(77,16);
            svSocket = sv.accept();
            System.out.println("Server ready!");
            os = svSocket.getOutputStream();
            os.write(55);
        }catch(IOException e){}
    }
}
```

13.5 Клиентский класс

```
class MyClient implements Runnable{
    public void run(){
        Socket clSocket = null;
        InputStream is = null;
        try{
            clSocket = new
Socket(InetAddress.getLocalHost(),77);
            is = clSocket.getInputStream();
            System.out.println("Cliend reads:"+is.read());
        }catch(UnknownHostException e){
            System.out.println("Unknown host  "+e);
        }catch(IOException e){
            System.out.println("IO error  "+e);
        }
    }
}
```

```
class RunClientServer {
    public static void main(String args[]){
     new Thread(new MyServer()).start();
     new Thread(new MyClient()).start();
    }
}
```

13.6 Содержимое web странички

```
import java.net.*;
import java.io.*;
class localURL {
    public static void main(String args[]) throws Exception {
        int c;
        URL url = new URL("http", "url address", 80, "/");
        URLConnection urlCon = url.openConnection();
        if (urlCon.getContentLength() > 0) {
            System.out.println("=== Content ===");
            InputStream input = urlCon.getInputStream();
            int i= urlCon.getContentLength();
            while (((c = input. read()) != -1) && (--i > 0)) {
                System.out.print((char) c);
            }
            input.close();
        } else {
            System.out.println("No Content Available");
        }
     }
}
```

14 Список рекомендуемой литературы

- П. Ноутон, Г. Шилдт, Java2
- Кей С. Хорстманн, Гари Корнелл “Java 2. Том I. Основы”

- Дж. Гослинг, К. Арнольд. «Язык программирования Java»
- Б. Эккель «Философия Java»
- С. Симкин, А. Лесли, Н. Бартлет «Путеводитель. Программирование на Java»