МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра ЮНЕСКО по новым информационным технологиям

С.Н. Карабцев

ПРОГРАММИРОВАНИЕ НА JAVA

Лабораторный практикум

Математический факультет

специальность 010300.62 – фундаментальная информатика и информационные технологии;
специальность 010400.62 – прикладная математика и информатика;
специальность 010500.62 – математическое обеспечение и администрирование информационных систем.

Кемерово, 2012

ЛАБОРАТОРНАЯ РАБОТА №1.
СОЗДАНИЕ ПРОГРАММ НА ЯЗЫКЕ JAVA

1. Цель работы

Получить общее представление о создании программ на языке Java и познакомиться с его основными понятиями. Изучить синтаксические единицы, основные операторы и структуру кода программы. Освоить способы компиляции исходного кода и запуска программы.

2. Методические указания

Лабораторная работа направлена на приобретение навыка написания программ на языке Java, а также умения выполнять компиляцию и запуск программы как из среды разработки (Eclipse, http://eclipse.org), так и из командной строки.

Требования к результатам выполнения лабораторного практикума:

- при выполнении задания необходимо сопровождать все реализованные процедуры и функции набором тестовых входных и выходных данных и описаниями к ним;
- компиляцию, запуск программ выполнять различными способами;
- по завершении выполнения задания составить отчет о проделанной работе.

При составлении и оформлении отчета следует придерживаться рекомендаций, представленных на следующей странице: http://unesco.kemsu.ru/student/rule/rule.html.

3. Теоретический материал

Язык Java ворвался в Интернет в конце 1995 года и немедленно завоевал популярность. Он обещал стать универсальным средством, обеспечивающим

связь пользователей с любыми источниками информации, независимо от того, где она расположена – на Web-сервере, в базе данных, хранилище данных и т.д. Этот хорошо разработанный объектно-ориентированный язык программирования поддержали все производители программного обеспечения. Он имеет встроенные средства, позволяющие решать задачи повышенной сложности такие как: работа с сетевыми ресурсами, управление базами данных, динамическое наполнение web-страниц, многопоточность приложений.

Инсталляция набора инструментальных средств Java Software Development Kit

JDK долгое время был базовым средством разработки приложений. Он не содержит никаких текстовых редакторов, а оперирует только с уже существующими java-файлами. Компилятор представлен утилитой javac (java compiler), виртуальная машина реализована программой java. Для тестовых запусков аплетов есть специальная утилита appletviewer.

Пакет Java Software Development Kit можно загрузить с web-страницы http://java.sun.com/javase/downloads/index.jsp. Способы инсталляции на разных платформах (Solaris, Windows, Linux) отличаются друг от друга. После инсталляции пакета JSDK нужно добавить имя каталога jdk\bin в список путей, по которым операционная система может найти выполняемые файлы. Правильность установки пакета можно проверить, набрав команду **java –version.** На экране должно появиться, примерно, следующее:

```
java version "1.5.0_01"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_01-b08)
Java HotSpot(TM) Client VM (build 1.5.0_01-b08, mixed mode, sharing)
```

Среда разработки программ

Для написания программ на языке Java достаточно использовать самый простой текстовый редактор, однако применение специализированных

средств разработки (Eclipse, Java WorkShop, Java Studio и др.) предоставляет большой набор полезных и удобных функций. Существует несколько способов компиляции и запуска на выполнение программ, написанных на языке Java: из командной строки или из другой программы, например, интегрированной среды разработки. Для компиляции программы из командной строки необходимо вызвать компилятор, набрав команду **javac** и указав через пробел имена компилируемых файлов:

```
javac file1.java file2.java file3.java
```

При успешном выполнении этапа компиляции в директории с исходными кодами появятся файлы с расширением .class, которые являются java байт-кодом. Виртуальная Java-машина (JVM) интерпретирует байт-код и выполняет программу. Для запуска программы необходимо в JVM загрузить основной класс, т.е. класс, который содержит функцию **main(String s[])**. Например, если в файле file1.java есть функция main(), которая располагается в классе file1, то для запуска программы после этапа компиляции необходимо набрать следующее:

```
java file1
```

Компиляцию и запуск программ из интегрированных сред разработки необходимо осуществлять в соответствии с документацией на программный продукт. Дополнительную информацию по инструментальным средствам JDK смотри в приложении 1.

Анализ программы

Технология Java, как платформа, изначально спроектированная для Глобальной сети Internet, должна быть многоязыковой, а значит, обычный набор символов ASCII (American Standard Code for Information Interchange, Американский стандартный код обмена информацией), включающий в себя лишь латинский алфавит, цифры и простейшие специальные знаки (скобки, знаки препинания, арифметические операции и т.д.), недостаточен. Поэтому для записи текста программы применяется более универсальная кодировка

Unicode. Например, если в программу нужно вставить знак с кодом 6917, необходимо его представить в шестнадцатеричном формате (1B05) и записать: **\u1B05**.

Компилятор, анализируя программу, сразу разделяет ее на:

- пробелы (white spaces);
- комментарии (comments);
- основные лексемы (tokens).

Пробелами в данном случае называют все символы, разбивающие текст программы на лексемы. Это как сам символ пробела (space, \u0020, десятичный код 32), так и знаки табуляции и перевода строки. Они используются для разделения лексем, а также для оформления кода, чтобы его было легче читать. Например, следующую часть программы (вычисление корней квадратного уравнения):

```
double a = 1, b = 1, c = 6;
double D = b * b - 4 * a * c;

if (D >= 0) {
  double x1 = (-b + Math.sqrt (D)) / (2 * a);
  double x2 = (-b - Math.sqrt (D)) / (2 * a);
}
```

можно записать и в таком виде:

```
double a=1,b=1,c=6;double D=b*b-4*a*c;if(D>=0)
{double x1=(-b+Math.sqrt(D))/(2*a);double
x2=(-b-Math.sqrt(D))/(2*a);}
```

В обоих случаях компилятор сгенерирует абсолютно одинаковый код. Единственное соображение, которым должен руководствоваться разработчик, - легкость чтения и дальнейшей поддержки такого кода.

Комментарии не влияют на результирующий бинарный код и используются только для ввода пояснений к программе. В Java комментарии бывают двух видов: строчные и блочные. Строчные комментарии

начинаются с ASCII-символов // и длятся до конца текущей строки, например:

int y=1970; // год рождения

Блочные комментарии располагаются между ASCII-символами /* и */, могут занимать произвольное количество строк. Кроме этого, существует особый вид блочного комментария – комментарий разработчика (/** комментарии*/). Он применяется для автоматического создания документации кода [1].

Лексика языка

Лексика описывает, из чего состоит текст программы, каким образом он записывается и на какие простейшие слова (лексемы) компилятор разбивает программу при анализе. Лексемы (или tokens в английском варианте) – это основные "кирпичики", из которых строится любая программа на языке Java [1]. Ниже перечислены все виды лексем в Java:

- идентификаторы (identifiers);
- ключевые слова (key words);
- литералы (literals);
- разделители (separators);
- операторы (operators).

Идентификаторы - это имена, которые даются различным элементам языка для упрощения доступа к ним. Имена имеют пакеты, классы, интерфейсы, поля, методы, аргументы и локальные переменные. Длина имени не ограничена. Идентификатор состоит из букв и цифр. Имя не может начинаться с цифры

Ключевые слова – специальные лексемы, зарегистрированные в системе для внутреннего использования, такие как abstract, default, if, private, this, boolean, implements, protected, static, try, void, native и др.

Литералы позволяют задать в программе значения для числовых, символьных и строковых выражений, а также null-литералов. В Java определены следующие виды литералов:

- целочисленный (integer);
- дробный (floating-point);
- булевский (boolean);
- символьный (character);
- строковый (string);
- null-литерал (null-literal).

Целочисленные (тип int занимает 4 байта, тип long – 8) литералы позволяют задавать целочисленные значения в десятеричном, восьмеричном и шестнадцатеричном виде. Запись нуля можно осуществить следующими способами:

- 0 (10-ричная система)
- 00 (8-ричная)
- 0x0 (16-ричная)

Если в конце литерала не стоит указателя на тип, то литерал по умолчанию имеет тип int.

Дробные литералы (тип float занимает 4 байта, тип double – 8) представляют собой числа с плавающей десятичной точкой. Дробный литерал состоит из следующих составных частей (по умолчанию имеет тип double):

- целая часть;
- десятичная точка (используется ASCII-символ точка);
- дробная часть;
- показатель степени (состоит из латинской ASCII-буквы «E» в произвольном регистре и целого числа с опциональным знаком «+» или «–»);
- окончание-указатель типа (D или F).

Символьные литералы. Представляют собой один символ и заключаются в одинарные кавычки ‘s’, ‘a’. Допускается запись через Unicode ‘\u0041’ – латинская буква “A”.

Строковые литералы состоят из набора символов и записываются в двойных кавычках: “символьный литерал”.

Null литерал может принимать всего одно значение: null. Это литерал ссылочного типа, причем эта ссылка никуда не ссылается.

Разделители – специальные символы, используемые в конструкциях языка “()”, “[]”, “{}”, “;”, “,”, “.”.

Операторы используются в различных операциях – арифметических, логических, битовых, операциях сравнения и присваивания.

Пример простой программы “Hello, world!” выглядит следующим образом:

```
public class Test {
/**
  * Основной метод, с которого начинается
  * выполнение любой Java программы.
  */
    public static void main (String args[])
    {
      System.out.println("Hello, world!");
    }
}
```

Типы данных

Java является строго типизированным языком. Это означает, что любая переменная и любое выражение имеют известный тип еще на момент компиляции. Такое строгое правило позволяет выявлять многие ошибки уже во время компиляции. Компилятор, найдя ошибку, указывает точную строку и причину ее возникновения, а динамические "баги" необходимо сначала выявить тестированием, а затем найти место в коде, которое их породило. Все типы данных разделяются на две группы. Первую составляют 8 простых или примитивных (от английского primitive) типов данных [1-3]. Они подразделяются на три подгруппы:

– целочисленные: byte, short, int, long, char;

– дробные: float, double;

– булевский: boolean.

Булевский тип представлен всего одним типом boolean, который может хранить всего два возможных значения - true и false. Величины именно этого типа получаются в результате операций сравнения.

Вторую группу составляют объектные или ссылочные (от английского reference) типы данных. Это все классы, интерфейсы и массивы.

Переменные

Переменные используются в программе для хранения данных. Любая переменная имеет три базовых характеристики: имя, тип, значение. Имя уникально идентифицирует переменную и позволяет к ней обращаться в программе. Тип описывает, какие величины может хранить переменная. Значение - текущая величина, хранящаяся в переменной на данный момент. Значение может быть указано сразу (инициализация), а в большинстве случаев задание начальной величины можно и отложить.

```
int a;
int b=0, c=2+3;
double d=e=5.5;
```

Ниже в таблице сведены данные по всем целым и дробным типам:

Название типа	Длина (байт)	Область значений
Целые типы		
byte	1	-128..127
short	2	-32768..32767
int	4	-2147483648..2147483647
long	8	-9223372036854775808.. 9223372036854775807
char	2	0..65535
Дробные типы		
float	4	3.40282347e+38f; 1.40239846e-45f
double	8	1.79769313486231570e+308; 4.94065645841246544e-324

Поток управления

В языке Java, как и в любом другом языке программирования, есть условные операторы и циклы для управления потоком. Блок, или составной оператор, произвольное количество простых операторов языка Java, заключенных в фигурные скобки. Блоки определяют область видимости своих переменных. Блоки могут быть вложенными один в другой. Однако невозможно объявить одинаково названные переменные в двух вложенных блоках.

```
public static void main(String [] args)
{  int n;
   …
   {
    int k;
    int n; // Ошибка – невозможно переопределить переменную n во вложенном цикле
   }       //переменная k определена только в этом блоке
}
```

Условные операторы

Условный оператор в языке Java имеет вид:

```
if (условие) оператор
// или
if (условие) {
оператор1;
оператор2;  }
```

Все операторы, заключенные в фигурные скобки, будут выполнены, если значение условия истинно. Общий случай условного оператора выглядит так:

```
if  (условие) оператор1 else оператор2
if (yourSale >= target)
{ performance=”Удовлетворительно”;
  Bonus = 100 + 0.01*( yourSale - target);
}
else
{ performance=”Неудовлетворительно”;
```

```
  Bonus =0;
}
```

Многовариантное ветвление представлено в виде повторяющихся операторов if … else if…

```
if (sale >=2*target)
{ performance=”Отлично”;
}
else if (sale >=1.5*target)
{ performance=”Удовлетворительно”;
}
else {System.out.println(“Вы уволены”);}
```

Неопределенные циклы

Существует два вида повторяющихся циклов, которые лучше всего подходят, если вы точно не знаете, сколько повторений должно быть выполнено. Первый из них, цикл while, выполняет тело цикла, только пока выполняется его условие.

while (условие) {операторы;}

Условие цикла while проверяется в самом начале. Следовательно, возможна ситуация, когда код, содержащийся в блоке, не будет выполнен ни разу. Если необходимо, чтобы блок выполнялся хотя бы один раз, проверку условия нужно перенести в конец. Это можно сделать с помощью цикла do/while.

do оператор while (условие);

Определенные циклы

Цикл for – распространенная конструкция для выполнения повторений, количество которых контролируется счетчиком, обновляемым на каждой итерации.

```
for (int i = 1; i < = 10; i++){
System.out.println(i);
}
```

Первый элемент оператора for обычно выполняет инициализацию счетчика, второй формулирует условие выполнения тела цикла, а третий определяет способ обновления счетчика.

```
for (int i = 10; i > 0; - i){
System.out.println("Обратный отсчет …" + i);
}
```

Многовариантное ветвление – оператор switch

Конструкция if/else может оказаться неудобной, если необходимо сделать выбор из многих вариантов. Например, создавая систему меню из трех альтернатив, можно использовать следующий код.

```
String input = JOptionPane.showInputDialog ("Выберите вариант (1, 2, 3)");
int choice = Integer.parseInt (input);
switch (choice){
case 1:
  …
  break;
case 2:
  …
  break;
case 3:
  …
  break;
default: // неверный выбор
 …
 break; }
```

Выполнение начинается с метки case, соответствующей значению переменной choice, и продолжается до следующего оператора break или конца оператора switch. Если ни одна метка не совпадает со значением переменной, выполняется раздел default. Метка case должна быть целочисленной!

Прерывание потока управления

Для выхода из цикла можно применять тот же оператор, что использовался для выхода из тела оператора switch.

```
while (balance <=100){
 balance += payments;
```

```
  if (balance .= goal) break; // выход из цикла
}
```

Пакеты

Программа на Java представляет собой набор пакетов (***packages***). Каждый пакет может включать вложенные пакеты, а так же может содержать классы и интерфейсы. Каждый пакет имеет свое пространство имен, что позволяет создавать одноименные классы в различных пакетах.

Имена бывают простыми (simple), состоящими из одного идентификатора, и составными (qualified), состоящими из последовательности идентификаторов, разделенных точкой. Составное имя любого элемента пакета составляется из составного имени этого пакета и простого имени элемента.

Простейшим способом организации пакетов и типов является обычная файловая структура. Например, исходный код класса

space.sunsystem.Moon хранится в файле ***space\sunsystem\Moon.java***

Запуск программы на JAVA стоит производить из директории, в которой содержатся пакеты. Было бы ошибкой запускать Java прямо из папки space\sunsystem и пытаться обращаться к классу Moon, несмотря на то, что файл-описание лежит именно в ней. Необходимо подняться на два уровня директорий выше, чтобы Java, построив путь из имени пакета, смогла обнаружить нужный файл.

Модуль компиляции

Модуль компиляции (complication unit)-хранится в текстовом .java-файле и является единичной порцией входных данных для компилятора. Состоит из трех частей:

- Объявление пакета;
- Import-выражения;
- Объявления верхнего уровня;

Объявление пакета указывает, какому пакету будут принадлежать все объявляемые ниже типы. Используется ключевое слово ***package***, после которого указывается полное имя пакета. Например, в файле java/lang/Object.java идет: package java.lang; что служит одновременно объявлением пакета lang, вложенного в пакет java, и указанием, что объявляемый ниже класс Object, находится в этом пакете. Так складывается полное имя класса java.lang.Object.

Область видимости типа - пакет, в котором он располагается. Внутри этого пакета допускается обращение к типу по его простому имени. Из всех других пакетов необходимо обращаться по составному имени.

Для решения этой проблемы вводятся ***import***-выражения, позволяющие импортировать типы в модуль компиляции и далее обращаться к ним по простым именам. Существует два вида таких выражений:

- импорт одного типа: import java.net.URL;
- импорт пакета: import java.awt.*;

4. Порядок выполнения работы

- изучить предлагаемый теоретический материал;
- создайте следующие программы:

1. Программа выдает на экран все аргументы, переданные ей через командную строку. Основной класс программы (с функцией main()) находится в пакете **test.first**. Привести команды компиляции и запуска программы с точным указанием места на жестком диске, откуда выполнялись эти команды.
2. Программа, в которой перебираются числа от 1 до 500 и выводятся на экран. Если число делится на 5, то вместо него выводится слово fizz, если на 7, то buzz. Если число делится на 5 и на 7, то выводить

слово fizzbuzz. Примечание*: остаток от деления в Java обозначается через символ %.

3. Программа, в которой все переданные во входную строку аргументы выводятся на экран в обратной порядке. Например, если было передано 2 аргумента – make install, то на экран должно вывестись llatsni ekam. Примечание*: для разбора слова по буквам необходимо использовать функцию charAt(). Например, str.charAt(i) вернет символ с позиции i в слове, записанном в строковую переменную str. Команда str.length() возвращает длину слова str.
4. Создайте программу, вычисляющую числа Фибоначчи. Числа Фибоначчи – последовательность чисел, в котором каждое следующее число равно сумме двух предыдущих. Начало этой последовательности – числа 1, 1, 2, 3, 5, 8, 13…
5. Создайте программу, вычисляющую факториал целого числа.
6. Создайте программу, вычисляющую простые числа с применением алгоритма «Решето Эратосфена». Этот алгоритм находит простые числа путем исключения всех чисел, кратных меньшим простым.

5. Содержание отчета

В отчете следует указать:

1. цель работы;
2. введение;
3. программно-аппаратные средства, используемые при выполнении работы;
4. основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения лабораторного практикума;
5. заключение (описание результатов и выводы);
6. список используемой литературы.

6. Литература

1. Вязовик, Н.А. Программирование на Java. [электронный ресурс] http://www.intuit.ru/department/pl/javapl (7.01.2012).
2. Хорстманн К.С., Корнелл Г. Библиотека профессионала. JAVA 2. Том 1. Основы. 8-е издание. Пер. с англ. – М.: ООО Издательский дом “Вильямс”, 2008. – 816 с.
3. Эккель Б. Философия JAVA. 4-е издание. – СПб.: Питер, 2009. – 638с.
4. Информационные материалы с официального сайт а разработчиков http://www.oracle.com/technetwork/java/javase/downloads/index.html [электронный ресурс] (7.01.2012).

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра ЮНЕСКО по новым информационным технологиям

С.Н. Карабцев

ПРОГРАММИРОВАНИЕ НА JAVA

Лабораторный практикум

Математический факультет

специальность 010300.62 – фундаментальная информатика и информационные технологии;
специальность 010400.62 – прикладная математика и информатика;
специальность 010500.62 – математическое обеспечение и администрирование информационных систем.

Кемерово, 2012

ЛАБОРАТОРНАЯ РАБОТА №2.
СОЗДАНИЕ СОБСТВЕННЫХ КЛАССОВ

1. Цель работы

Получить основные понятия по следующим разделам языка Java:

- объектно-ориентированное программирование;
- создание объектов и классов из стандартной библиотеки Java;
- создание собственных классов.

2. Методические указания

Лабораторная работа направлена на приобретение навыков конструирования и реализации объектов на языке Java. В практическом задании большое внимание уделяется созданию в классах методов, а также их замещению.

Требования к результатам выполнения лабораторного практикума:

- при выполнении задания необходимо сопровождать все реализованные процедуры и функции набором тестовых входных и выходных данных и описаниями к ним;
- реализацию программы можно осуществлять как из интегрированной среды разработки Eclipse, так и в блокноте с применением вызовов функций командной строки;
- по завершении выполнения задания составить отчет о проделанной работе.

При составлении и оформлении отчета следует придерживаться рекомендаций, представленных на следующей странице: http://unesco.kemsu.ru/student/rule/rule.html.

3. Теоретический материал

Объектно-ориентированное программирование (ООП) в настоящее время стало доминирующей парадигмой программирования, вытеснив «структурные», процедурно-ориентированные подходы, разработанные в 1970 годах. Java представляет собой полностью объектно-ориентированный язык.

Объектно-ориентированная программа состоит из объектов. Каждый объект имеет определенную функциональность, которую предоставляет в распоряжение пользователей, а также скрытую реализацию. Многие объекты программ могут быть взяты программистами в готовом виде из стандартных пакетов Java, а некоторые написаны самостоятельно.

Классы и объекты

Класс – это шаблон, или проект, по которому будет сделан объект. При разработке собственных классов необходимо пользоваться абстрагированием от совокупности свойств реальных предметов, и выбирать только те характеристики и свойства предмета, которые удовлетворяют семантике решаемой задачи. Например, если разрабатывается информационная система, которая отвечает за распечатывание квитков при получении сотрудниками заработной платы, то для описания сотрудника в такой системе может быть достаточно указать ФИО сотрудника, размер и дату выдачи заработной платы. Если же речь идет о информационной системе по регистрации и обслуживанию больных в поликлинике, то значения размера и даты выдачи заработной платы больных не имеют никакого значения, а вот информация о месте их проживания, дате рождения и истории болезни играют существенную роль.

Объявление класса на языке Java может быть сделано следующим образом:

```
модификатор class class_name [extends parent_class]{//тело класса

//объявление полей класса
  тип  имя_поля; //свойства (поля) класса class_name
```

```
//объявление конструктора класса
  модификатор class_name(аргументы){тело конструктора};

//объявление методов класса
  модификатор тип method_name (аргументы){тело метода};
}
```

Например, класс сотрудник Employee можно описать следующим образом:

```
public class Employee{
//перечисление полей класса
  private String name;   // имя
  private double salary; // размер заработной платы
  private Date hiredate; // дата приема на работу

//конструктор класса, задача которого – присвоение значений полям класса
  public Employee(String n, double s, int year, int month, int day){
    name=n;
    salary=s;
    hiredate=(new GregorianCalendar(year,month-1,day)).getTime();
  }

//методы класса
public String getName{ //возвращает имя сотрудника
 return name}

public double getSalary{ //возвращает размер заработной платы сотрудника
 return salary}

public Date getDate{ // возвращает дату приема на работу
 return hiredate}
}
```

Объект - это мыслимая или реальная сущность, обладающая характерным поведением, отличительными характеристиками и являющаяся важной в предметной области. Объектом является экземпляр класса, созданный путем вызова конструктора класса. Каждый объект обладает состоянием, поведением и уникальностью.

Состояние (state) - совокупный результат поведения объекта, одно из стабильных условий, в которых объект может существовать. В любой конкретный момент времени состояние объекта включает в себя перечень свойств объекта и текущие значения этих свойств.

Поведение (behavior) - действия и реакции объекта, выраженные в терминах передачи сообщений и изменения состояния; видимая извне и воспроизводимая активность объекта.

Уникальность (identity) - природа объекта; то, что отличает его от других объектов.

Объекты и объектные переменные

Чтобы работать с объектами, их нужно сначала создать и задать исходное состояние. Затем к этим объектам применяются методы. В языке Java для создания новых экземпляров используются конструкторы. ***Конструктор*** – специальный метод, предназначенный для создания и инициализации экземпляра класса. Имя конструктора всегда совпадает с именем класса. Следовательно, конструктор класса Employee называется Employee и объявляется как

```
public Employee(String n, double s, int year, int month, int day){
  name=n;
  salary=s;
  hiredate=(new GregorianCalendar(year,month-1,day)).getTime();
}
```

В одном классе может быть объявлено несколько конструкторов, если их сигнатуры разные. Например, можно создать конструктор в классе Employee, который принимает только имя сотрудника и устанавливает ему заработную плату 1 условная единица. Дата выхода на работу всем таким сотрудникам будет установлена 31 декабря 2009 года.

```
public Employee(String n){
   name=n;
   salary=1;
```

```
  hiredate=(new GregorianCalendar(2009,12,31)).getTime();
 }
```

Для создания объекта необходимо объявить объектную переменную, затем вызвать конструктор класса. Например, объявим две переменные e1 и e2 с типом Employee. Создадим двух сотрудников в информационной системе с помощью вызова различных конструкторов.

```
Employee e1 = new Employee(“James Bond”, 100000, 1950,1,1);
Employee e2 = new Employee(“James NeBond”);
```

В результате вызова конструкторов сотрудник James Bond был принят на работу 1 января 1950 года с заработной платой 100000 у.е. (ссылка на объект сохранена в объектной переменной e1), сотрудник James NeBond был принят на работу 31 декабря 2009 года с заработной платой 1 у.е. (ссылка на объект сохранена в объектной переменной e2), т.к. для его создания использовался второй конструктор, который принимает одно-единственное значение.

Основные понятия ООП – инкапсуляция, наследование и полиморфизм

Наследование (inheritance) - это отношение между классами, при котором класс использует структуру или поведение другого (одиночное наследование) или других (множественное наследование) классов. Наследование вводит иерархию "общее/частное", в которой подкласс наследует от одного или нескольких более общих суперклассов. Подклассы обычно дополняют или переопределяют унаследованную структуру и поведение.

Расширим класс Employee следующим образом. Необходимо описать класс, экземпляры которого представляли бы менеджера предприятия. Менеджер является таким же сотрудником, однако у него есть дополнительное поле – премия. Соответственно, метод, который возвращал в

классе сотрудник Employee размер заработной платы, больше не подходит для менеджера.

```
class Manager extends Employee{ //наследование от класса сотрудник
private double bonus;  // размер премии

// конструктор класса
public Manager (String n, double s, int year, int month, int day){
 super(n, s, year, month, day); // т.к. класс унаследован от другого класса, то
                                // первой командой в конструкторе класса-
                                // потомка необходимо вызвать конструктор
                                // родителя. Т.к. в скобках после super указано 5
                                // аргументов, то будет вызван первый
                                // конструктор Employee
 }
}
```

Теперь каждый экземпляр класса Manager имеет 4 поля – name, salary, hiredate, bonus. Определяя подкласс, нужно указать лишь отличия между подклассом (потомком) и суперклассом (родителем). Разрабатывая классы, следует помещать методы общего назначения в суперкласс, а более специальные – в подкласс.

В приведенном выше примере не все методы родительского класса Employee подходят для класса Manager. В частности, метод getSalary() должен возвращать сумму базовой зарплаты и премии. Следовательно, нужно реализовать новый метод, ***замещающий*** (overriding) метод класса родителя. Сделать это можно следующим образом:

```
class Manager extends Employee{
….
public void getSalary() {новое тело метода….} // перекрытие (замещение)
                                               //метода класса родителя
….
}
```

Новый (замещенный) метод будет выглядеть так:

```
public void getSalary(){
double basesalary=super.getSalary();
```

```
return basesalary+bonus;
}
```

Инкапсуляция (encapsulation) - это сокрытие реализации класса и отделение его внутреннего представления от внешнего (интерфейса). При использовании объектно-ориентированного подхода не принято использовать прямой доступ к свойствам какого-либо класса из методов других классов. Для доступа к свойствам класса принято использовать специальные методы этого класса для получения и изменения его свойств.

Открытые члены класса составляют внешний интерфейс объекта. Эта та функциональность, которая доступна другим классам. Закрытыми обычно объявляются все свойства класса, а так же вспомогательные методы, которые являются деталями реализации и от которых не должны зависеть другие части системы. Благодаря сокрытию реализации за внешним интерфейсом класса можно менять внутреннюю логику отдельного класса, не меняя код остальных компонентов системы.

Полиморфизм (polymorphism) - положение теории типов, согласно которому имена (например, переменных) могут обозначать объекты разных (но имеющих общего родителя) классов. Следовательно, любой объект, обозначаемый полиморфным именем, может по-своему реагировать на некий общий набор операций.

Рекомендации по проектированию классов

- Всегда храните данные в переменных, объявленных как private
- Всегда инициализируйте данные
- Не используйте в классе слишком много простых типов
- Не для всех полей надо создавать методы доступа и модификации
- Используйте стандартную форму определения класса
- Разбивайте на части слишком большие классы
- Выбирайте для классов и методов осмысленные имена

4. Порядок выполнения работы

- изучить предлагаемый теоретический материал;
- реализуйте в виде программ на языке Java следующие задачи:

7. Создайте класс Rectangle, представляющий прямоугольник, экземпляры которого обладают четырьмя полями целого типа (x1, y1) (левый верхний угол), (x2, y2) (правый нижний угол). Для данного класса создать три конструктора, которые инициализируют поля следующим образом:
   - конструктор принимает 4 параметра целого типа и присваивает их значения полям (x1, y1), (x2, y2);
   - конструктор принимает 2 параметра целого типа – ширину и высоту прямоугольника, а левый верхний угол прямоугольника помещает в координату (0,0);
   - конструктор не принимает никаких параметров – создает вырожденный прямоугольник с координатами углов (0,0) и (0,0).

   В классе Rectangle создать метод ***rect_print()***, выдающий текущее состояние экземпляра прямоугольника (значение полей). Создать метод ***move (int dx, int dy)***, перемещающий прямоугольник по горизонтали на заданное dx, по вертикали на заданное dy. Создать метод ***Union***(подумать какие входные параметры), возвращающий объединение этого прямоугольника с другим прямоугольником (возвращается наименьший прямоугольник, содержащий оба прямоугольника).

   Для проверки работоспособности класса Rectangle создайте в отдельном файле класс Test, содержащий функцию main(…). Протестируйте в ней поведение экземпляров класса Rectangle

следующим образом: создайте три объекта Rectangle тремя различными созданными конструкторами, выведите состояние всех трех объектов. Воспользуйтесь вызовом функции move(…) с различными значениями параметров для каждого объекта и выведите новое положение созданных прямоугольников. Протестируйте работу функции Union на одном примере.

8. Расширьте класс Rectangle новым классом DrawableRect, у которого есть метод прорисовки draw(Graphics g) и поле outColor с типом данных Color (из пакета java.awt.*). Это поле служит для задания цвета границы прямоугольника. Для отображения прямоугольника в пакете java.awt.* существует специальный класс Graphics (его нужно импортировать в вашу программу с помощью import java.awt.*;). У экземпляров этого класса есть метод по установлению значения цвета рисуемого объекта (например, setColor(Color.red)) и метод рисования прямоугольника по 2 координатам (x,y), ширине w и высоте h drawRect(x,y,h,w) (рисует только границы прямоугольника, внутренность не закрашена).
9. Расширьте класс DrawableRect новым классом ColoredRect, в котором есть поле inColor с типом Color. Метод прорисовки draw(Graphics g) перекрывается следующим образом: прямоугольник отображается двумя цветами – граница цветом outColor, внутренность – inColor. Метод fillRect(x,y,h,w) (рисует закрашенный прямоугольник). Тестирование унаследованных от Rectangle классов DrawableRect и ColoredRect производить не нужно.

5. Содержание отчета

В отчете следует указать:

1. цель работы;

2. введение;
3. программно-аппаратные средства, используемые при выполнении работы;
4. основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения лабораторного практикума;
5. заключение (описание результатов и выводы);
6. список используемой литературы.

6. Литература

1. Вязовик, Н.А. Программирование на Java. [электронный ресурс] http://www.intuit.ru/department/pl/javapl (7.01.2012).
2. Хорстманн К.С., Корнелл Г. Библиотека профессионала. JAVA 2. Том 1. Основы. 8-е издание. Пер. с англ. – М.: ООО Издательский дом “Вильямс”, 2008. – 816 с.
3. Эккель Б. Философия JAVA. 4-е издание. – СПб.: Питер, 2009. – 638с.
4. Информационные материалы с официального сайт а разработчиков http://www.oracle.com/technetwork/java/javase/downloads/index.html [электронный ресурс] (7.01.2012).

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра ЮНЕСКО по новым информационным технологиям

С.Н. Карабцев

ПРОГРАММИРОВАНИЕ НА JAVA

Лабораторный практикум

Математический факультет

специальность 010300.62 – фундаментальная информатика и информационные технологии;
специальность 010400.62 – прикладная математика и информатика;
специальность 010500.62 – математическое обеспечение и администрирование информационных систем.

Кемерово, 2012

ЛАБОРАТОРНАЯ РАБОТА №3.
ГРАФИКА В JAVA. ПАКЕТ JAVA.AWT

1. Цель работы

В данной лабораторной работе вы получите первое представление о создании неконсольных приложений на языке java, а также научитесь использовать классы и методы пакета java.awt для отображения графических фигур на апплете.

2. Методические указания

Лабораторная работа направлена на приобретение навыка создания апплетов, их инициализации и запуска как из среды разработки (Eclipse, http://eclipse.org), так и с помощью инструмента разработчика appletviewer. Практические задания в данной лабораторной работе непосредственно связаны с результатами выполнения лабораторной работы №2. В данной теме вы получите знания и практический опыт использовании ранее разработанных классов в новых проектах, в частности при создании апплета.

Требования к результатам выполнения лабораторного практикума:

- при выполнении задания необходимо сопровождать все реализованные процедуры и функции набором тестовых входных и выходных данных и описаниями к ним;
- компиляцию, запуск программ выполнять различными способами;
- по завершении выполнения задания составить отчет о проделанной работе.

При составлении и оформлении отчета следует придерживаться рекомендаций, представленных на следующей странице: http://unesco.kemsu.ru/student/rule/rule.html.

3. Теоретический материал

Эта лабораторная работа начинает рассмотрение базовых библиотек Java, которые являются неотъемлемой частью языка и входят в его спецификацию [1], а именно описывается пакет java.awt, предоставляющий технологию AWT для создания графического (оконного) интерфейса пользователя – GUI. Ни одна современная программа, предназначенная для пользователя, не обходится без удобного, понятного, в идеале – красивого пользовательского интерфейса. С самой первой версии в Java существует специальная технология для создания GUI. Она называется AWT, Abstract Window Toolkit.

Поскольку Java-приложения предназначены для работы на разнообразных платформах, реализация графического пользовательского интерфейса (GUI) должна быть либо одинаковой для любой платформы, либо, напротив, программа должна иметь вид, типичный для данной операционной системы. В силу ряда причин, для основной библиотеки по созданию GUI был выбран второй подход [1]. Во-первых, это лишний раз показывало гибкость Java – действительно, пользователи разных платформ могли работать с одним и тем же Java-приложением, не меняя своих привычек. Во-вторых, такая реализация обеспечивала большую производительность, поскольку была основана на возможностях операционной системы. В частности, это означало и более компактный, простой, а значит, и более надежный код.

Библиотеку назвали AWT – Abstract Window Toolkit. Слово abstract в названии указывает, что все стандартные компоненты не являются самостоятельными, а работают в связке с соответствующими элементами операционной системы [1].

GUI всегда собирается из готовых строительных блоков, хранящихся в библиотеках. В Java их называют общим термином **компонент** (**component**), поскольку все они являются подклассами **java.awt.Component**.

Класс Component

Абстрактный класс **Component** является базовым для всех компонентов AWT и описывает их основные свойства. Визуальный компонент в AWT имеет прямоугольную форму, может быть отображен на экране и может взаимодействовать с пользователем [1].

Рассмотрим ***основные свойства*** этого класса.

Положение

Положение компонента описывается двумя целыми числами (тип int) x и y. В Java ось x проходит горизонтально, направлена вправо, а ось y – вертикально, но направлена вниз, а не вверх, как принято в математике.

Для описания положения компонента предназначен специальный класс – **Point** (точка). В этом классе определено два public int поля x и y, а также множество конструкторов и вспомогательных методов для работы с ними. Класс Point применяется во многих типах AWT, где надо задать точку на плоскости.

Для компонента эта точка задает положение левого верхнего угла.

Установить положение компонента можно с помощью метода **setLocation(),** который может принимать в качестве аргументов пару целых чисел, либо Point. Узнать текущее положение можно с помощью метода **getLocation()**, возвращающего Point, либо с помощью методов getX() и getY() [1].

Размер

Как было сказано, компонент AWT имеет прямоугольную форму, а потому его размер описывается также двумя целочисленными параметрами – **width** (ширина) и **height** (высота). Для описания размера существует специальный класс **Dimension** (размер), в котором определено два public int поля **width** и **height**, а также вспомогательные методы.

Установить размер компонента можно с помощью метода **setSize**, который может принимать в качестве аргументов пару целых чисел, либо **Dimension**. Узнать текущие размеры можно с помощью метода **getSize**(), возвращающего Dimension, либо с помощью методов **getWidth**() и **getHeight**().

Совместно положение и размер компонента задают его границы. Область, занимаемую компонентом, можно описать либо четырьмя числами (x, y, width, height), либо экземплярами классов Point и Dimension, либо специальным классом Rectangle (прямоугольник).

Задать границу объекта можно с помощью метода **setBounds**, который может принимать четыре числа, либо Rectangle. Узнать текущее значение можно с помощью метода **getBounds()**, возвращающего Rectangle.

Видимость

Существующий компонент может быть как виден пользователю, так и быть скрытым. Это свойство описывается булевским параметром visible. Методы для управления – **setVisible**, принимающий булевский параметр, и **isVisible**, возвращающий текущее значение.

Доступность

Даже если компонент отображается на экране и виден пользователю, он может не взаимодействовать с ним. В результате события от клавиатуры, или мыши не будут получаться и обрабатываться компонентом. Такой компонент называется **disabled**. Если же компонент активен, его называют **enabled**. Как правило, компонент некоторым образом меняет свой внешний вид, когда становится недоступным (например, становится серым, менее заметным), но, вообще говоря, это необязательно (хотя очень удобно для пользователя).

Для изменения этого свойства применяется метод **setEnabled**, принимающий булевский параметр (true соответствует enabled, false – disabled), а для получения текущего значения – **isEnabled**.

Цвета

Разумеется, для построения современного графического интерфейса пользователя необходима работа с цветами. Компонент обладает двумя свойствами, описывающими цвета, – **foreground** и **background** цвета. Первое свойство задает, каким цветом выводить надписи, рисовать линии и т.д. Второе – задает цвет фона, которым закрашивается вся область, занимаемая компонентом, перед тем, как прорисовывается внешний вид.

Для задания цвета в AWT используется специальный класс **Color**. Этот класс обладает довольно обширной функциональностью, поэтому рассмотрим основные характеристики.

Цвет задается 3 целочисленными характеристиками, соответствующими модели RGB, – красный, зеленый, синий. Каждая из них может иметь значение от 0 до 255 (тем не менее, их тип определен как int). В результате (0, 0, 0) соответствует черному, а (255, 255, 255) – белому.

Класс **Color** является неизменяемым, то есть, создав экземпляр, соответствующий какому-либо цвету, изменить параметры RGB уже невозможно. Это позволяет объявить в классе Color ряд констант, описывающих базовые цвета: белый, черный, красный, желтый и так далее. Например, вместо того, чтобы задавать синий цвет числовыми параметрами (0, 0, 255), можно воспользоваться константами **Color.blue** или **Color.BLUE** (второй вариант появился в более поздних версиях).

Для работы со свойством компонента foreground применяют методы **setForeground** и **getForeground**, а для background – **setBackground** и **getBackground**.

Класс Container

Контейнер описывается классом **Container**, который является наследником **Component**, а значит, обладает всеми свойствами графического компонента. Однако основная его задача – группировать другие компоненты. Для этого в нем объявлен целый ряд методов. Для добавления служит метод add, для удаления – remove и removeAll (последний удаляет все компоненты).

Добавляемые компоненты хранятся в упорядоченном списке, поэтому для удаления можно указать либо ссылку на компонент, который и будет удален, либо его порядковый номер в контейнере. Также определены методы для получения компонент, присутствующих в контейнере, – все они довольно очевидны, поэтому перечислим их с краткими пояснениями:

- **getComponent**(int n) – возвращает компонент с указанным порядковым номером;
- **getComponents**() – возвращает все компоненты в виде массива;
- **getComponentCount**() – возвращает количество компонент;
- **getComponentAt**(int x, int y) или (Point p) – возвращает компонент, который включает в себя указанную точку;
- **findComponentAt**(int x, int y) или (Point p) – возвращает видимый компонент, включающий в себя указанную точку.

Положение компонента (location) задается координатами левого верхнего угла. Важно, что эти значения отсчитываются от левого верхнего угла контейнера, который таким образом является центром системы координат для каждого находящегося в нем компонента. Если важно расположение компонента на экране безотносительно его контейнера, можно воспользоваться методом **getLocationOnScreen**.

Благодаря наследованию контейнер также имеет свойство **size**. Этот размер задается независимо от размера и положения вложенных компонент. Таким образом, компоненты могут располагаться частично или полностью за пределами своего контейнера (что это означает, будет рассмотрено ниже, но принципиально это допустимо).

Раз контейнер наследуется от Component, он сам является компонентом, а значит, может быть добавлен в другой, вышестоящий контейнер. В то же время компонент может находиться лишь в одном контейнере. Это означает, что все элементы сложного пользовательского интерфейса объединяются в иерархическое дерево. Такая организация не только облегчает операции над

ними, но и задает основные свойства всей работы AWT. Одним из них является принцип отрисовки компонентов.

Алгоритм отрисовки компонента

Рассмотрим алгоритм отрисовки отдельного компонента, что определяет его внешний вид? Для этой задачи предназначен метод **paint**. Этот метод вызывается каждый раз, когда необходимо отобразить компонент на экране. У него есть один аргумент, тип которого – абстрактный класс **Graphics**. В этом классе определено множество методов для отрисовки простейших графических элементов – линий, прямоугольников и многоугольников, окружностей и овалов, текста, картинок и т.д.

Наследники *класса* ***Component переопределяют*** *метод* ***paint*** *и, пользуясь методами Graphics, задают алгоритм прорисовки своего внешнего вида*:

```
public void paint(Graphics g) {
  g.drawLine(0, 0, getWidth(), getHeight());
  g.drawLine(0, getHeight(), getWidth(), 0);
}
```

Ключевым классом при выполнении всех графических операций является **Graphics**. Назначение класса:

- определяет поверхность рисования;
- определяет методы рисования;
- определяет атрибуты для методов рисования.

Рассмотрим некоторые методы класса Graphics.

drawLine(x1, y1, x2, y2)

Этот метод отображает линию толщиной в 1 пиксел, проходящую из точки (x1, y1) в (x2, y2).

drawRect(int x, int y, int width, int height)

Этот метод отображает прямоугольник, чей левый верхний угол находится в точке (x, y), а ширина и высота равняются width и height

соответственно. Правая сторона пройдет по линии x+width, а нижняя – y+height.

fillRect(int x, int y, int width, int height)

Этот метод закрашивает прямоугольник. Левая и правая стороны прямоугольника проходят по линиям x и x+width-1 соответственно, а верхняя и нижняя – y и y+height-1 соответственно. Таким образом, чтобы зарисовать все пикселы компонента, необходимо передать следующие аргументы:

g.fillRect(0, 0, getWidth(), getHeight());

drawOval(int x, int y, int width, int height)

Этот метод рисует овал, вписанный в прямоугольник, задаваемый указанными параметрами. Очевидно, что если прямоугольник имеет равные стороны (т.е. является квадратом), овал становится окружностью.

fillOval(int x, int y, int width, int height)

Этот метод закрашивает указанный овал.

drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)

Этот метод рисует дугу – часть овала, задаваемого первыми четырьмя параметрами. Дуга начинается с угла startAngle и имеет угловой размер arcAngle. Начальный угол соответствует направлению часовой стрелки, указывающей на 3 часа. Угловой размер отсчитывается против часовой стрелки. Таким образом, размер в 90 градусов соответствует дуге в четверть овала (верхнюю правую). Углы "растянуты" в соответствии с размером прямоугольника. В результате, например, угловой размер в 45 градусов всегда задает границу дуги по линии, проходящей из центра прямоугольника в его правый верхний угол.

fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)

Этот метод закрашивает сектор, ограниченный дугой, задаваемой параметрами.

drawString(String text, int x, int y)

Этот метод выводит на экран текст, задаваемый первым параметром. Точка (x, y) задает позицию самого левого символа. Для наглядности приведем пример:

```
g.drawString("abcdefgh", 15, 15);
g.drawLine(15, 15, 115, 15);
```

Методы repaint и update

Кроме paint в классе Component объявлены еще два метода, отвечающие за прорисовку компонента. Как было рассмотрено, вызов paint инициируется операционной системой, если возникает необходимость перерисовать окно приложения, или часть его. Однако может потребоваться обновить внешний вид, руководствуясь программной логикой. Например, отобразить результат операции вычисления, или работы с сетью. Можно изменить состояние компонента (значение его полей), но операционная система не отследит такое изменение и не инициирует перерисовку.

Для программной инициализации перерисовки компонента служит метод **repaint**. Конечно, у него нет аргумента типа Graphics, поскольку программист не должен создавать экземпляры этого класса (точнее, его наследников, ведь Graphics – абстрактный класс). Метод repaint можно вызывать без аргументов. В этом случае компонент будет перерисован максимально быстро. Можно указать аргумент типа long – количество миллисекунд. Система инициализирует перерисовку спустя указанное время. Можно указать четыре числа типа int (x, y, width, height), задавая прямоугольную область компонента, которая нуждается в перерисовке. Наконец, можно указать все 5 параметров – и задержку по времени, и область перерисовки.

Если перерисовка инициируется приложением, то система вызывает не метод paint, а метод **update**. У него уже есть аргумент типа Graphics и по умолчанию он лишь закрашивает всю область компонента фоновым цветом

(свойство background), а затем вызывает метод paint. Зачем же было вводить этот дополнительный метод, если можно было сразу вызвать paint? Дело в том, что поскольку перерисовка инициируется приложением, для сложных компонентов становится возможной некоторая оптимизация обновления внешнего вида. Например, если изменение заключается в появлении нового графического элемента, то можно избежать повторной перерисовки остальных элементов – переопределить метод update и реализовать в нем отображение одного только нового элемента. Если же компонент имеет простую структуру, можно оставить метод update без изменений.

Апплеты

Кроме приложений, язык Java позволяет создавать **апплеты** (applets). Это программы, работающие в среде другой программы — браузера. Апплеты не нуждаются в окне верхнего уровня — им служит окно браузера. Они не запускаются JVM — их загружает браузер, который сам запускает JVM для выполнения апплета. Эти особенности отражаются на написании программы апплета [2].

С точки зрения языка Java, апплет — это всякое расширение класса Applet, который, в свою очередь, расширяет класс panel. Таким образом, апплет — это панель специального вида, контейнер для размещения компонентов с дополнительными свойствами и методами. Менеджером размещения компонентов по умолчанию, как и в классе Panel, служит FiowLayout. Класс Applet находится в пакете java. applet, в котором кроме него есть только три интерфейса, реализованные в браузере. Надо заметить, что не все браузеры реализуют эти интерфейсы полностью.

Поскольку JVM не запускает апплет, отпадает необходимость в методе main (), его нет в апплетах. В апплетах редко встречается конструктор. Дело в том, что при запуске первого создается его контекст. Во время выполнения

конструктора контекст еще не сформирован, поэтому не все начальные значения удается определить в конструкторе.

Начальные действия, обычно выполняемые в конструкторе и методе main(), в апплете записываются в метод **init**() класса Applet. Этот метод автоматачески запускается исполняющей системой Java браузера сразу же после загрузки апплета. Вот как он выглядит в исходном коде класса Applet:

public void init(){}

Метод init () не имеет аргументов, не возвращает значения и должен переопределяться в каждом апплете — подклассе класса Applet. Обратные действия — завершение работы, освобождение ресурсов — записываются при необходимости в метод **destroy** о, тоже выполняющийся автоматически при выгрузке апплета. В классе Applet есть пустая реализация этого метода.

Кроме методов init() и destroy() в классе Applet присутствуют еще два пустых метода, выполняющихся автоматически. Браузер должен обращаться к методу **start**() при каждом появлении апплета на экране и обращаться к методу stop(), когда апплет уходит с экрана. В методе **stop**() можно определить действия, приостанавливающие работу апплета, в методе start() — возобновляющие ее. Надо сразу же заметить, что не все браузеры обращаются к этим методам как должно [2].

Метод **paint**(Graphics g) вызывается каждый раз при повреждении апплета. AWT следит за состоянием окон в системе и замечает такие случаи, как, например, перекрытие окна апплета другим окном. В таких случаях, после того, как апплет снова оказывается видимым, для восстановления его изображения вызывается метод paint(Graphics g).

Перерисовка содержимого апплета выполняется методом **update**(). Для инициации update() предусмотрены несколько вариантов метода repaint, который в свою очередь вызывает метод update:

repaint();

repaint(time);

repaint(x, y, w, h);

repaint(time, x, y, w, h);

Приведем пример простого апплета, выводящего текст на экран.

```
import java.awt.*;
import java.applet.*;
public class HelloWorld extends Applet{
public void paint(Graphics g){
g.drawstring("Hello, XXI century World!!!", 10, 30);
}
}
```

Эта программа записывается в файл **HelloWorld.java** и компилируется как обычно: **javac HelloWorld.java**.

Компилятор создает файл HelloWorkLclass, но воспользоваться для его выполнения интерпретатором java теперь нельзя — нет метода main(). Вместо интерпретации надо дать указание браузеру для запуска апплета.

Все указания браузеру даются пометками, тегами (tags), на языке HTML (HyperText Markup Language). В частности, указание на запуск апплета дается в теге <applet>. В нем обязательно задается имя файла с классом апплета параметром code, ширина width и высота height панели апплета в пикселах. Полностью текст HTML для запуска нашего апплета на странице браузера приведен ниже:

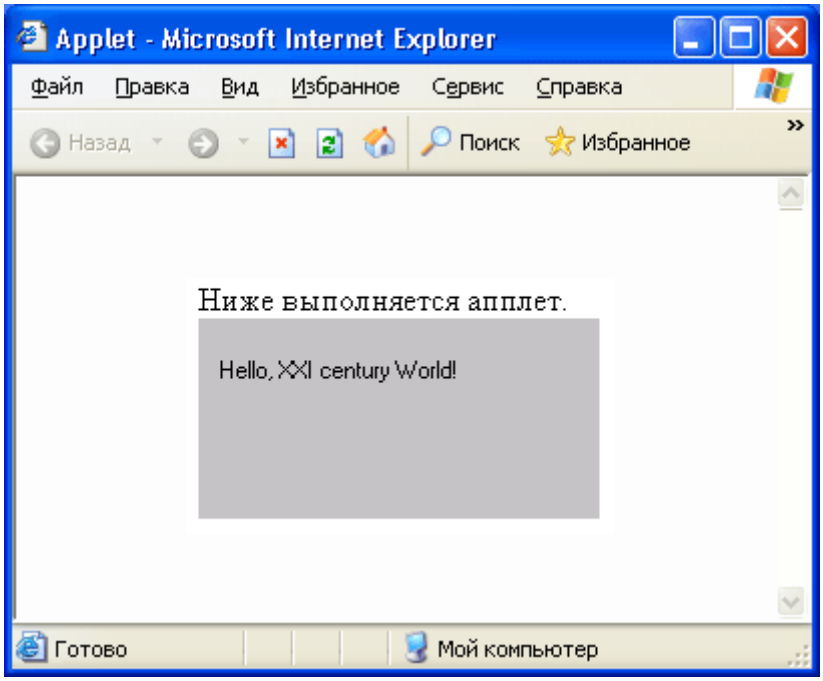
```
<html>
<head><title> Applet</title></head> <body>
Ниже выполняется апплет.<br>
<applet code = "HeiioWorid.class" width = "200" height = "100">
</applet>
</body>
</html>
```

Этот текст заносится в файл с расширением **html** или **htm**, например. HelloWorld.html. Имя файла произвольно, никак не связано с апплетом или классом апплета. Оба файла — HelloWorld.html и HelloWorld.class — помещаются в один каталог на сервере, и файл HelloWorld.html загружается в

браузер, который может находиться в любом месте Internet. Браузер, просматривая HTML-файл, выполнит тег <appiet> и загрузит апплет.

После загрузки апплет появится в окне браузера, как показано ниже [2].

(**Hello.zip**)



Апплет HelloWorld в окне Internet Explorer

В этом простом примере можно заметить еще две особенности апплетов. Во-первых, размер апплета задается не в нем, а в теге <applet>. Это очень удобно, можно менять размер апплета, не компилируя его заново. Можно организовать апплет невидимым, сделав его размером в один пиксел. Кроме того, размер апплета разрешается задать в процентах по отношению к размеру окна браузера, например,

<applet code = "HelloWorld.class" width = "100%" height = "100%">

Во-вторых, как видно на рисунке, у апплета серый фон. Такой фон был в первых браузерах, и апплет не выделялся из текста в окне браузера. Теперь в браузерах принят белый фон, его можно установить обычным для компонентов методом setBackground(Color.white), обратившись к нему в методе init ().

В настоящее время синтаксис тега <APPLET> таков [1]:

```
<APPLET
  CODE = appletFile
  WIDTH = pixels
  HEIGHT = pixels
  [ARCHIVE = jarFiles]
  [CODEBASE = codebaseURL]
  [ALT = alternateText]
  [NAME = appletInstanceName]
  [ALIGN = alignment]
  [VSPACE = pixels]
  [HSPACE = pixels]
>
[HTML-текст, отображаемый при отсутствии
поддержки Java]
</APPLET>
```

- CODE = appletClassFile ; CODE – обязательный атрибут, задающий имя файла, в котором содержится описание класса апплета. Имя файла задается относительно codebase, то есть либо от текущего каталога, либо от каталога, указанного в атрибуте CODEBASE.
- WIDTH = pixels
- HEIGHT = pixels ; WIDTH и HEIGHT - обязательные атрибуты, задающие размер области апплета на HTML -странице.
- ARCHIVE = jarFiles ; Этот необязательный атрибут задает список jar -файлов (разделяется запятыми), которые предварительно загружаются в Web -браузер. В них могут содержаться классы, изображения, звук и любые другие ресурсы, необходимые апплету. Архивирование наиболее необходимо именно апплетам, так как их код и ресурсы передаются через сеть.
- CODEBASE = codebaseURL ; CODEBASE – необязательный атрибут, задающий базовый URL кода апплета; является каталогом, в котором будет выполняться поиск исполняемого файла апплета (задаваемого в признаке CODE). Если этот атрибут не задан, по умолчанию используется каталог данного HTML -документа. С помощью этого атрибута можно на странице одного сайта разместить апплет, находящийся на другом сайте.

- ALT = alternateAppletText ; Признак ALT – необязательный атрибут, задающий короткое текстовое сообщение, которое должно быть выведено (как правило, в виде всплывающей подсказки при нахождении курсора мыши над областью апплета) в том случае, если используемый браузер распознает синтаксис тега <applet>, но выполнять апплеты не умеет. Это не то же самое, что HTML -текст, который можно вставлять между <applet> и </applet> для браузеров, вообще не поддерживающих апплетов.
- NAME = appletInstanceName ; NAME – необязательный атрибут, используемый для присвоения имени данному экземпляру апплета. Имена апплетам нужны для того, чтобы другие апплеты на этой же странице могли находить их и общаться с ними, а также для обращений из Java Script.
- ALIGN = alignment
- VSPACE = pixels
- HSPACE = pixels ; Эти три необязательных атрибута предназначены для того же, что и в теге IMG. ALIGN задает стиль выравнивания апплета, возможные значения: LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM, ABSBOTTOM.

В состав JDK любой версии входит программа **appletviewer**. Это простейший браузер, предназначенный для запуска апплетов в целях отладки. Если под рукой нет Internet-браузера, можно воспользоваться им. Appletviewer запускается из командной строки:

appletviewer HelloWorld.html

На рисунке appletviewer показывает апплет HelloWorld.

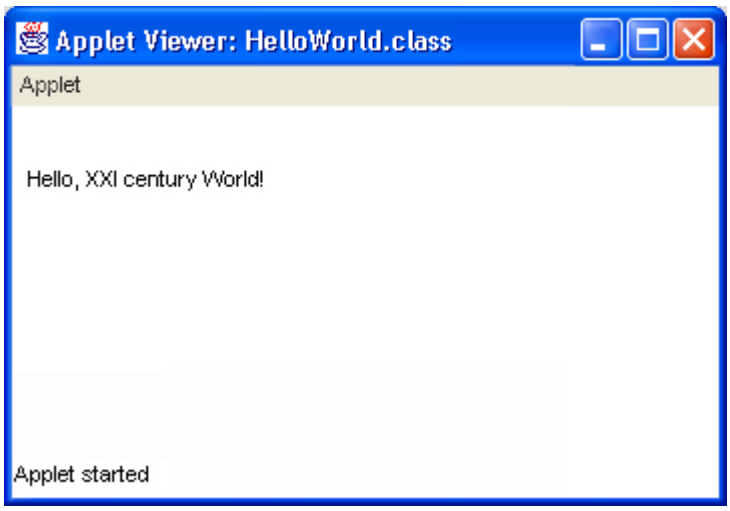


Рисунок. Апплет HelloWorld в окне программы appletviewer

Приведем пример невидимого апплета. В нижней строке браузера — строке состояния (status bar) — отражаются сведения о загрузке файлов. Апплет может записать в нее любую строку str методом showstatus(string str).

```
// Файл RunningString.Java
import java.awt.*;
import java.applet.*;
public class Runningstring extends Applet{
private Boolean go;
public void start(){
go = true;
sendMessage("Эта строка выводится апплетом"); }

public void sendMessage(String s){
String s1 = s+" ";
while(go){
showStatus(s);
try{
Thread.sleep(200);
}catch(Exception e){}
s = s1.substring(1)+s.charAt(0);
s1 =s; }
}
public void stop(){
go = false; } }
```

(**Running.zip**)

Следующий апплет имеет более сложную структуру: по экрану движется круг, который упруго отражается от границ области. Движение происходит непрерывно. Особенность данного апплета состоит в том, что он реализован с помощью дополнительного потока Thread, который отвечает за движение круга. Класс апплета реализует интерфейс Runnable. Программный код приведен ниже (ссылка на файл **bouncingcircle.java**).

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


//Создаем собственный класс,который наследуется от класса Applet
//Данный класс реализует методы интерфейса Runnable

public class BouncingCircle extends Applet implements Runnable {
   int x = 150, y = 50, r = 50;  // Центр и радиус круга
   int dx = 11, dy = 7;          // Движение круга по горизонт и вертик
   Thread animator;              // Нить, которая осуществляет анимацию
   volatile boolean pleaseStop;  // Флаг остановки движения

//Метод для рисования окружности красным цветом
   public void paint(Graphics g) {
          g.setColor(Color.red);           //установка цвета для g
          g.fillOval(x-r, y-r, r*2, r*2); //прорисовка круга
   }

//Метод двигает круг и "отражает" его при ударе круга о стенку,
//затем вызывает перерисовку.
//Данный метод вызывается многократно анимационным потоком

   public void animate() {
          Rectangle bounds = getBounds();  //Получение размера окна программы

          if ((x - r + dx < 0) || (x + r + dx > bounds.width)) dx = -dx;
          if ((y - r + dy < 0) || (y + r + dy > bounds.height)) dy = -dy;

          // Изменение координат круга, по сути - движение.
          x += dx;  y += dy;

          //"Просим" браузер вызвать метод paint() для отрисовки
          // круга в новой позиции
          repaint();
   }
```

```
        /*Это метод из интерфейса Runnable. Это тело потока исполнения,
        осуществляющего анимацию. Сам поток создается и
        запускается методом start()
        */
    public void run() {
        while(!pleaseStop) {            // Цикл до тех пор, пока не будет
                                        //команды остановиться.
            animate();                  // Обновляем положение и перерисовываем
            try { Thread.sleep(100); }  // Ждем 100 миллисекунд
            catch(InterruptedException e) {} // Игнорируем прерывания
        }
    }

    //Запускаем анимацию при запуске апплета браузером
    public void start() {
        animator = new Thread(this);   // Создаем поток исполнения
        pleaseStop = false;            // Пока не просим его остановиться
        animator.start();              // Запускаем поток
    }
      //Останавливаем анимацию, когда браузер останавливает апплет
    public void stop() {    pleaseStop = true;    }
}
```

Данный апплет работает следующим образом:

(**BouncingCircle**)

4. Порядок выполнения работы

- изучить предлагаемый теоретический материал;
- создайте следующие программы:

- Запустить апплет, рассмотренный в примере презентации (HelloWorld), используя два метода запуска – через html-страницу и через appletviewer.

- Модифицировать код программы bouncingcircle таким образом, чтобы вместо круга движение осуществлял экземпляр реализованного ранее (в лабораторной работе №2) класса ColorableRect(), как показано ниже: (**BouncingBox**).
- Модифицировать код предыдущей программы таким образом, чтобы движение осуществляли сразу 10 экземпляров класса Rectangle, 10 класса DrawableRect и 10 экземпляров класса ColorableRect. Все созданные объекты должны хранится в одном массиве с типом класса родителя Rectangle. Ниже показано, как должен выглядеть апплет. (**BouncingBox2**).

5. Содержание отчета

В отчете следует указать:

1. цель работы;
2. введение;
3. программно-аппаратные средства, используемые при выполнении работы;
4. основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения лабораторного практикума;
5. заключение (описание результатов и выводы);
6. список используемой литературы.

6. Литература

1. Вязовик, Н.А. Программирование на Java. [электронный ресурс] http://www.intuit.ru/department/pl/javapl (7.01.2012).
2. Основы программирования в Java [электронный ресурс] http://www.projava.net (07.01.2012).
3. Хорстманн К.С., Корнелл Г. Библиотека профессионала. JAVA 2. Том 1. Основы. 8-е издание. Пер. с англ. – М.: ООО Издательский дом “Вильямс”, 2008. – 816 с.
4. Эккель Б. Философия JAVA. 4-е издание. – СПб.: Питер, 2009. – 638с.
5. Информационные материалы с официального сайт а разработчиков http://www.oracle.com/technetwork/java/javase/downloads/index.html [электронный ресурс] (7.01.2012).

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра ЮНЕСКО по новым информационным технологиям

С.Н. Карабцев

ПРОГРАММИРОВАНИЕ НА JAVA

Лабораторный практикум

Математический факультет

специальность 010300.62 – фундаментальная информатика и информационные технологии;
специальность 010400.62 – прикладная математика и информатика;
специальность 010500.62 – математическое обеспечение и администрирование информационных систем.

Кемерово, 2012

ЛАБОРАТОРНАЯ РАБОТА №4.
ОБРАБОТКА СОБЫТИЙ В JAVA

1. Цель работы

Познакомиться с моделью событий в графических компонентах и научиться обрабатывать возникающие события с помощью интерфейсов.

2. Методические указания

Лабораторная работа направлена на приобретение навыка создания программ на Java, в которых присутствуют графические элементы интерфейса, генерирующие события. В данной работе вы познакомитесь с общими принципами обработки событий, а также научитесь создавать слушатели событий для наиболее распространенных элементов графического интерфейса – кнопок Button и курсора мыши.

Требования к результатам выполнения лабораторного практикума:

- при выполнении задания необходимо сопровождать все реализованные процедуры и функции набором тестовых входных и выходных данных и описаниями к ним;
- компиляцию, запуск программ выполнять различными способами;
- по завершении выполнения задания составить отчет о проделанной работе.

При составлении и оформлении отчета следует придерживаться рекомендаций, представленных на следующей странице: http://unesco.kemsu.ru/student/rule/rule.html.

3. Теоретический материал

Модель обработки событий в AWT представляет собой, по существу, модель обратных вызовов (callback). При создании GUI-элемента ему сообщается, какой метод или методыон должен вызывать при возникновении в нем определенного события (нажатия кнопки, мыши и т.п.). Эту модель очень легко использовать в C++, поскольку этот язык позволяет оперировать указателями на методы (чтобы определить обратный вызов, необходимо всего лишь передать указатель на функцию). Однако в Java это недопустимо (методы не являются объектами). Поэтому для реализации модели необходимо определить класс, реализующий некоторый специальный интерфейс. Затем можно передать экземпляр такого класса GUI-элементу, обеспечивая таким образом обратный вызов. Когда наступит ожидаемое событие, GUI-элемент вызовет соответствующий метод объекта, определенного ранее.

Модель обработки событий Java используется как в пакете AWT, так и в JavaBeans API. В этой модели разным типам событий соответствуют различные классы Java. Каждое событие является подклассом класса java.util.EventObject. События пакета AWT, которые и рассматриваются в данной главе, являются подклассом java.awt.AWTEvent. Для удобства события различных типов пакета AWT (например, MouseEvent или ActionEvent) помещены новый пакет java.awt.event. Для каждого события существует порождающий его объект, который можно получить с помощью метода getSource(), и каждому событию пакета AWT соответствует определенный идентификатор, который позволяет получить метод getID(). Это значение используется для того, чтобы отличать события различных типов, которые могут описываться одним и тем же классом событий. Например, для класса FocusEvent возможны два типа событий:

FocusEvent.FOCUS_GAINED и FocusEvent.FOCUS_LOST. Подклассы событий содержат информацию, связанную с данным типом события. Например, в классе MouseEvent существуют методы getX(), getY() и

getClickCount (). Этот класс наследует, в числе прочих, и методы getModifiers() и getWhen().

Модель обработки событий Java базируется на концепции слушателя событий. Слушателем события является объект, заинтересованный в получении данного события. В объекте, который порождает событие (в источнике событий), содержится список слушателей, заинтересованных в получении уведомления о том, что данное событие произошло, а также методы, которые позволяют слушателям добавлять или удалять себя из этого списка. Когда источник порождает событие (или когда объект источника зарегистрирует событие, связанное с вводом информации пользователем), он оповещает все объекты слушателей событий о том, что данное событие произошло.

Источник события оповещает объект слушателя путем вызова специального метода и передачи ему объекта события (экземпляра подкласса EventObject). Для того чтобы источник мог вызвать данный метод, он должен быть реализован для каждого слушателя. Это объясняется тем, что все слушатели событий определенного типа должны реализовывать соответствующий интерфейс. Например, объекты слушателей событий ActionEvent должны реализовывать интерфейс ActionListener. В пакете java.awt.event содержатся интерфейсы слушателей для каждого из определенных в нем типов событий (например, для событий MouseEvent здесь определено два интерфейса слушателей: MouseListener и MouseMotionListener). Все интерфейсы слушателей событий являются расширениями интерфейса java.util.EventListener. В этом интерфейсе не определяется ни один из методов, но он играет роль интерфейса-метки, в котором однозначно определены все слушатели событий как таковые.

В интерфейсе слушателя событий может определяться несколько методов. Например, класс событий, подобный MouseEvent, описывает несколько событий, связанных с мышью, таких как события нажатия и отпускания кнопки мыши. Эти события вызывают различные методы

соответствующего слушателя. По установленному соглашению, методам слушателей событий может быть передан один единственный аргумент, являющийся объектом того события, которое соответствует данному слушателю. В этом объекте должна содержаться вся информация, необходимая программе для формирования реакции на данное событие. В таблице 1 приведены определенные в пакете java.awt.event типы событий, соответствующие им слушатели, а также методы, определенные в каждом интерфейсе слушателя [1].

Таблица 1. Типы событий, слушатели и методы слушателей в Java

Класс события	Интерфейс слушателя	Методы слушателя
ActionEvent	ActionListener	actionPerformed()
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged()
ComponentEvent	ComponentListener	componentHidden() componentMoved() componentResized() componentShown()
ContainerEvent	ContainerListener	componentAdded() componentRemoved()
FocusEvent	FocusListener	focusGained() focusLost ()
ItemEvent	ItemListener	itemStateChanged()
KeyEvent	KeyListener	keyPressed() keyReleased() keyTyped()
MouseEvent	MouseListener	mouseClicked() mouseEntered() mouseExited() mousePressed() mouseReleased()
MouseMotionEvent	MouseMotionListener	mouseDragged() mouseMoved()
TextEvent	TextListener	textValueChanged()
WindowEvent	WindowListener	windowActivated() windowClosed() windowClosing() windowDeactivated() windowDeiconified() windowIconified() windowOpened()

Для каждого интерфейса слушателей событий, содержащего несколько методов, в пакете java.awt.event определен простой класс-адаптер, который обеспечивает пустое тело для каждого из методов соответствующего интерфейса. Когда нужен только один или два таких метода, иногда проще получить подкласс класса-адаптера, чем реализовать интерфейс самостоятельно. При получении подкласса адаптера требуется лишь

переопределить те методы, которые нужны, а при прямой реализации интерфейса необходимо определить все методы, в том числе и ненужные в данной программе. Заранее определенные классы-адаптеры называются так же, как и интерфейсы, которые они реализуют, но в этих названиях Listener заменяется на Adapter: MouseAdapter, WindowAdapter и т.д. Как только реализован интерфейс слушателя или получены подклассы класса-адаптера, необходимо создать экземпляр нового класса, чтобы определить конкретный объект слушателя событий. Затем этот слушатель должен быть зарегистрирован соответствующим источником событий. В программах пакета AWT источником событий всегда является какой-нибудь элемент пакета. В методах регистрации слушателей событий используются стандартные соглашения об именах: если источник событий порождает события типа X, в нем существует метод addXListener () для добавления слушателя и метод removeXListener() для его удаления. Одной из приятных особенностей модели обработки событий Java является возможность легко определять типы событий, которые могут порождаться данным элементом. Для этого следует просто просмотреть, какие методы зарегистрированы для его слушателя событий. Например, из описания API для объекта класса Button следует, что он порождает события ActionEvent. В таблице 2 приведен список элементов пакета AWT и событий, которые они порождают [1].

Таблица 2. Элементы пакета AWT и порождаемые ими события в Java1.1

Элемент	Порождаемое событие	Значение
Button	ActionEvent	Пользователь нажал кнопку
CheckBox	ItemEvent	Пользователь установил или сбросил флажок
CheckBoxMenuItem	ItemEvent	Пользователь установил или сбросил флажок рядом с пунктом меню
Choice	ItemEvent	Пользователь выбрал элемент списка или отменил его выбор
Component	ComponentEvent	Элемент либо перемещен, либо он стал скрытым, либо видимым
	FocusEvent	Элемент получил или потерял фокус ввода
	KeyEvent	Пользователь нажал или отпустил клавишу
	MouseEvent	Пользователь нажал или отпустил кнопку мыши, либо курсор мыши вошел или покинул область, занимаемую элементом, либо пользователь просто переместил мышь или переместил мышь при нажатой кнопке мыши
Container	ContainerEvent	Элемент добавлен в контейнер или удален из него
List	ActionEvent	Пользователь выполнил двойной щелчок мыши на элементе списка
	ItemEvent	Пользователь выбрал элемент списка или отменил выбор
MenuItem	ActionEvent	Пользователь выбрал пункт меню
Scrollbar	AdjustmentEvent	Пользователь осуществил прокрутку
TextComponent	TextEvent	Пользователь внес изменения в текст элемента
TextField	ActionEvent	Пользователь закончил редактирование текста элемента
Window	WindowEvent	Окно было открыто, закрыто, представлено в виде пиктограммы, восстановлено или требует восстановления

Примеры

Рассмотрим код программы, в котором приведена обработка события, возникающего в экземплярах класса JButton. Для обработки события применяется внутренний класс ButtonListener, который объявлен и описан внутри класса апплета Button2. При нажатии на кнопку, с нее считывается надпись и помещается в компонент JTextField.

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

public class Button2 extends JApplet {
```

```
  private JButton
    b1 = new JButton("Button 1"),
    b2 = new JButton("Button 2");
    private JTextField txt = new JTextField(10);

//класс обработчик событий, реализующий интерфейс ActionListener
  class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      String name = ((JButton)e.getSource()).getText();
      txt.setText(name);
    }
  }
//создание экземпляра класса-обработчика
  private ButtonListener bl = new ButtonListener();

public void init() {
    b1.addActionListener(bl);   //регистрация слушателя события
    b2.addActionListener(bl);  // экземпляра bl класса ButtonListener
    setLayout(new FlowLayout());
    add(b1);
    add(b2);
    add(txt);
  }
}
```

Данный апплет работает следующим образом. (**Button_Label**)

Еще один более сложный пример обработки событий, возникающих в кнопках Button, приведен здесь (**Button_Color/ButtonApplet.java**).

Следующий пример – рисование каракулей на апплете с помощью мыши. Это классический апплет, в котором используется модель обработки событий Java. В этом примере реализованы интерфейсы MouseListener и MouseMotionListener, регистрирующие себя с помощью своих же методов addMouseListener() и addMouseMotionListener().

Данную программу можно реализовать различными способами. Рассмотрим сначала ***способ*** [1], в котором в качестве слушателя событий выступает сам апплет Scribble2, а не некоторый внутренний класс.

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

```
import java.awt.*;
import java.awt.event.*;

//сам апплет реализует интерфейсы,т.е. является слушателем
public class Scribble2 extends Applet implements
MouseListener, MouseMotionListener {
private int last_x, last_y;

public void init() {
// Сообщает данному апплету о том, какие объекты
// классов MouseListener и MouseMotionListener он должен оповещать
// о событиях, связанных с мышью и ее перемещением.
// Поскольку интерфейс реализуется в самом апплете,
// при этом будут вызываться методы апплета.
this.addMouseListener(this) ;
this.addMouseMotionListener(this);
}

// Метод интерфейса MouseListener. Вызывается при нажатии
// пользователем кнопки мыши.
public void mousePressed(MouseEvent e) {
last_x = e.getX();
last_y = e.getY();
}

// Метод интерфейса MouseMotionListener. Вызывается при
// перемещении мыши с нажатой кнопкой.
public void mouseDragged(MouseEvent e) {
Graphics g = this.getGraphics();
int x = e.getX(), y = e.getY();
g.drawLine(last_x, last_y, x, y);
last_x = x; last_y = y;
}
// Другие, не используемые методы интерфейса MouseListener.
public void mouseReleased(MouseEvent e) {;}
public void mouseClicked(MouseEvent e) {;}
public void mouseEntered(MouseEvent e) {;}
public void mouseExited(MouseEvent e) {;}
// Другой метод интерфейса MouseMotionListener.
public void mouseMoved(MouseEvent e) {;}
}
```

Демонстрация работы данного апплета представлена ниже. (**Scribble_simple/Scribble2**)

Рассмотрим ***второй вариант*** [1] реализации данного примера с привлечением встроенных классов. Модель обработки событий Java разработана с учетом того, чтобы хорошо сочетаться с другой особенностью Java: встроенными классами. В следующем примере показано, как изменится данный апплет, если слушатели событий будут реализованы в виде анонимных встроенных классов. Обратите внимание на компактность данного варианта программы. Новая особенность, добавленная в апплет - кнопка Clear. Для этой кнопки зарегистрирован объект ActionListener, а сама она выполняет очистку экрана при наступлении соответствующего события.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Scribble3 extends Applet {
int last_x, last_y;
public void init() {
// Определяет, создает и регистрирует объект MouseListener.
this.addMouseListener(new MouseAdapter() {
public void mousePressed(MouseEvent e) {
last_x = e.getX(); last_y = e.getY();
}
});
// Определяет, создает и регистрирует объект MouseMotionListener.
this.addMouseMotionListener(new MouseMotionAdapter() {
public void mouseDragged(MouseEvent e) {
Graphics g = getGraphics();
int x = e.getX(), y= e.getY();
g.setColor(Color.black);
g.drawLine(last_x, last_y, x, y);
last_x = x; last_y = y;
}
});
// Создает кнопку Clear.
Button b = new Button("Clear");
// Определяет, создает и регистрирует объект слушателя
// для обработки события, связанного с нажатием кнопки.
b.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
// стирание каракулей
Graphics g = getGraphics();
```

```
g.setColor(getBackground());
g.fillRect(0, 0, getSize().width, getSize().height);
}
});
// Добавляет кнопку в апплет.
this.add(b);
}
}
```

Демонстрация работы данного апплета представлена ниже. (**Scribble_simple2/Scribble3**)

Стоит заметить, что в аппретах с рисованием каракулей не выполняется никаких действий по различению левой и правой кнопок мыши, т.е. рисовать можно ими обеими. Java предоставляет такие возможности с помощью констант Button1_Mask. В программном коде ниже приведен пример, отличающий левую и правую кнопки в программе:

```
public void mousePressed(MouseEvent e) {
if((e.getModifiers()&MouseEvent.BUTTON1_MASK)!=0){ //левая кнопка
  //действия}
 if((e.getModifiers()&MouseEvent.BUTTON1_MASK)==0){// правая кнопка
  //действия }
```

4. Порядок выполнения работы

- изучить предлагаемый теоретический материал;
- создайте следующие программы:

1. Модифицируйте пример с рисованием каракулей следующим образом:

- рисование только по нажатию левой кнопки мыши;
- правой кнопкой мыши осуществлять изменение цвета (по рандому);

- заменить стиль пера: вместо прямой линии рисовать кружками или квадратиками. Полученный результат должен функционировать следующим образом : (**Scribble_color/Scribble4**)

2. Создать апплет, содержащий три кнопки, по нажатию на которые создаются экземпляры классов Rect, ColoredRect, DrawableRect (реализованные в лабораторной работе №2). Созданные экземпляры помещаются в один массив с типом класса родителя Rect. Реализовать метод захвата мышкой и перемещения по апплету любого прямоугольника, отображенного в окне. Полученный результат должен функционировать следующим образом (**Move_Rect/Dif_Rect**)

5. Содержание отчета

В отчете следует указать:

1. цель работы;
2. введение;
3. программно-аппаратные средства, используемые при выполнении работы;
4. основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения лабораторного практикума;
5. заключение (описание результатов и выводы);
6. список используемой литературы.

6. Литература

1. Вязовик, Н.А. Программирование на Java. [электронный ресурс] http://www.intuit.ru/department/pl/javapl (7.01.2012).

1. Хорстманн К.С., Корнелл Г. Библиотека профессионала. JAVA 2. Том 1. Основы. 8-е издание. Пер. с англ. – М.: ООО Издательский дом “Вильямс”, 2008. – 816 с.
2. Эккель Б. Философия JAVA. 4-е издание. – СПб.: Питер, 2009. – 638с.
3. Информационные материалы с официального сайт а разработчиков http://www.oracle.com/technetwork/java/javase/downloads/index.html [электронный ресурс] (7.01.2012).

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра ЮНЕСКО по новым информационным технологиям

С.Н. Карабцев

ПРОГРАММИРОВАНИЕ НА JAVA

Лабораторный практикум

Математический факультет

специальность 010300.62 – фундаментальная информатика и информационные технологии;
специальность 010400.62 – прикладная математика и информатика;
специальность 010500.62 – математическое обеспечение и администрирование информационных систем.

Кемерово, 2012

ЛАБОРАТОРНАЯ РАБОТА №5.
ПАКЕТ java.io. СЕРИАЛИЗАЦИЯ ОБЪЕКТОВ

1. Цель работы

Целью работы является ознакомление со средствами стандартного пакета языка Java java.io для успешного выполнения операций ввода-вывода из создаваемых программ, а также приобретение практических навыков работы при обмене данными через файл и консоль.

2. Методические указания

Лабораторная работа направлена на приобретение знаний о системе ввода-вывода, реализованной в языке Java с помощью пакета java.io, а также приобретение основных понятии о сериализации и десериализации объектов для подготовки их к дальнейшей отправке по сети.

Требования к результатам выполнения лабораторного практикума:

- при выполнении задания необходимо сопровождать все реализованные процедуры и функции набором тестовых входных и выходных данных и описаниями к ним;
- реализацию программы можно осуществлять как из интегрированной среды разработки Eclipse, так и в блокноте с применением вызовов функций командной строки;
- по завершении выполнения задания составить отчет о проделанной работе.

При составлении и оформлении отчета следует придерживаться рекомендаций, представленных на следующей странице: http://unesco.kemsu.ru/student/rule/rule.html.

3. Теоретический материал

Подавляющее большинство программ обменивается данными с внешним миром. Это, безусловно, делают любые сетевые приложения - они передают и получают информацию от других компьютеров и специальных устройств, подключенных к сети. Оказывается удобным точно таким же образом представлять обмен данными между устройствами внутри одной машины. Так, например, программа может считывать данные с клавиатуры и записывать их в файл, или же наоборот - считывать данные из файла и выводить их на экран. Таким образом, устройства, откуда может производиться считывание информации, могут быть самыми разнообразными - файл, клавиатура, (входящее) сетевое соединение и т.д. То же самое касается и устройств вывода - это может быть файл, экран монитора, принтер, (исходящее) сетевое соединение и т.п. В конечном счете, все данные в компьютерной системе в процессе обработки передаются от устройств ввода к устройствам вывода. Обычно часть вычислительной платформы, которая отвечает за обмен данным, так и называется - система ввода/вывода. В Java она представлена пакетом java.io (input/output). Реализация системы ввода/вывода осложняется не только широким спектром источников и получателей данных, но еще и различными форматами передачи информации. Ею можно обмениваться в двоичном представлении, символьном или текстовом с применением некоторой кодировки.

Поток данных

В Java для описания работы по вводу/выводу используется специальное понятие ***поток данных*** (stream). Поток данных связан с некоторым ***источником*** или ***приемником*** данных, способных получать или предоставлять информацию. Соответственно, потоки делятся на ***входные*** - читающие данные, и на ***выходные*** - передающие (записывающие) данные. Введение концепции stream позволяет отделить программу, обменивающуюся информацией одинаковым образом с любыми

устройствами, от низкоуровневых операций с такими устройствами ввода/вывода.

В Java потоки естественным образом представляются объектами. Описывающие их классы как раз и составляют основную часть пакета java.io. Они довольно разнообразны и отвечают за различную функциональность. Все классы разделены на две части - одни осуществляют ввод данных, другие вывод.

Минимальной "порцией" информации является бит, принимающий значение 0 или 1. Традиционно используется более крупная единица измерения байт, объединяющая 8 бит. Таким образом, значение, представленное 1 байтом, находится в диапазоне от 0 до 28-1=255, или, если использовать знак, от -128 до +127. Примитивный тип **byte** в Java в точности соответствует последнему, знаковому диапазону.

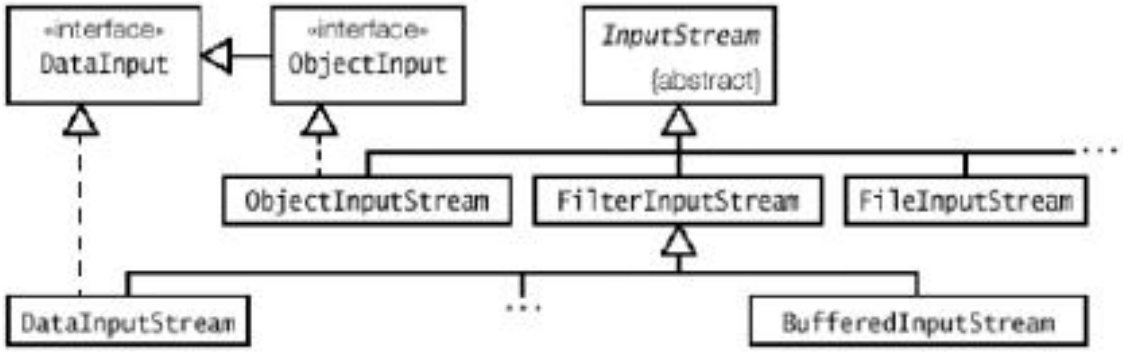
Базовые, наиболее универсальные классы позволяют считывать и записывать информацию именно в виде набора байт. Чтобы их было удобно применять в различных задачах, java.io содержит также классы, преобразующие любые данные в набор байт. Например, если нужно сохранить результаты вычислений - набор значений типа double - в файл, то их можно сначала легко превратить в набор байт, а затем эти байты записать в файл. Аналогичные действия совершаются и в ситуации, когда требуется сохранить объект (т.е. его состояние) - преобразование в набор байт и последующая их запись в файл. Понятно, что при восстановлении данных в обоих рассмотренных случаях проделываются обратные действия - сначала считывается последовательность байт, а затем она преобразовывается в нужный формат.

Байтовые потоки

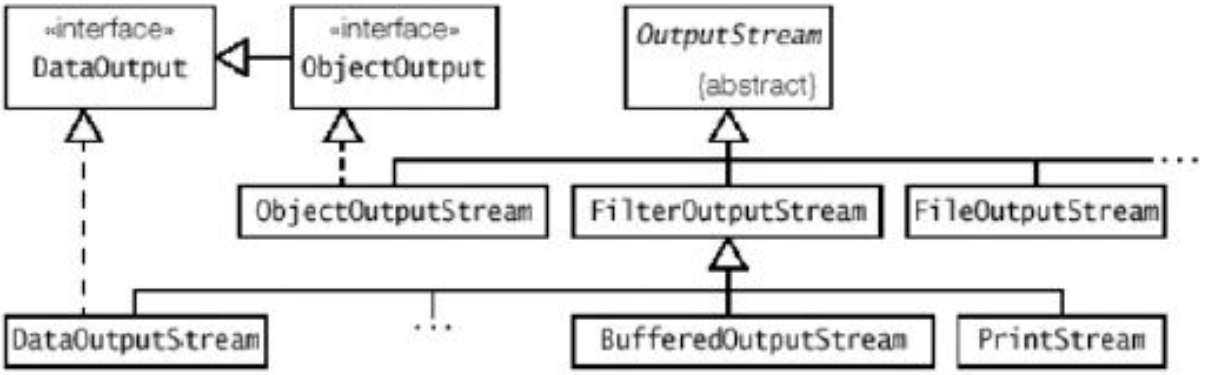
Байтовые потоки определяются в двух иерархиях классов. Наверху этой иерархии – два абстрактных класса: Inputstream и Outputstream (рисунок 1). Каждый из этих абстрактных классов имеет несколько конкретных

подклассов, которые обрабатывают различия между разными устройствами, такими как дисковые файлы, сетевые соединения и даже буферы памяти.

Абстрактные классы Inputstream и Outputstream определяют несколько ключевых методов, которые реализуются другими поточными классами. Два наиболее важных— ***read()*** и ***write()***, которые, соответственно, читают и записывают байты данных. Оба метода объявлены как абстрактные внутри классов inputstream и outputstream и переопределяются производными поточными классами.



а)



б)

Рисунок 1. Иерархия классов. а) – потоки чтения; б) – потоки записи

InputStream - это базовый класс для потоков ввода, т.е. чтения. Соответственно, он описывает базовые методы для работы с байтовыми потоками данных. Эти методы необходимы всем классам, наследующимся от InputStream:

– Метод int ***read()*** – абстрактный. Считывает один байт из потока. Если чтение успешно, то значение между 0..255 представляет собой полученный байт, если конец потока, то значение равно –1.

– Метод int ***read(byte[] b)*** возвращает количество считанных байт и сохраняет их в массив b. Метод так же абстрактный.
– Метод ***read(byte[] b, int offset, int length)*** считывает в массив ***b*** начиная с позиции ***offset length*** символов.
– Метод int ***available()*** возвращает количество байт, на данный момент готовых к чтению из потока. Этот метод применяют, что бы операции чтения не приводили к зависанию, когда данные не готовы к считыванию.
– Метод ***close()*** заканчивает работу с потоком.

В классе ***OutputStream***, аналогичным образом, определяются три метода ***write()*** – один принимающий в качестве параметра int, второй byte[], и третий byte[], плюс два int-числа. Все эти методы возвращают void:

– Метод void ***write(int i)*** пишет только старшие 8 бит, остальные игнорирует.
– Метод void ***write(byte[] b)*** записывает массив в поток.
– Метод void ***write(byte[] b, int offset, int length)*** записывает length-элементов из массива байт в поток, начиная с элемента offset.
– Метод ***flush()*** используется для очистки буфера и записи данных

Существует достаточно большое количество классов-наследников от InputStream, часть которых представлена в таблице

Поточный класс	Значение
BufferedInputstream	Буферизованный поток ввода
BufferedOutputStream	Буферизованный поток вывода
ByteArrayInputstream	Поток ввода, который читает из байт-массива
ByteArrayOutputStream	Поток вывода, который записывает в байт-массив
DataInputstream	Поток ввода, который содержит методы для чтения данных стандартных типов
DataOutputStream	Поток вывода, который содержит методы для записи данных стандартных типов
FileInputstream	Поток ввода, который читает из файла
FileOutputStream	Поток вывода, который записывает в файл
FilterInputStream	Реализует Inputstream
FilterOutputStream	Реализует OutputStream

Inputstream	Абстрактный класс, который описывает поточный ввод
OutputStream	Абстрактный класс, который описывает поточный вывод
PipedInputStream	Канал ввода
PipedOutputStream	Канал вывода
Printstream	Поток вывода, который поддерживает print() и println()
PushbackInputStream	Поток (ввода), который поддерживает однобайтовую операцию "unget", возвращающую байт в поток ввода
RandomAccessFile	Поддерживает ввод/вывод файла произвольного
SequenceInputStream	Поток ввода, который является комбинацией двух или нескольких потоков ввода, которые будут читаться последовательно, один за другим

ByteArrayInputStream и ByteArrayOutputStream

Самый естественный и простой источник, откуда можно считывать байты - это, конечно, массив байт. Класс ByteArrayInputStream представляет поток, считывающий данные из массива байт. Этот класс имеет конструктор, которому в качестве параметра передается массив byte[]. Соответственно, при вызове методов read(), возвращаемые данные будут браться именно из этого массива.

```
byte[] bytes ={1,-1,0};
ByteArrayInputStream in =new ByteArrayInputStream(bytes);
int readedInt =in.read();  //readedInt=1
readedInt =in.read();      //readedInt=255
readedInt =in.read();      //readedInt=0
```

Для записи байт в массив, используется класс ByteArrayOutputStream. Этот класс использует внутри себя объект byte[], куда записывает данные, передаваемые при вызове методов write(). Что бы получить записанные в массив данные, вызывается метод toByteArray().

```
ByteArrayOutputStream out =new ByteArrayOutputStream();
out.write(10);
out.write(11);
byte[] bytes =out.toByteArray();
```

FileInputStream и FileOutputStream

Классы FileInputStream используется для чтения данных из файла. Конструктор этого класса в качестве параметра принимает название файла, из которого будет производиться считывание.

Для записи байт в файл используется класс FileOutputStream. При создании объектов этого класса, то есть при вызовах его конструкторов кроме указания файла, так же можно указать, будут ли данные дописываться в конец файла либо файл будет перезаписан. При этом, если указанный файл не существует, то сразу после создания FileOutputStream он будет создан.

```
byte[] bytesToWrite ={1,2,3};          //что записываем
byte[] bytesReaded =new byte[10];  //куда считываем
String fileName ="d:\\test.txt";

try {
FileOutputStream outFile =new FileOutputStream(fileName);
outFile.write(bytesToWrite);            //запись в файл
outFile.close();

FileInputStream inFile =new FileInputStream(fileName);
int bytesAvailable =inFile.available();  //сколько можно считать
int count =inFile.read(bytesReaded,0,bytesAvailable);
inFile.close();}

catch (FileNotFoundException e){
System.out.println("Невозможно произвести запись в файл:"+fileName);}
catch (IOException e){
System.out.println("Ошибка ввода/вывода:"+e.toString());}
```

При работе с FileInputStream метод ***available***() практически наверняка вернет длину файла, то есть число байт, сколько вообще из него можно считать. Но не стоит закладываться на это при написании программ, которые должны устойчиво работать на различных платформах - метод available() возвращает число, сколько байт может быть на данный момент считано без блокирования.

Классы-фильтры

Задачи, возникающие при вводе/выводе крайне разнообразны - этот может быть считывание байтов из файлов, объектов из файлов, объектов из массивов, буферизованное считывание строк из массивов. В такой ситуации решение путем простого наследования приводит к возникновению слишком большого числа подклассов. Решение же, когда требуется совмещение нескольких свойств, высокоэффективно в виде надстроек. (В ООП этот паттерн называется адаптер.) ***Надстройки*** - наложение дополнительных объектов для получения новых свойств и функций. Таким образом, необходимо создать несколько дополнительных объектов - адаптеров к классам ввода/вывода. В java.io их еще называют ***фильтрами***. При этом надстройка-фильтр, включает в себя интерфейс объекта, на который надстраивается, и поэтому может быть в свою очередь дополнительно быть надстроена.

В java.io интерфейс для таких надстроек ввода/вывода предоставляют классы ***FilterInputStream*** (для входных потоков) и ***FilterOutputStream*** (для выходных потоков). Эти классы унаследованы от основных базовых классов ввода/вывода - InputStream и OutputStream соответственно. Конструкторы этих классов принимают в качестве параметра объект InputStream и имеют модификатор доступа protected. Сами же эти классы являются базовыми для надстроек. Поэтому только наследники могут вызывать его(при их создании), передавая переданный им поток. Таким образом обеспечивается некоторый общий интерфейс для надстраиваемых объектов.

Классом-фильтром являются классы ***BuffereInputStream*** и ***BufferedOutputStream***. На практике, при считывании с внешних устройств, ввод данных почти всегда необходимо буферизировать. ***BufferedInputStream*** - содержит в себе массив байт, который служит буфером для считываемых данных. То есть, когда байты из потока считываются (вызов метода read()) либо пропускаются (метод skip()), сначала перезаписывается этот буферный массив, при этом считываются сразу много байт за раз. Так же класс

BufferedInputStream добавляет поддержку методов ***mark()*** и ***reset()***. Эти методы определены еще в классе InputStream, но их реализация по умолчанию бросает исключение IOException. Метод mark() запоминает точку во входном потоке и метод reset() приводит к тому, что все байты, считанные после наиболее позднего вызова метода mark(), будут считаны заново, прежде чем новые байты будут считываться из содержащегося входного потока.

BufferedOutputStream - при использовании объекта этого класса, запись производится без необходимости обращения к устройству ввода/вывода при записи каждого байта. Сначала данные записываются во внутренний буфер. Непосредственное обращение к устройству вывода и, соответственно, запись в него произойдет, когда буфер будет полностью заполнен. Освобождение буфера с записью байт на устройство вывода можно обеспечить и непосредственно - вызовом метода flush(). Так же буфер будет освобожден непосредственно перед закрытием потока (вызов метода close()). При вызове этого метода также будет закрыт и поток, над которым буфер надстроен.

```
String fileName ="d:\\file1";
InputStream inStream =null;
OutputStream outStream =null; //Записать в файл некоторое количество байт

long timeStart =System.currentTimeMillis();

outStream =new FileOutputStream(fileName);
outStream =new BufferedOutputStream(outStream);

for(int i=1000000;--i>=0;){
outStream.write(i);}

inStream =new FileInputStream(fileName);
inStream =new BufferedInputStream(inStream);

while(inStream.read()!=-1) {…//чтение данных
}
```

Символьные потоки

Символьные потоки определены в двух иерархиях классов. Наверху этой иерархии два абстрактных класса: Reader и Writer. Они обрабатывают потоки символов Unicode. Абстрактные классы Reader и Writer определяют несколько ключевых методов, которые реализуются другими поточными классами. Два самых важных метода — ***read()*** и ***write()***, которые читают и записывают символы данных, соответственно. Они переопределяются производными поточными классами.

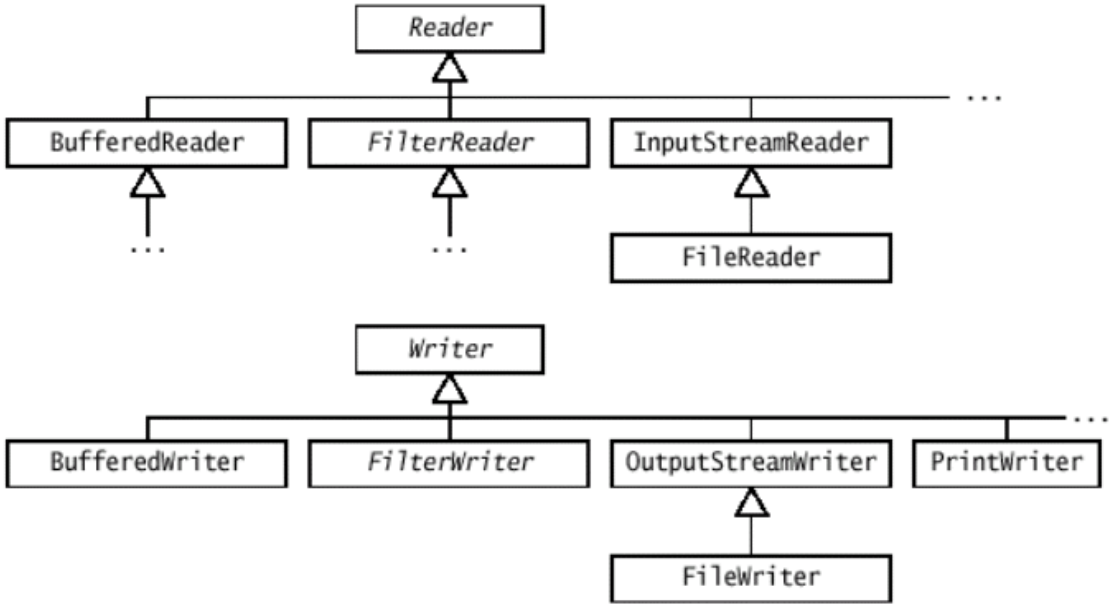


Рисунок 2. Иерархия символьных потоков

До сих пор речь шла только о считывании и записи в поток данных в виде ***byte***. Для работы с другими примитивными типами данных java, определены интерфейсы ***DataInput*** и ***DataOutput***, и существующие их реализации - классы-фильтры ***DataInputStream*** и ***DataOutputStream***.

Интерфейсы DataInput и DataOutput определяют, а классы DataInputStream и DataOutputStream, соответственно реализуют, методы считывания и записи всех примитивных типов данных. При этом происходит конвертация этих данных в ***byte*** и обратно. При этом в поток будут записаны байты, а ответственность за восстановление данных лежит только на разработчике - нужно считывать данные в виде тех же типов, в той же последовательности, как и производилась запись. То есть, можно, конечно,

записать несколько раз int или long, а потом считывать их как short или что-нибудь еще - считывание произойдет корректно и никаких предупреждений о возникшей ошибке не возникнет, но результат будет соответствующий - значения, которые никогда не записывались.

- Запись производится методом ***writeXxx()***, где Xxx – тип данных – int, long, double, byte.
- Для считывания необходимо соблюдать и применять те операторы ***readXxx()***, в котором были записаны типы данных с помощью writeXxx().

Интерфейсы DataInput и DataOutput представляют возможность записи/считывания данных ***примитивных*** типов Java. Для аналогичной работы с объектами определены унаследованные от них интерфейсы ObjectInput и ObjectOutput соответственно.

Основные символьные классы приведены в таблице.

Поточный класс	Значение
BufferedReader	Буферизированный символьный поток ввода
BufferedWriter	Буферизированный символьный поток вывода
CharArrayReader	Поток ввода, которой читает из символьного массива
CharArrayWrite	Выходной поток, который записывает в символьный массив
FileReader	Поток ввода, который читает из файла
FileWriter	Выходной поток, который записывает в файл
FilterReader	Отфильтрованный поток ввода
FilterWriter	Отфильтрованный поток вывода
InputstreamReader	Поток ввода, который переводит байты в символы
LineNumberReader	Поток ввода, который считает строки
OutputStreamWriter	Поток ввода, который переводит символы в байты
PipedReader	Канал ввода
PipedWriter	Канал вывода
PrintWriter	Поток вывода, который поддерживает print() и println()
PushbackReader	Поток ввода, возвращающий символы в поток ввода
Reader	Абстрактный класс, который описывает символьный поток ввода
StringReader	Поток ввода, который читает из строки
StringWriter	Поток вывода, который записывает в строку
Writer	Абстрактный класс, который описывает символьный поток вывода

Пример на запись и считывание символьного потока.

```
String fileName = "d:\\file.txt";
FileWriter fw = null;
BufferedWriter bw = null;
FileReader fr = null;
BufferedReader br = null;

//Строка, которая будет записана в файл
String data = "Some data to be written and readed \n";

try{
fw = new FileWriter(fileName);
bw = new BufferedWriter(fw);
System.out.println("Write some data to file: " + fileName);
// Несколько раз записать строку
for(int i=(int)(Math.random()*10);--i>=0;)bw.write(data);
bw.close();

fr = new FileReader(fileName);
br = new BufferedReader(fr);

String s = null;
int count = 0;
System.out.println("Read data from file: " + fileName);
// Считывать данные, отображая на экран
while((s=br.readLine())!=null)
System.out.println("row " + ++count + " read:" + s);
br.close();
}catch(Exception e){
e.printStackTrace(); }
```

Консольный ввод-вывод

Все программы Java автоматически импортируют пакет java.lang. Этот пакет определяет класс с именем ***System***, инкапсулирующий некоторые аспекты исполнительной среды Java. Класс System содержит также три предопределенные поточные переменные ***in***, ***out*** и ***err***. Эти поля объявлены в System со спецификаторами public и static.

Объект ***System.out*** называют потоком стандартного вывода. По умолчанию с ним связана консоль.

На объект ***System.in*** ссылаются как на стандартный ввод, который по умолчанию связан с клавиатурой.

К объекту ***System.err*** обращаются как к стандартному потоку ошибок, который по умолчанию также связан с консолью.

Пример считывания символов с консоли.

```
// Использует BufferedReader для чтения символов с  консоли,
import java.io.*;
class BRRead
{
public static void main(String args [ ])
throws IOException {
char c;
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Введите символы, 'q' - для завершения.");

// чтение символов
do {
c = (char) br.read();
System.out.println(c); }
while(c != 'q');
} }
```

Пример считывания строк с консоли.

```
import  java.io..*;
class TinyEdit {
public static void main(String args[];
throws IOException (

// Создать BufferedReader-объект,используя System.in
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String str[] = new String[100];
System.out.println("Введите строки текста.");
System.out.println("Введите 'stop' для завершения.");
for (int i=0; i<100; i++)
{
str[i] = br.readLine();
if(str[i].equals("stop"))
break; }
System.out.println(“\n Вот ваш файл:");
// Вывести строки на экран.
for (int i=0; i<100; i++) {
```

```
if(str[i].equals("stop")) break;
System.out.println(str[i]); }
} }
```

Сериализация

В java имеется стандартный механизм превращения объекта в набор байт - ***сериализации***. Для того, что бы объект мог быть сериализован, он должен реализовать интерфейс ***java.io.Serializable*** (соответствующее объявление должно явно присутствовать в классе объекта или, по правилам наследования, неявно в родительском классе вверх по иерархии). Интерфейс java.io.Serializable не определяет никаких методов. Его присутствие только определяет, что объекты этого класса разрешено сериализовывать.

После того, как объект был сериализован, то есть превращен в последовательность байт, его по этой последовательности можно восстановить, при этом восстановление можно проводить на любой машине (вне зависимости от того, где проводилась сериализация), то есть последовательность байт можно передать на другую машину по сети или любым другим образом, и там провести десериализацию. При этом не имеет значения операционная система под которой запущена Java - например, можно создать объект на машине с ОС Windows, превратить его в последовательность байт, после чего передать их по сети на машину с ОС Unix, где восстановить тот же объект.

Для работы по сериализации в java.io определены интерфейсы ***ObjectInput***, ***ObjectOutput*** и реализующие их классы ***ObjectInputStream*** и ***ObjectOutputStream*** соответственно. Для сериализации объекта нужен выходной поток OutputStream, который следует передать при конструировании ObjectOutputStream. После чего вызовом метода writeObject() сериализовать объект и записать его в выходной поток. Например:

```
// сериализация объекта Integer(1)
ByteArrayOutputStream os =new ByteArrayOutputStream();
```

```
Object objSave =new Integer(1);
ObjectOutputStream oos =new ObjectOutputStream(os);
oos.writeObject(objSave);
```

Что бы просмотреть, во что превратился объект objSave, можно просмотреть содержимое массива

byte[] bArray = os.toByteArray();

А чтобы получить этот объект, можно десериализовать его из этого массива:

```
byte[]  bArray =os.toByteArray(); //получение содержимого массива
ByteArrayInputStream is =new ByteArrayInputStream(bArray);
ObjectInputStream ois =new ObjectInputStream(is);
Object objRead =ois.readObject();
```

Сериализация объекта заключается в сохранении и восстановлении состояния объекта. Состояние описывается значением полей. Причем не только описанных в классе, но и унаследованных. При попытке самостоятельно написать механизм восстановления возникли бы следующие проблемы:

- Как установить значения полей, тип которых private
- Объект создается с помощью вызова конструктора. Как восстановить ссылку в этом случае
- Даже если существуют set-методы для полей, как выбрать значения и параметры.

Сериализуемый объект может хранить ссылки на другие объекты, которые в свою очередь так же могут хранить ссылки на другие объекты. И все они тоже должны быть восстановлены при десериализации. При этом, важно, что если несколько ссылок указывают на один и тот же объект, то в восстановленных объектах эти ссылки так же указывали на один и тот же объект. Если класс содержит в качестве полей другие объекты, то эти объекты так же будут сериализовываться и поэтому тоже должны быть сериализуемы. В свою очередь, сериализуемы должны быть и все объекты, содержащиеся в этих сериализуемых объектах и т.д. Полный путь ссылок объекта по всем объектным ссылкам, имеющимся у него и у всех объектов на

которые у него имеются ссылки, и т.д. - называется графом исходного объекта.

Однако, вопрос, на который следует обратить внимание - что происходит с состоянием объекта, унаследованным от суперкласса. Ведь состояние объекта определяется не только значениями полей, определенными в нем самом, но так же и таковыми, унаследованными от супепркласса. Сериализуемый подтип берет на себя такую ответственность, но только в том случае, если у суперкласса определен конструктор по умолчанию, объявленный с модификатором доступа таким, что будет доступен для вызова из рассматриваемого наследника. Этот конструктор будет вызван при десериализации. В противном случае, во время выполнения будет брошено исключение java.io.InvalidClassException.

В процессе десериализации, поля НЕ сериализуемых классов (родительских классов, НЕ реализующих интерфейс Serializable) инициируются вызовом конструктора без параметров. Такой конструктор должен быть доступен из сериализуемого их подкласса. Поля сериализуемого класса будут восстановлены из потока.

Пример сериализации и десериализации объекта.

```
public class Parent{
  public String firstname, lastname;
    public parent(){ firstname='old_first';   lastname='old_last';}
}
public class Child extends Parent implements Serializable{
    private int age;
    public Child (int age){ this.age=age;}
}….
FileOutputStream fos=new FileOutputStream("output.bin");
ObjectOutputStream oos=new ObjectOutputStream(fos);
Child c= new Child(2);
oos.writeObject(c); oos.close();

FileInputStream fis=new FileInputStream("output.bin");
ObjectInputStream ois=new ObjectInputStream(fis);
Ois.readObject();
ois.close();
```

4. Порядок выполнения работы

- изучить предлагаемый теоретический материал;
- реализуйте в виде программ на языке Java следующие задачи:

1. осуществить запись в файл большого произвольного массива данных, используя бинарные потоки, двумя способами: запись **без** и запись **с** использованием классов-фильтров, позволяющих выполнить буферизацию. Произвести замеры времени выполнения записи в файл и сравнить полученное ускорение. Указание: для отсчета времени в миллисекундах необходимо использовать системную функцию следующим образом: long timeStart =System.currentTimeMillis(). То же самое проделать для операции считывания данных из файла.
2. Расширить программу (апплет, на котором размещены кнопки для создания экземпляров Rectangle, ColoredRect, DrawableRect и реализована функция по перетаскиванию объектов мышью), созданную на прошлом лабораторном занятии, следующим образом. Добавить на апплет 2 кнопки – LoadFromFile и SaveToFile, одна из которых сериализует созданные объекты прямоугольников и сохраняет их в файл, другая – считывает из файла и продолжает работу программы с того момента, когда была осуществлена сериализация. Если программа была закрыта сразу после сериализации, то при повторном запуске программы и десериализации объектов, объекты должны располагаться на том же месте (обладать тем же состоянием), что и до закрытия программы. Программа должна позволять неограниченное число раз считывать объекты из файла, не удаляя при этом уже существующие. (**SaveRect/Saverect**)

5. Содержание отчета

В отчете следует указать:

1. цель работы;
2. введение;
3. программно-аппаратные средства, используемые при выполнении работы;
4. основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения лабораторного практикума;
5. заключение (описание результатов и выводы);
6. список используемой литературы.

6. Литература

1. Вязовик, Н.А. Программирование на Java. [электронный ресурс] http://www.intuit.ru/department/pl/javapl (7.01.2012).
2. Хорстманн К.С., Корнелл Г. Библиотека профессионала. JAVA 2. Том 1. Основы. 8-е издание. Пер. с англ. – М.: ООО Издательский дом “Вильямс”, 2008. – 816 с.
3. Эккель Б. Философия JAVA. 4-е издание. – СПб.: Питер, 2009. – 638с.
4. Информационные материалы с официального сайт а разработчиков http://www.oracle.com/technetwork/java/javase/downloads/index.html [электронный ресурс] (7.01.2012).

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра ЮНЕСКО по новым информационным технологиям

С.Н. Карабцев

ПРОГРАММИРОВАНИЕ НА JAVA

Лабораторный практикум

Математический факультет

специальность 010300.62 – фундаментальная информатика и информационные технологии;
специальность 010400.62 – прикладная математика и информатика;
специальность 010500.62 – математическое обеспечение и администрирование информационных систем.

Кемерово, 2012

ЛАБОРАТОРНАЯ РАБОТА №6.
ПАКЕТ javax.swing. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС

1. Цель работы

Целью работы является ознакомление со средствами построения графического интерфейса пользователя GUI с помощью компонент пакетов java.awt и javax.swing.

2. Методические указания

Лабораторная работа направлена на приобретение основных понятий и навыков работы с пакетами java.awt и javax.swing и их компонентами для построения графического интерфейса пользователя. В работе рассматриваются вопросы создания многооконных графических инетрфейсов.

Требования к результатам выполнения лабораторного практикума:

- при выполнении задания необходимо сопровождать все реализованные процедуры и функции набором тестовых входных и выходных данных и описаниями к ним;
- реализацию программы можно осуществлять как из интегрированной среды разработки Eclipse, так и в блокноте с применением вызовов функций командной строки;
- по завершении выполнения задания составить отчет о проделанной работе.

При составлении и оформлении отчета следует придерживаться рекомендаций, представленных на следующей странице: http://unesco.kemsu.ru/student/rule/rule.html.

3. Теоретический материал

Для создания графического интерфейса пользователя используются классы пакетов java.awt и javax.swing.

Создание элементов пользовательского интерфейса библиотека AWT (Abstract Window Toolkit) поручала встроенным инструментальным средствам. Если нужно вывести окно с текстом, то фактически оно отображалось с помощью платформенно-ориентированных средств. Теоретически такие программы должны работать на любых платформах, имея внешний вид характерный для платформы. Однако с помощью AWT трудно создавать переносимые графические библиотеки, зависящие от родных интерфейсных элементов платформы (например, меню и скроллинг на разных платформах могут вести себя по-разному).

Абстрактный Оконный Инструментарий Java 1.0 вводил GUI, который выглядел достаточно заурядно на всех платформах. Кроме того, он был ограничен: можно было использовать только четыре шрифта и было невозможно получить доступ к любому более сложному и тонкому GUI элементу, имеющемуся в ОС. Модель программирования Java 1.0 AWT также была слабая и не объектно-ориентированная.

В 1996 году Netscape создала библиотеку программ для создания GUI и назвала ее Internet Foundation Class (IFC). Элементы пользовательского интерфейса рисовались в пустом окне. Единственно, что требовалось от оконной системы платформы, - отображение окна и рисование в нем. Таким образом, элементы GUI выглядели и вели себя одинаково, но не зависели от платформы, на которой запускались. Компании Sun и Netscape объединили свои усилия и создали библиотеку Swing. Преимущества Swing:

- содержит более богатый и удобный набор элементов пользовательского интерфейса;
- библиотека намного меньше зависит от платформы (меньше ошибок);
- обеспечивает пользователям однотипные средства для работы на разных платформах

Создание фрейма

Окно верхнего уровня в языке ява называется фреймом. В библиотеке awt для такого окна предусмотрен класс Frame. В библиотеке Swing для такого окна предусмотрен класс JFrame (аналог). Класс JFrame расширяет класс Frame и представляет собой один из немногих компонентов в библиотеке Swing, которые не отображаются на холсте (canvas). Кнопки, строка заголовков, пиктограммы и другие компоненты реализуются с помощью пользовательской оконной системы, а не библиотекой Swing.

Рассмотрим самые распространенные приемы работы с классом JFrame.

```
import javax.swing.*;
public class SimpleFrameTest{
  public static void main(String[] args){
   simple frame=new simple();  // создание экземпляра фрейма
// здесь можно создать несколько фреймов или включить их в описание
// класса SimpleFrameTest как поля

   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true); //делает фрейм видимым }
}

/* в данном классе можно реализовывать различную функциональность окна.
Например, разместить кнопки, списки, меню.
*/
class simple extends JFrame{
  public simple(){
    setSize(200,300); // задание размера окна}
}
```

По умолчанию фрейм имеет размер 0x0 пикселей. Для задания размера фрейма используется метод **setSize**(int x, int y). Метод **setDefaultCloseOperation** определяет поведение фрейма при закрытии окна. Задание константы JFrame.EXIT_ON_CLOSE определяет закрытие приложения при закрытии фрейма.

Для задания позиционирования фрейма используется метод **setLocation** (int x, int y), где x и y – координаты левого верхнего угла фрейма.

Разработчики Java для запуска приложения с фреймом рекомендуют использовать диспетчер событий. Тогда предыдущий пример будет выглядеть следующим образом.

```
Import javax.swing.*;
public class SimpleFrameTest{
  public static void main(String[] args){
  EventQueue.invokeLater (new Runnable() {
    public void run()
    {
      simple frame=new simple();  // создание экземпляра фрейма
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true); //делает фрейм видимым
   }
  });
}}

class simple extends JFrame{
  public simple(){
    setSize(200,300); // задание размера окна}
}
```

Метод **setIconImage**() сообщает оконной системе о том, какая пиктограмма должна отображаться в строке заголовка.

Метод **setTitle**() позволяет изменить текст в окне заголовка.

Метод **setResizible**(), получающий в качестве параметра логическое значение и определяющий, имеет ли пользователь право изменять размеры фрейма.

Определение подходящего размера фрейма можно выполнить с помощью методов класса Toolkit.

```
import javax.swing.*;
public class SimpleFrameTest{
  public static void main(String[] args){
  EventQueue.invokeLater (new Runnable() {
    public void run()
    {
      SizedFrame frame=new SizedFrame();  // создание экземпляра фрейма
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true); //делает фрейм видимым
```

```
   }
  });
}}

class SizedFrame extends JFrame{
  public SizedFrame(){
  //класс взаимодействия с оконной системой ОС
  Toolkit kit = Toolkit.getDefaultToolkit();
  Dimension screens = kit.getScreenSize();
  int w,h;
  w = screens.width;
  h = screens.height;
  setSize(w/2,h/2);
  setLocation(w/4, h/4);
  setTitle("My Frame");
  Image img = kit.getImage(“Icon.gif”);
  setIconImage(img);
  }
}
```

Многооконные приложения

Как известно, окно в современных графических операционных системах используются для разделения задач, выполняемых пользователем, в одном из окон он может редактировать текст, в другом окне просматривать фотографии. Использование окон позволяет ему временно забыть про остальные свои задачи, не смешивая их, заниматься основным делом (в терминах оконной системы это будет активное в данный момент окно), и, как только возникает необходимость, переместиться к другой задаче (сделав активным другое окно, то есть вытащив его на первый план и временно убрав на задний план все остальное).

Как только концепция окон была изобретена и довольно успешно начала справляться с разделением задач пользователя на основную и второстепенные, возникло желание продвинуться немного дальше и применить ее уже внутри одного приложение, то есть разделить задачу внутри окна на несколько задач используя все те же окна, но на этот раз уже внутри другого, главного, окна. Подобные интерфейсы и называются

многодокументными. Приложения, берущие на вооружение такие интерфейсы, как правило, способны выполнять задачи такого широкого спектра, что уместить все в одно окно становится затруднительно без ущемления удобства пользователя. На помощь приходят дополнительные окна, в которых можно разместить информацию, которая имеет косвенное отношение к основной, выполняемой в данный момент, задаче.

Применение многодокументных интерфейсов на практике вызывает жаркие споры. Сам по себе интерфейс операционной системы является многодокументным приложением, и встраивание в него еще одного подобного интерфейса может запутывать пользователя. Окна операционной системы работают по своему, окна приложения по своему, и все это не прибавляет уверенности пользователю в том, что происходит и как этим следует управлять. Поэтому многие развитые многодокументные приложения постарались уменьшить количество и разнообразие дополнительных окон, чтобы уменьшить сложность для пользователя. Тем не менее, как показывает практика, полностью избавиться от дополнительных окон в сложных приложениях затруднительно, так что мы все-таки изучим возможности Swing в данном вопросе.

Создание внутренних окон

Внутренние окна в Swing реализованы классом **JInternalFrame**. Данный класс унаследован от общего предка всех компонентов Swing JComponent, и, таким образом, представляет собой обычный легковесный компонент, который вы при желании можете добавить в любую панель или окно вашего приложения. Так как данный компонент призван эмулировать окно внутри приложения, у него есть элементы управления, такие как меню окна, кнопки для закрытия, развертывания и свертывания, а также заголовок и рамка. Роль рабочего стола, на котором окна размещаются, внутри которого перемещаются и занимают его пространство, исполняет компонент **JDesktopPane**.

```
import javax.swing.*;

public class SimpleMDI extends JFrame {
   public SimpleMDI() {
      super("SimpleMDI");
      setSize(400, 300);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      // создаем рабочий стол Swing
      JDesktopPane desktopPane = new JDesktopPane();
      // добавляем его в центр окна
      add(desktopPane);
      // создаем несколько внутренних окон, применяя доступные
конструкторы
      JInternalFrame frame1 = new JInternalFrame("Frame1", true);
      JInternalFrame frame2 = new JInternalFrame(
"Frame2", true, true, true, true);
      // добавляем внутренние окна на рабочий стол
      desktopPane.add(frame1);
      desktopPane.add(frame2);
      // задаем размеры и расположения, делаем окна видимыми
      frame1.setSize(200, 100);
      frame1.setLocation(80, 100);
      frame1.setVisible(true);
      frame2.setSize(200, 60);
      frame2.setVisible(true);
      // выводим окно на экран
      setVisible(true);
   }

   public static void main(String[] args) {
      new SimpleMDI();
   }
}
```

В обычном окне JFrame размещается рабочий стол JDesktopPane. Как видно, никаких параметров конструктору не требуется, так же как не требуется для нормальной работы и дополнительной настройки. О рабочем столе можно думать как о специализированной панели, специально предназначенной для размещения и управления внутренними окнами. Добавляем его в центр окна методом **add**() напрямую.

Далее создаются внутренние окна. Интересно, что класс JInternalFrame обладает целым набором конструкторов с различным набором параметров, которые позволяют задать все атрибуты внутренних окон. Обязательно присутствует лишь заголовок окна (как первый параметр всех конструкторов).

Первое окно создается с заголовком и изменяемого размера, за что отвечает второй параметр конструктора. По умолчанию, если не указывать **true**, внутренние окна считаются неизменяемого размера, без возможности сворачивания, развертывания, более того, их даже невозможно закрыть. Все эти возможности включаются либо передачей параметров в конструктор, либо меняются через соответствующие свойства.

Второе окно создается вызовом наиболее развернутого конструктора. Здесь мы указываем заголовок окна, а также делаем его размер изменяемым, для самого окна включаем поддержку развертывания, свертывания и способность закрываться по кнопке закрытия окна. Довольно своеобразное решение создателей не включать все эти свойства по умолчанию, и передавать их в довольно длинный конструктор.

Внутренние окна стараются во всем походить на своих старших братьев – окна высокого уровня, унаследованные от Window, и поэтому они по умолчанию невидимы, имеют нулевой размер и располагаются в начале экрана. Работа с внутренними окнами здесь не отличается от работы с обычными. Мы меняем размер окон, устанавливаем их позиции и делаем их видимыми. Только после этого они появятся на рабочем столе JDesktopPane.

Запустив приложение, вы увидите рабочий стол и располагающиеся на нем внутренние окна. Отличие от обычной панели лишь в том, что мы использовали для расположения окон все те же методы, что для окон, и в том, что вы можете таскать окна, сворачивать их и менять их размеры, если, конечно, они вам это позволяют. В дополнение ко всему, рабочий стол позаботится о том, чтобы активное окно перекрывало все остальные и было на переднем плане.

На этом можно считать многодокументное приложение Swing готовым. Вы можете конструировать свои интерфейсы и добавлять их во внутренние окна, располагая их, так как вас удобно. Пользователь будет в состоянии переключать свое внимание между окнами, концентрируясь на определенном аспекте приложения. Пример создания многодокументного приложения с компонентом **меню** приведено в **Examples/internalframedemo.java**.

Создание двух и более окон в приложении

При разработке интерфейса можно не создавать внутренние окна, а воспользоваться классом JFrame. В примере, показанном ниже, создается основное окно приложения, на котором размещена кнопка. Данное окно является экземпляром класса simple1. Нажатие кнопки but на фрейме frame (класса simple1 в методе main) ведет к вызову конструктора для создания второго фрейма класса simple2. Обратите внимание, что экземпляр класса simple2 включен как поле в класс simple1. Это поле называется s2. При таком подходе вся информация, доступная на фрейме s2, будет доступна и основному классу. Это демонстрируется путем вызова метода getV(), который распечатывает на экран значение переменной r, являющейся полем в классе simple2.

```
//фрейм посередине окна
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class t2 {
  public static void main(String[] args){
    simple1 frame=new simple1();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.show();}
}

class simple1 extends JFrame{
simple2 s2;
JButton but=new JButton("Create");
  public simple1(){
```

```
  ButtonListener blistener = new ButtonListener();

  Toolkit kit = Toolkit.getDefaultToolkit();
  Dimension screens = kit.getScreenSize();
  int w,h;
  w = screens.width;
  h = screens.height;
  setSize(w/2,h/2);
  setLocation(w/4, h/4);
  setTitle("My Frame");
  setLayout(new FlowLayout());
  setLayout(null);  //отключает менеджер расположения компонентов
  add(but);
  but.addActionListener(blistener);
}

public void getV(){
System.out.println("VALUES+++"+s2.r);
}

class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
    if(s2==null){s2=new simple2(); }
        s2.show();
        getV();                                          }
                                                           }
}//class

class simple2 extends JFrame{
int r=5;

  public simple2(){
  Toolkit kit = Toolkit.getDefaultToolkit();
  Dimension screens = kit.getScreenSize();
  int w,h;
  w = screens.width;
  h = screens.height;
  setSize(w/3,h/3);
  setLocation(w/6, h/6);
  setTitle("My Frame");
}
}
```

Управление компоновкой

Все компоненты, с которыми мы работали до сих пор, размещались "вручную" вызовом метода add() и установкой позиции с помощью метода setLocation() или setBounds(). Вызов метода setLayout(null) запрещал использование предусмотренного по умолчанию механизма управления размещением компонентов. В языке Java реализована концепция динамической компоновки - все компоненты контейнера управляются менеджером компоновки (layout manager).

Существуют следующие виды менеджеров компоновки – потоковая компоновка (flow layout manager), компоновка рамок (border layout), сеточная компоновка (grid layout) и другие более сложные менеджеры.

Для установки компоненту менеджера компоновки используется метод **setLayout**(LayoutManager m). Данный метод необходимо вызывать в объектах классов JApplet, Applet, JFrame, Frame, JPanel и других контейнерах перед размещением на них графических компонентов построения интерфейса, таких как кнопки (JButton, Button), варианты выбора (JCheckBox), текстовые поля (TextArea, JTextArea) и др.

В качестве аргумента при вызове метода **setLayout**(LayoutManager m) указывается конкретный экземпляр класса **m** менеджера компоновки.

Менеджер FlowLayout

Класс FlowLayout реализует простой стиль размещения, при котором компоненты располагаются, начиная с левого верхнего угла, слева направо и сверху вниз. Если в данную строку не помещается очередной компонент, он располагается в левой позиции новой строки. Справа, слева, сверху и снизу компоненты отделяются друг от друга небольшими промежутками. Ширину этого промежутка можно задать в конструкторе FlowLayout. Каждая строка с компонентами выравнивается по левому или правому краю, либо центрируется в зависимости от того, какая из констант LEFT, RIGHT или

CENTER была передана конструктору. Режим выравнивания по умолчанию - CENTER, используемая по умолчанию ширина промежутка - 5 пикселей.

Конструкторы:

FlowLayout();

FlowLayout(int align);

FlowLayout(int align, int hgap, int vgap);

Примеры программ с использованием FlowLayout приведены в папке Examples/ex1 и ex2.

Менеджер BorderLayout

Класс BorderLayout реализует обычный стиль размещения для окон верхнего уровня, в котором предусмотрено четыре узких компонента фиксированной ширины по краям, и одна большая область в центре, которая может расширяться и сужаться в двух направлениях, занимая все свободное пространство окна. У каждой из этих областей есть строки-имена: String.North, String.South, String.East и String.West соответствуют четырем краям, а Center - центральной области.

Для вызова данного менеджера необходимо использовать следующую конструкцию

```
setLayout(new BorderLayout());
add(yellowButton, BorderLayout.SOUTH);
```

Пример программы с использованием BorderLayout приведен в папке Examples/ex3.

Менеджер GridLayout

Класс GridLayout размещает компоненты в простой равномерной сетке. Конструктор этого класса позволяет задавать количество строк и столбцов. Например, setLayout(new GridLayout(5,4)) – указывает желаемое количество строк (5) и столбцов (4). А setLayout(new GridLayout(5,4,3,3)) – указывает

желаемое количество строк (5) и столбцов (4) и расстояние между компонентами по горизонтали (3) и вертикали (3).

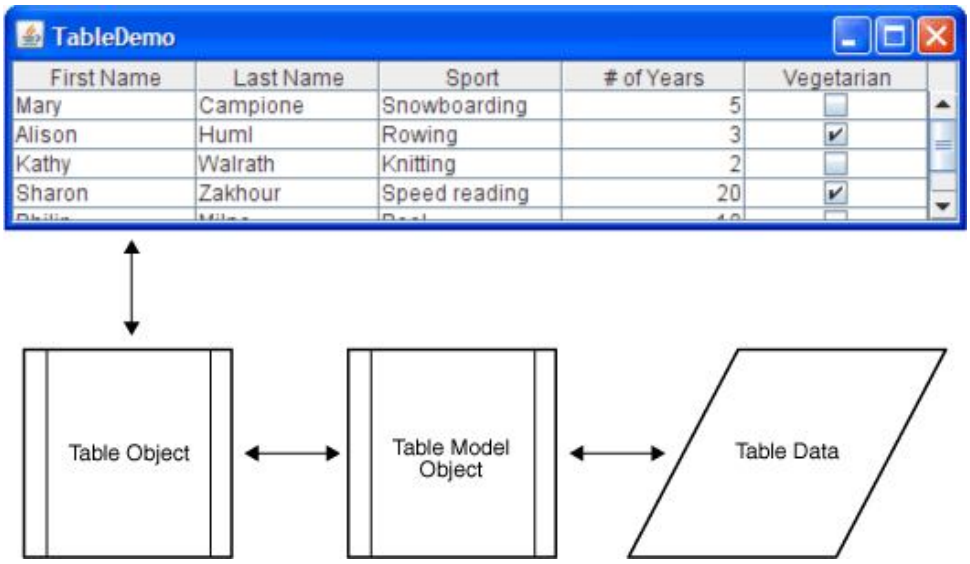
Класс JTable

Класс javax.swing.JTable используется для отображения и редактирования регулярных плоских таблиц. Класс JTable содержит следующие конструкторы:

- public JTable() – создание таблицы по умолчанию, со столбцами и строками по умолчанию.
- public JTable(TableModel dm) – столбцы и строки таблицы инициализируются моделью данных dm.
- public JTable(int numRows, int numColumns) – создание пустой таблицы размером (numRows, numColumns), используя DefaultTableModel. Название столбцов – A, B, C…
- JTable(Object[][] rowData, Object[] columnNames), где rowData – массив данных по строкам, а columnNames – массив с названиями столбцов.
- JTable(TableModel dm, TableColumnModel cm).
- JTable(TableModel dm, TableColumnModel cm, ListSelectionModel sm).
- public JTable(Vector rowData, Vector columnNames) – создает таблицу и показывает в ней данные из rowdata, которые есть Vector of Vectors. Название столбцов – вектор columnNames. ((Vector)rowData.elementAt(1)).elementAt(5); - доступ к элементу (1,5)

Для создания таблицы, которая может отображать ***актуальные*** данные, заголовки столбцов (не дефолтные) и позволяет выполнять над собой определенный набор действий (перетаскивание столбцов, выделение и форматирование ячеек и т.д.) необходимо реализовывать интерфейс **TableModel** (табличная модель) и передавать объект класса, реализующего

интерфейс табличной модели, в конструктор. На рисунке ниже показана взаимосвязь между объектом Таблица (Table Object), объектом, реализующим интерфейс табличной модели (Table Model Object) и данными, которые отображаются в таблице (Table data).



Если при создании таблицы JTable явно не реализуется интерфейс **TableModel,** то в таблицу передается объект интерфейса по умолчанию DefaultTableModel. Чтобы создать таблицу, подчиняющуюся табличной модели, необходимо:

```
TableModel myData = new MyTableModel();
JTable table = new JTable(myData);
```

При этом класс MyTableModel должен реализовывать интерфейс (т.е. перекрывать все его методы) TableModel.

Интерфейс **TableModel** содержит большое количество методов, которые управляют поведением таблицы. Поэтому на практике стараются реализовать некоторый класс-потомок интерфейса TableModel (чтобы не все методы интерфейса перекрывать). Например,

public abstract class ***AbstractTableModel*** extends Object implements ***TableModel***, Serializable

Данный абстрактный класс обеспечивает встроенную (по умолчанию) реализацию большинства методов интерфейса TableModel. Для создания табличной модели данных необходимо наследовать подкласс от этого класса

и обязательно реализовать следующие методы интерфейса TableModel: getRowCount(), getColumnCount() и getValueAt(int rowIndex, int columnIndex). Например, чтобы создать таблицу размером 10 на 10, которая хранит значение таблицы умножения необходимо:

```
TableModel dataModel = new AbstractTableModel() {
        public int getColumnCount() { return 10; } // перекрыли метод, он всегда
                                                   //возвращает 10 столбцов
        public int getRowCount() { return 10;} // всегда возвращает 20 строк

//сами значения, которые помещаются в таблицу! Обратите внимание, что
//метод называется getValueAt, а не setValueAt
        public Object getValueAt(int row, int col) { return new Integer(row*col); }
    };

    JTable table = new JTable(dataModel);
    JScrollPane scrollpane = new JScrollPane(table);
```

Тот же самый пример можно написать более явно (но длинно) следующим образом:

```
class MyTableModel extends AbstractTableModel {
        public int getColumnCount() { return 10; } // перекрыли метод, он всегда
                                                   //возвращает 10 столбцов
        public int getRowCount() { return 10;} // всегда возвращает 20 строк

//сами значения, которые помещаются в таблицу! Обратите внимание, что
//метод называется getValueAt, а не setValueAt
        public Object getValueAt(int row, int col) { return new Integer(row*col); }
    }

   MyTableModel dataModel = new MyTableModel();
    JTable table = new JTable(dataModel);
    JScrollPane scrollpane = new JScrollPane(table);
```

Вообще говоря, интерфейс TableModel содержит следующие методы:

- public int ***getRowCount***() – возвращает количество строк в табличной модели.

- public int ***getColumnCount***() – возвращает количество столбцов в табличной модели.
- public String ***getColumnName***(int columnIndex) – возвращает название столбца в модели.
- public Object ***getValueAt***(int rowIndex, int columnIndex) – возвращает значение в ячейке с номером (rowIndex, columnIndex).
- public void ***setValueAt***(Object aValue, int rowIndex, int columnIndex) – устанавливает значение aValue в ячейку с номером (rowIndex, columnIndex).

Более полную информацию с примерами о классе JTable можно найти на сайте создателя http://java.sun.com/docs/books/tutorial/uiswing/components/table.html.

4. Порядок выполнения работы

- изучить предлагаемый теоретический материал;
- реализуйте в виде программ на языке Java следующие задачи:

3. Создайте Frame с компонентом меню Файл -> Создать. При нажатии на элемент меню «Создать» создается новое окно, на котором размещен компонент JRadioButton и кнопка JButton. Компонент JRadioButton состоит из 2 элементов – «отображать таблицу умножения» и «отображать таблицу сложения». При выборе конкретного пункта (умножения или сложения) и нажатии кнопки JButton окно закрывается, а на основном фрейме появляется таблица умножения или сложения соответственно.

4. Окно, которое появляется при нажатии на «Файл -> Создать» можно создать по своему усмотрению одним из следующих двух способов:
   - с помощью классов JDesktopFrame и JInternalFrame (за основу можно взять файл **Examples/internalframedemo.java**).
   - с помощью классов Frame.

5. Содержание отчета

В отчете следует указать:

1. цель работы;
2. введение;
3. программно-аппаратные средства, используемые при выполнении работы;
4. основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения лабораторного практикума;
5. заключение (описание результатов и выводы);
6. список используемой литературы.

6. Литература

1. Вязовик, Н.А. Программирование на Java. [электронный ресурс] http://www.intuit.ru/department/pl/javapl (7.01.2012).
2. Хорстманн К.С., Корнелл Г. Библиотека профессионала. JAVA 2. Том 1. Основы. 8-е издание. Пер. с англ. – М.: ООО Издательский дом “Вильямс”, 2008. – 816 с.
3. Эккель Б. Философия JAVA. 4-е издание. – СПб.: Питер, 2009. – 638с.
4. Информационные материалы с официального сайт а разработчиков http://www.oracle.com/technetwork/java/javase/downloads/index.html [электронный ресурс] (7.01.2012).

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра ЮНЕСКО по новым информационным технологиям

С.Н. Карабцев

ПРОГРАММИРОВАНИЕ НА JAVA

Лабораторный практикум

Математический факультет

специальность 010300.62 – фундаментальная информатика и информационные технологии;
специальность 010400.62 – прикладная математика и информатика;
специальность 010500.62 – математическое обеспечение и администрирование информационных систем.

Кемерово, 2012

ЛАБОРАТОРНАЯ РАБОТА №7.
JDBC

1. Цель работы

Получить общее представление о прикладном программном интерфейсе JDBC для соединения с СУБД.

2. Методические указания

Лабораторная работа направлена на приобретение навыка написания программ на языке Java с применением интерфейса JDBC для подключения к СУБД Oracle.

Требования к результатам выполнения лабораторного практикума:

- при выполнении задания необходимо сопровождать все реализованные процедуры и функции набором тестовых входных и выходных данных и описаниями к ним;
- компиляцию, запуск программ выполнять различными способами;
- по завершении выполнения задания составить отчет о проделанной работе.

При составлении и оформлении отчета следует придерживаться рекомендаций, представленных на следующей странице: http://unesco.kemsu.ru/student/rule/rule.html.

3. Теоретический материал

JDBC – прикладной программный интерфейс (API) для выполнения SQL-запросов. Состоит из множества классов и интерфейсов, написанных на JAVA.

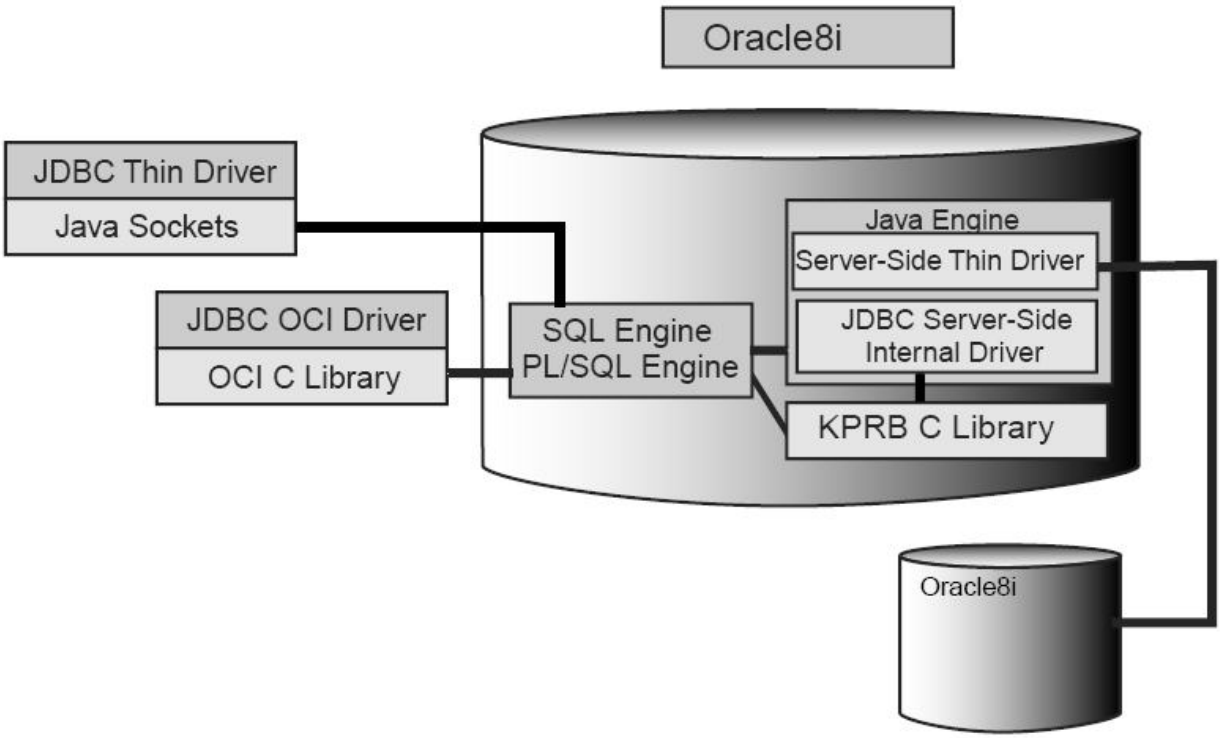
Преимущества:

1. Легкость отсылки запросов на сервер БД

2. Использование JDBC API освобождает от написания приложения для каждой БД
3. Поддержка всех расширений (типов, соединений, классов…) СУБД. Соответствие SQL (но, как всегда, есть исключения)
4. …

JDBC позволяет устанавливать соединения с БД, используя различные типы подключений, отсылать SQL-запросы и обрабатывать результаты. В состав драйверов к СУБД Oracle входит:

- Thin Driver – драйвер для создания клиентских приложений, не требующий установки клиента Oracle.
- OCI Drivers - драйвер для создания клиентских приложений, требующий установки клиента Oracle (OCI 7, 8).
- Server-side Thin Driver – драйвер, функциональность которого как у Thin Driver, но применяется для выполнения кода внутри СУБД. Код может подключаться к удаленной СУБД или реализовывать 3-х звенные приложения.
- Server-side Internal Driver – драйвер, применяющийся при создании приложений внутри СУБД. Исполняет хранимые Java-процедуры и подключается к ядру СУБД, на которой работает.



Thin driver

- Драйвер на 100% написан на Java. Предназначен для апплетов, но годится и для клиентских приложений.
- Драйвер платформонезависимый, не требует клиента СУБД.
- Закачивается браузером и начинает работу вместе с апплетом.
- Драйвер обеспечивает прямое соединение с СУБД через стек TCP/IP путем эмуляции работы библиотеки OCI8 и TTC.
- Со стороны СУБД обязательно должен быть Listener.
- Для работы с этим типом драйвера в браузере д.б. разрешена поддержка Java-сокетов.

OCI driver

- Драйвер написан на Java и Си. Предназначен для создания клиентских приложений.
- Требует установки клиента Oracle и является платформозависимым.
- Драйвер переводит JDBC-вызовы в вызовы OCI. Использует библиотеки OCI8, Net8, Core …

– Предоставляет широкую совместимость с различными версиями СУБД (7,8i,9,10), а также более широкие возможности по работе с СУБД – named pipe …

Последовательность действий для подключения к СУБД Oracle

Для того чтобы подключиться к СУБД и выполнить запрос, необходимо написать код для выполнения ***следующих действий***:

1. Import Packages
2. Register the JDBC Drivers
3. Open a Connection to a Database
4. Create a Statement Object
5. Execute a Query and Return a Result Set Object
6. Process the Result Set
7. Close the Result Set and Statement Objects
8. Make Changes to the Database
9. Commit Changes
10. Close the Connection

Шаг 1.

– import java.sql.*; // стандартный пакет JDBC
– import oracle.jdbc.driver.*; // расширение JDBC для Oracle
– import oracle.sql.*; // особенности языка SQL для Oracle
– Необходимо переменной classpath указать положение драйвера JDBC ([ORACLE_HOME\jdbs\lib\zip-классы]). Сделать это можно 2 способами:
    1. через переменную среды окружения
    2. при компиляции и запуске указать ключ javac -classpath ".; ORACLE_HOME\jdbs\lib\classes12.zip; ORACLE_HOME\jdbs\lib\nls_charset12.zip"

Шаг 2.

Данная операция осуществляется вызовом статического метода:

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

Шаг 3.

Для открытие соединения вызывается метод getConnection() класса DriverManager, возвращающий объект типа Connection.

getConnection(String URL, String user, String password);

Строка URL выглядит следующим образом:

jdbc:oracle:<drivertype>:@<database>.

Например,

DriverManager.getConnection ("jdbc:oracle:thin:@myhost:1521:orcl", "scott", "tiger");

или

DriverManager.getConnection("jdbc:oracle:oci8:@(description=(address=(host=myhost)(protocol=tcp)(port=1521))(connect_data=(sid=orcl)))","scott", "tiger");

Connection conn = DriverManager.getConnection ("jdbc:oracle:oci8:@mics", "stud01", "stud01");

Шаг 4.

Создание объекта Statement для описания запроса используется метод createStatement класса соединения

Statement stmt = conn.createStatement();

Шаг 5.

Для выполнения запроса к БД используется метод executeQuery класса Statement. Полученный результат возвращается в переменную класса ResultSet, которую в дальнейшем необходимо обработать.

ResultSet rset = stmt.executeQuery ("SELECT ename FROM emp");

Шаг 6.

После получения данных в переменную ResultSet необходимо вызывать метод next() для построчного доступа к данным до тех пор, пока не будет достигнут конец данных.

Для извлечения данных используется метод getXXX() класса ResultSet, где XXX – предопределенный тип Java.

```
while (rset.next())
System.out.println (rset.getString(1));
```

Шаг 7.

Необходимо после использования явно закрывать экземпляры типов Statement и ResultSet вызовом метода close().

Драйвер не содержит метод finalizer(), поэтому очистка памяти происходит при вызове close().

Если переменная rset имеет тип ResultSet, а stmt – Statement, то

```
rset.close();
stmt.close();
```

Шаг 8.

Для записи данных в базу через операции Insert или Update используется класс PreparedStatement. Объект данного класса позволяет выполнить выражение с переменным числом входных параметров.

Для подстановки значений в выражение PreparedStatement используется метод setXXX() класса PreparedStatement.

Например,

```
PreparedStatement pstmt =
conn.prepareStatement ("insert into EMP (EMPNO, ENAME) values (?,
?)");
// Add LESLIE as employee number 1500
pstmt.setInt (1, 1500); // The first ? is for EMPNO
```

pstmt.setString (2, "LESLIE"); // The second ? is for ENAME

// Do the insertion

pstmt.execute ();

Шаг 9.

- По умолчанию, операции DML (Insert, Update, Delete) фиксируются автоматически после их выполнения. Для отключения такого режима используется команда

conn.setAutoCommit(false);

- Если автоматический режим отключен, то необходимо вручную выполнять операции commit и rollback:

conn.commit() или conn.rollback()

- Неявный commit всегда срабатывает при разрыве соединения или выполнении функций DDL.

Шаг 10.

- После завершения работы необходимо закрыть соединение

conn.close()

Рассмотрим пример соединения с СУБД Oracle

```
import java.sql.*; import java.io.*; import java.awt.*;
class JdbcTest {
public static void main (String args []) throws SQLException {
// Load Oracle driver
DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());
Connection conn = DriverManager.getConnection
("jdbc:oracle:thin:@myhost:1521:ORCL","scott", "tiger");
// Query the employee names
Statement stmt = conn.createStatement ();
ResultSet rset = stmt.executeQuery ("SELECT ename FROM emp");
```

```
while (rset.next ())
  System.out.println (rset.getString (1));
  //close the result set, statement, and the connection
  rset.close(); stmt.close(); conn.close();
}
}
```

4. Порядок выполнения работы

- изучить предлагаемый теоретический материал;
- создайте следующие программы:

5. Содержание отчета

В отчете следует указать:

1. цель работы;
2. введение;
3. программно-аппаратные средства, используемые при выполнении работы;
4. основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения лабораторного практикума;
5. заключение (описание результатов и выводы);
6. список используемой литературы.

6. Литература

1. Вязовик, Н.А. Программирование на Java. [электронный ресурс] http://www.intuit.ru/department/pl/javapl (7.01.2012).
2. Хорстманн К.С., Корнелл Г. Библиотека профессионала. JAVA 2. Том 1. Основы. 8-е издание. Пер. с англ. – М.: ООО Издательский дом “Вильямс”, 2008. – 816 с.
3. Эккель Б. Философия JAVA. 4-е издание. – СПб.: Питер, 2009. – 638с.
4. Информационные материалы с официального сайт а разработчиков http://www.oracle.com/technetwork/java/javase/downloads/index.html [электронный ресурс] (7.01.2012).

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра ЮНЕСКО по новым информационным технологиям

С.Н. Карабцев

ПРОГРАММИРОВАНИЕ НА JAVA

Лабораторный практикум. Примеры решения творческих работ с применением языка Java

Математический факультет

специальность 010300.62 – фундаментальная информатика и информационные технологии;
специальность 010400.62 – прикладная математика и информатика;
специальность 010500.62 – математическое обеспечение и администрирование информационных систем.

Кемерово, 2012

ПРИМЕРЫ РЕШЕНИЯ ТВОРЧЕСКИХ ЗАДАНИЙ КОНТРОЛЬНЫХ РАБОТ НА ЯЗЫКЕ JAVA

В данной главе приводятся наиболее удачные варианты программ, написанных студентами математического факультета Кемеровского государственного университета, начиная с 2006-2007 учебного года. В данных программах реализованы некоторые объекты реального мира, поведение которых необходимо смоделировать. Такие творческие задания выдавались студентам в качестве контрольных или семестровых работ. Рассмотрение готового кода может быть полезно на ранних стадиях изучения языка Java. Данный код не претендует на звание «эталонного», т.к. реализован не профессиональными программистами на Java, а обучающимися студентами.

Само задание в общем виде звучало следующим образом:
«*необходимо реализовать программу в виде фрейма или апплета, реализующую поведение некоторого объекта. Для примера выберем в качестве объекта моделирования воздушный шар. Объект должен содержать набор методов, отвечающих за его поведение: шар умеет взлетать, делать посадку, ускоряться в случае, когда происходит нагревание шара газом или шар освобождается от балласта. Необходимо проверять согласованность моделируемого объекта, например, шар не может взлететь, если на борт не было взято газа. Шар может разбиться, если посадка произошла слишком быстро. Весь процесс моделирования должен сопровождаться графическими изображениями и управляться либо клавишами клавиатуры, либо кнопками на окне приложения. Поведение*

моделируемого объекта (воздушного шара) должно быть похоже на поведение реального объекта, приветствуется, если в основе лежат физические или математические законы. Например, для взлета шара необходимо рассчитать подъемную силу».

Варианты моделируемых объектов:

1. Воздушный шар
2. Аккумулятор
3. Телевизор
4. Часы
5. Облако
6. Цветок
7. Хищное животное
8. Сотовый телефон
9. Рыба в озере (реке, океане)
10. Пешеход и многое другое.

1. **Моделирование работы перекрестка**.

   В данном апплете реализован класс, имитирующий работу светофора на перекрестке. По перекрестку могут передвигаться автомобили, появление и направление движения которых задаются с помощью кнопок в верхней части апплета. Светофор «умеет» переключаться самостоятельно. Кнопка «UpDownCar» отвечает за появление автомобиля в направлении движения снизу вверх, «DownUpCar» - сверху вниз, «LeftRightCar» - слева направо, и последняя кнопка – справа налево. (**Road\code.zip**).

2. **Самолет**

   В данном примере моделируется полет самолета. Для управления самолетом реализованы слушатели событий, поступающих от клавиатуры. В самолет можно заливать топливо до взлета клавишей «q». Самолету можно задать скорость клавишей «w». Пара клавиш «a»

и «z» управляет увеличением и уменьшением мощности моторов самолета, а клавиши «s» и «x» увеличивают и уменьшают усилие торможения. С помощью клавиш «d» и «c» можно управлять набором высоты. При полете самолет может врезаться в препятствия и падать. Взлет происходит только при наборе определенной скорости. Для начала работы апплета необходимо сначала заправить самолет, а затем включить скорость клавишей «w». После этого управление происходит клавишами разгона, торможения и набора высоты. Значения всех полей класса выводится на апплете.(**Air\code.zip**)

3. **Воздушный шар**

В данной работе моделируется поведение воздушного шара. В основе полета шара заложены упрощенные физические законы (закон Архимеда, а также реализован процесс остывания газа). Шар может взлетать при сжигании газа и повышении температуры газа внутри шара. Шар можно дозаправлять газом. Если шаром неаккуратно управлять, то он взрывается или разбивается. Управление реализовано четырьмя клавишами клавиатуры:

[стрелка вверх] – добавить огня (сжигание пропана)

[стрелка влево] – движение влево

[стрелка вправо] – движение вправо

[space] – заправить болоны с пропаном

Данный пример реализован студентом Швачичем Андреем.

(**Baloon\code.zip**)

4. **Рыба в пруду**

В данной работе моделируется объект «рыба в пруду». Рыба умеет самостоятельно плавать. Доплывая до границ пруда, рыба разворачивается и плывет в противоположном направлении. Пока рыба плавает, она может проголодаться. С помощью клавиш мыши рыбу можно покормить, однако необходимо действовать так, чтобы корм попадал на рыбу. Чем больше рыба есть, тем больших размеров она

становится. С течением времени работы апплета день меняется на ночь, и рыба меняет свой внешний вид. Данная работа является одной из самых ярких студенческих работ (автор – Бабенко Алексей Викторович). Каждое действие в ней сопровождается звуковым сигналом! Поэтому рекомендуется включить колонки.

За рыбой можно охотиться. Для этого в верхней части апплета есть кнопка, вызывающая полет ракеты, и поле для ввода вертикальной координаты выстрела. (**Fish\code.zip**)

5. **Черепаха.**

   В данной работе моделируется поведение черепахи. Модель черепахи была реализована в виде игры. Цель которой заключается в том, что черепаха должна есть рыбок, чтобы не умереть с голоду, но не должна и переедать. Черепаха перемещается за указателем мыши. При перемещении снижается уровень жизни (на 10 при каждой миллисекунде). При нажатии на «пробел» появляется одна основная рыбка-еда, а вторая выбирается случайным образом (или аналогичная, или рыбка-убийца, или никого). Координаты рыбок и их направление выбираются случайным образом. Загружаемая картинка зависит от какого края и в каком направлении будет двигаться рыба. Если черепаха переест или, наоборот, не будет есть до тех пор, пока ее уровень жизни не упадет до нуля, то она погибает. Автор работы – Богомолова Зинаида Викторовна. (**Tortilla\code.zip**)

6. **Кактус**

   В данной работе моделируется жизнь цветка. Модель цветка зависит от следующих параметров:

- возраст
- уровень полива
- времени суток

Возраст в начале работы равен нулю. Уровень полива ноль. Время равно 12 часам, время суток день.

На апплете представлены 2 кнопки. Первая кнопка – это время, при нажатии нее считается, что проходит 3 часа. На часах изменяется время и у цветка отнимается один уровень полива. Уровень полива изменяется от -6 до 6. Уровни -2, -1, 0, 1, 2 полива считаются оптимальными. Уровни -3, -4, 3, 4 считаются опасными. Существует опасность засыхания или перенасыщения водой. Уровни 5, -5 критические. Достигая уровня 6, -6 цветок погибает. С течением времени день и ночь меняются, а также изменяется а также меняются время на часах. При недостаточном уровне полива цветок желтеет, при избыточном уровне увеличивается в размерах. При правильном уровне полива (-2, -1, 0, 1, 2) и возрасте > 15, цветок начинает цвести. Если уровень полива равен -4, -3, 3, 4, 5, то цветочки становятся грустными, показывая что уровень полива неправильный. Ночью (от 6 вечера до 6 утра) цветы спят. При уровне полива -5 цветы опадают, если его сейчас не полить, то цветок завянет. Цветок завял (уровень полива равен -6), поливать уже бесполезно.

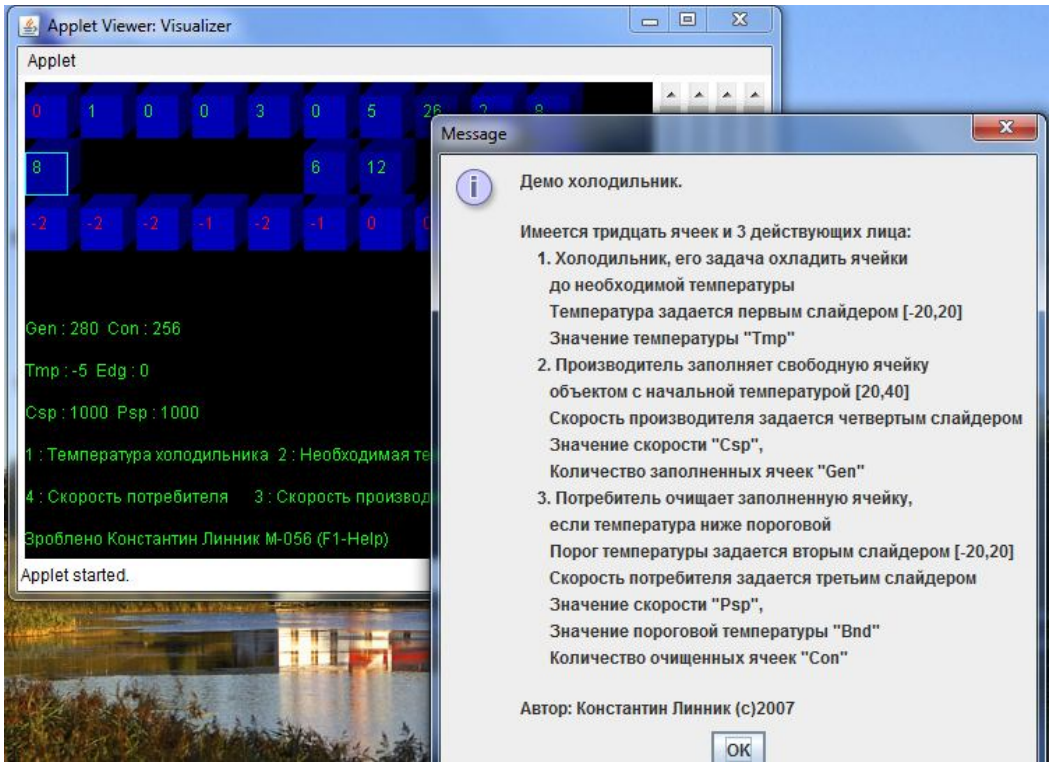
(**Kaktus\code.zip**)

7. **Пешеход**

   В данной работе реализовано поведение пешехода на пешеходном переходе, по которому проезжают автомобили. Данную работу нельзя назвать законченной, т.к. поведение пешехода не слишком логично. Однако достоинством работы является применение технологии с двойной буферизацией для прорисовки движущихся объектов без мерцания экрана. Основные управляющие клавиши: стрелки «Вверх», «Вниз», «Влево» и «Вправо», удержание которых приводит к движению пешехода. С помощью клавиши «Пробел» выполняется переключение светофора. Также задействованы клавиши Esc, Inter, PgUP, PgDown, но их функциональность не реализована в полной мере. Автор работы – Кудрявцев Александр. (**Roads2\code.zip**)

8. **Рефрижератор**

В данной работе моделируется работа рефрижератора по охлаждению помещаемых в него продуктов. Приложение реализовано с применением 4 потоков. Чтобы начать работу с приложением, необходимо ознакомиться со справкой к нему, вызывающейся по нажатию на клавишу F1. Автор работы – Линник Константин.



К сожалению, полную функциональность данной программы невозможно показать через html-страницу. Лучше воспользоваться архивом с исходным кодом и запустить апплет оттуда.(**Refrigerator\code.zip**)

9. Электронные часы

В данной работе предметом моделирования выбраны часы. Описан класс цифра, который содержит 6 основных точек цифры (по ним цифры отрисовываются (fillrect)), при создании объекта класса, в него передаются значения место положения цифры и само значение цифры. Так же реализованы методы: настройка часов, будильник, смена режима 24ч. на 12ч. Автор работы – Яковлев Антон.

Описание кнопок:
“set” – настройка :
первое нажатие – настройка минут

второе нажатие – настройка часов

третье нажатие – выход из режима настройки

“+” и “-“:

активны лишь тогда, когда кнопка “set” находится в двух первых состояниях.

При нажатии кнопок отнимаются и прибавляются часы/минуты (в зависимости от состояния “set”).

“R” - будильник:

первое нажатие – включение будильника, отображение времени, когда сработает будильник и возможность его настроить (опять же используем кнопку “set”).

При установки будильника появляется колокол с буквой S (jpg), означающий настройку будильника. Второе нажатие – возврат к часам.

При настроенном будильнике появляется колокол(jpg), оповещающий об установленном будильнике.

При срабатывании будильника играет мелодия (wave).

Действие можно повторить и перенастроить будильник.

“24h/12h” – смена режима 24ч на 12ч и наоборот :

первое нажатие – режим 24ч

второе нажатие – режим 12ч

(**Clock\code.zip**)

10.Игра «Змейка»

В данной работе реализована развлекательная игра «Змейка». По полю, разделенному на ячейки, движется змейка. При соприкосновении с непустыми ячейками длина змейки увеличивается. Необходимо двигать змейку так, чтобы ее длина в результате оказалась как можно длиннее. Управление осуществляется стрелками перемещения курсора «Вверх», «Вниз», «Влево» и «направо» для указания направления движения змейки. Клавиша «Пробел» приостанавливает игру, а «CapsLock» - значительно ускоряет. Автор – Доронин Евгений. (**Snake\code.zip**)

11. Сотовый телефон

Модель телефона была реализована как игра и представлена в java-апплете. При нажатии на клавишу «F1» выводится экран помощи. Модель обладает следующими свойствами:

- уровень заряда батареи (батарея);
- уровень сигнала (прием);

- количество денег на счету (баланс).

В игре также имеются счетчики сделанных звонков и отправленных sms.

Реализованы следующие возможности:

- перемещение по экрану со сменой областей (фона): «горка», «лес», «салон связи», «блок питания»;
- подзарядка;
- пополнение счета;
- отправка sms-сообщений;
- звонки.

Перемещение осуществляется при помощи клавиш «вверх», «вниз», «влево», «вправо». После преодоления границ области, она меняется (изменяется фон и возможности модели). Всего в игре представлены четыре области: «горка» (начальная область); «блок питания»; «лес»; «салон связи». При перемещении снижается уровень заряда батареи (на 0,1 при каждом шаге), с другой стороны, скорость перемещения зависит от уровня заряда: чем ниже уровень, тем ниже скорость.

Уровень сигнала меняется при каждом шаге случайным образом, но зависит также и от области: в области «горка» сигнал зависит от высоты, в областях «блок питания» и «салон связи» уровень составляет в среднем 50%, в области «лес» прием практически отсутствует. От уровня сигнала зависит успешность отправки sms или дозвона.

Если уровень заряда батарее ниже 35 в левом верхнем углу выводится сообщение «Пора бы подзарядиться!!!». Подзарядка осуществляется в соответствующем месте области «блок питания» при нажатии на кнопку «пробел».

Счет (баланс) можно пополнить в области «салон связи», нажав кнопку «пробел». Баланс уменьшается при успешных дозвонах (на $0.5) и при успешной отправке sms-сообщений (на $0.1). При нулевом балансе невозможно сделать звонок или отправить sms.

Отправка sms-сообщений осуществляется нажатием клавиши «shift», звонок – нажатием клавиши «enter», при этом, независимо от успешности отправки sms или дозвона, уменьшается уровень заряда батареи.

Если уровень заряда упал до нуля, телефон не сможет перемещаться, звонить, либо отправлять sms, и игра заканчивается.

Автор работы – Анженко Алексей. (**Mobile\code.zip**)

12. Рыба

В данной работе моделируется поведение рыбы в реке. Состояние объекта рыба выведено в верхней части апплета. Нажатие клавиши «H» моделирует процесс течения времени. Если несколько раз нажать на «H», то рыбка без приема пищи начинает уменьшаться в размерах.

Если рыбка проголодалась (состояние пункта Golod слишком мало), то её можно покормить одним из трёх типов еды: крупа, червяк, мальки (клавиши 0, 1 и 2).

Мальки и крупа не представляют никакой опасности для рыбки. А вот червяк может быть наживкой рыбака. То есть, съев червяка, рыбка может либо утолить голод, либо попасться на удочку.

Если рыба сыта, то она может двигаться с большой скоростью. Чем больше скорость рыбы, тем больше вероятность, что её не поймает хищник (клавиша P).

Если рыбка голодна, то её скорость может оказаться ниже, чем скорость хищника. В этом случае хищник съедает рыбку. Автор – Коледа Станислав. (**Fish2\code.zip**)

13. Пешеход

В данной работе реализована объект – пешеход. Управление апплетом осуществляется с помощью трех кнопок Button, расположенных в верхней части апплета. Автор – Сюваев В. (**Peshehod\code.zip**)

14. Наблюдатель

В данной работе запрограммирован апплет, который реализует модель «Наблюдателя» - класса Observer. Смысл данной модели состоит в том, что существует два класса, один из которых наблюдаемый, а второй – наблюдатель. Как только в наблюдаемом классе происходят какие-либо интересующие наблюдателя действия, наблюдатель немедленно реагирует. В данном апплете окно рисования разделено на 2 равных части, на которые помещены панели. Если рисовать мышкой по правой панели, то на панели слева сразу происходит зеркальное отображение рисунка. (**2Panel\code.zip**)

15. Дождь

В данной работе реализован апплет, имитирующий дождь. Капли дождя выпадают по событию класса Timer. Каждая капля продолжает свое движение из верхней части экрана до самой нижней. (**geom\code.zip**)

16. Игра «Змейка»

В данной работе реализована знакомая всем с детства игра. На прямоугольном поле установлена точка. Стрелками курсора можно управлять точкой и двигать ее в заданно направлении. В результате рисуется ломанная линия, состоящая из отрезков различной длины, которые параллельны одной из осей координат. Чтобы ускорить движение змейки нужно нажать клавишу shift и, удерживая ее, двигать курсор. (**Sketch\code.zip**)

17.Цветок

В данной работе предметом моделирования выбран цветок. Его жизненный цикл разделен на 4 этапа. Каждый этап характеризуется уменьшением «прожитого» времени на 25%.

Основной функциональностью цветка является размножение. Так же цветок требует полива. Без воды цветок не способен размножаться и погибает через определенное время.

На трех первых этапах цветок обладает полной функциональностью.

Таким образом, жизнь цветка можно поддерживать, лишь поливая его. Но существует смертоносный для цветка субъект «Змей Горыныч». Его появление не предсказуемо и пагубно влияет на цветок. (**Flower\code.zip**)

18. Облако

В данном апплете реализовано упрощенное поведение облака. Класс облако зависит от двух параметров: влажность(Vlagnost) и Солнце(Sun). Начальные параметры: влажность-20, Солнце-40. Пока уровень солнца ниже 50, влажность не повышается. При увеличении или уменьшении влажности изменяется размер облака и количество. Когда уровень солнца превысит 90 наступит засуха. Если уровень Солнца ниже 10, то наступает «ледниковый период». При влажности больше 10 идет дождь с грозой☺

Кнопки:

«1»-увеличение Солнца;

«2»-уменьшение Солнца;

«enter»- увеличение влажности без изменения Солнца;

«d»-прошел дождь и уменьшилась влажность;

(**Oblako\code.zip**)