ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ЦЕНТР ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «ПОВТ и АС»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам

Составитель
Габрельян Б.В.

Ростов-на-Дону

2003

Аннотация

В методической разработке рассматриваются основные математические функции, реализованные в классе Math, и отображение графиков гладких функций на заданном интервале значений аргумента в Java-апплетах, классы-оболочки, используемые в Java для "заворачивания" значений примитивного типа в объекты, и основы создания собственных пакетов (библиотек), класс ArrayList, позволяющий в Java-программах работать с динамическими массивами, и приемы организации работы с пользователем с помощью простейшего текстового меню. Даны задания по выполнению лабораторной работы.

Методические указания предназначены для студентов специальностей 075200 "Компьютерная безопасность", 220400 "Программное обеспечение вычислительной техники и автоматизированных систем", 351500 "Математическое обеспечение и администрирование информационных систем".

Составитель

Габрельян Б.В., к.ф.-м.н., доцент

Оглавление

Лабораторная работа № 3 Класс Math, рисование графиков функций в апплетах 5

1. Методы класса Math. 5
2. Рисование в апплете 7

Литература 9

Лабораторная работа № 4 Классы-оболочки, пакеты в Java 11

1. Классы-оболочки. 11
2. Пакеты. 13

Литература 16

Лабораторная работа № 5 Класс ArrayList, организация простейшего текстового меню 17

I. Класс ArrayList. 17
II. Стандартный ввод. 20
III. Организация простейшего текстового меню. 23

Литература 26

Лабораторная работа № 6 27

Лабораторная работа № 7 29

Лабораторная работа № 8 39

Лабораторная работа № 9 41

I. Форматирование данных. 41
II. Потоки ввода/вывода. 45

Ссылки: 52

Лабораторная работа № 10 53

I. Использование NetBeans. 53
II. Движущиеся изображения. 54

Лабораторная работа № 11 57

I. Двойная буферизация. 57
II. Таймеры. 59

Лабораторная работа № 12. 64

I. События, связанные с мышью и клавиатурой. 64

II. Рисование вне метода paint() (в обработчике сообщения). 70
Лабораторная работа № 13 73

I. Классы Frame, Applet и Panel. 73
II. Кнопки, метки и текстовые поля. 76
Лабораторная работа № 14 80

I. Класс Choice. 80
II. Классы Checkbox и CheckboxGroup. 81
Лабораторная работа №16. 85

ЛАБОРАТОРНАЯ РАБОТА № 3
КЛАСС MATH, РИСОВАНИЕ ГРАФИКОВ ФУНКЦИЙ В АППЛЕТАХ

1. Методы класса Math.

Стандартные математические функции в Java определены в классе (типе) Math. Они реализованы как статические методы и поэтому могут вызываться даже в случае, если не создан ни один экземпляр класса Math. Помимо арифметических, в Math определены функции min(x,y) и max(x,y). Кроме того, в этом же классе объявлены константы PI и E (основание экспоненты).

Функция	*Описание*	*Возвр. значение*
sin(x)	x - в радианах	double
cos(x)	x - в радианах	double
tan(x)	x - в радианах	double
asin(x)	арксинус от x, x из [-1.0 .. 1.0]	double
acos(x)	арккосинус от x, x из [-1.0 .. 1.0]	double
atan(x)	арктангенс от x, x из [-pi/2 .. pi/2]	double
exp(x)	e в степени x	double
log(x)	натуральный логарифм от x	double
sqrt(x)	корень квадратный из x	double
pow(x,y)	x в степени y, x - положительное	double
ceil(x)	наименьшее целое >= x	double
floor(x)	наибольшее целое <= x	double

round(x)	floor(x+0.5)	для x типа float - int, для x типа double - long
abs(x)	абсолютная величина x	соответствует типу x

Значения углов для тригонометрических функций указываются в радианах. Для удобства работы, в Math определены также функции преобразования значений из радиан в градусы (toDegrees(x), x - в радианах) и наоборот (toRadians(x), x - в градусах).

Например, значение y=2*cos(x)+3*tn(2*x) для x = e и x = 30є:

```
double x = Math.E, y1, y2;
y1 = 2*Math.cos(x)+3*Math.tan(2*x);
x = Math.toRadians(30);
y2 = 2* Math.cos(x)+3*Math.tan(2*x);
```

ЗАДАНИЕ 1.

Вывести на экран в виде таблицы значения функций sin(x), e^x / x * lg(x) для x из интервала [pi/15..pi], меняющихся с шагом pi/15.

ЗАДАНИЕ 2.

Реализовать алгоритм нахождения наибольшего отрицательного элемента двумерного массива, содержащего произвольное число строк и столбцов и, быть может, различное число элементов в каждой строке.

ЗАДАНИЕ 3.

Дан массив из 20 целых переменных. Конкретные значения для элементов задаются в конструкции инициализации. Найти

среднее геометрическое отрицательных элементов.

2. Рисование в апплете.

Апплет - это класс-наследник стандартного класса Applet из пакета java.applet (или класса JApplet из пакета javax.swing). Апплет выполняется в окне браузера. Точкой входа для апплета обычно является метод public void init() {…}, определенный в базовом классе Applet и переопределяемый для конкретного апплета.

Рисование в апплете проводится в методе paint() с использованием методов класса Graphics из библиотеки (пакета) java.awt. Например, отрезок прямой линии от точки (10,20) до точки (100,250), можно нарисовать, вызвав метод drawLine():

```
import java.applet.*;       // пакет, содержащий класс Applet
import java.awt.*;          // пакет, содержащий класс Graphics

public class MyApplet extends Applet {
// поля класса
  private int x1, x2, y1, y2;
// точка входа
  public void init()
  {
          x1 = 10;          y1 = 20;
          x2 = 100;         y2 = 250;
  }
// рисование апплета
  public void paint(Graphics g)
```

```
    {
            g.drawLine(x1,y1,x2,y2);
    }
}
```

При отображении графика функции на экране приходится проводить преобразование координат точек из системы координат, используемой в предметной области к системе координат окна, в котором выполняется апплет. Например, предположим, что апплет выполняется в окне 100 x 200 пикселей. Тогда минимальное возможное значение x-координаты точки видимой в окне равно 0, а максимальное 100. Для y-координаты 0 и 200 соответственно. Координаты точек реальной задачи могут принимать значения в любом другом диапазоне и, в общем случае не являются целыми числами и не обязательно положительные. Обозначим оконные координаты wndXMin, wndXMax, wndYMin и wndYMax. В примере выше их значения таковы: wndXMin = 0, wndYMin = 0, wndXMax = 100, wndYMax = 200. Пусть координаты конкретной точка реальной задачи x и y, а максимальные и минимальные возможные значения в этой системе координат - xMin, xMax, yMin и yMax. при переходе к оконной системе координат x и y должны перейти в wndX и wndY соответственно. Формулы преобразований таковы:

wndX = (x - xMin) / (xMax - xMin) * (wndXMax - wndXMin) + wndXMin

wndY = wndYMax - (y - yMin) / (yMax - yMin) * (wndYMax - wndYMin)

Далее нужно округлить полученные значения до ближайших целых и привести к типу int: (int)Math.round(…).

ЗАДАНИЕ 4.

Создайте апплет, отображающий в окне 300x300 пикселей график кривой f(x) = sin(x) на интервале x от -pi до pi.

ЗАДАНИЕ 5.

Создать класс Graph для отображения 2D-графика гладкой функции f(x) на заданном интервале изменения аргумента x. В классе должны быть предусмотрены два массива (или один двумерный массив) для хранения таблицы значений аргумента и функции, и метод setData, позволяющий пользователю класса задавать данные для конкретной функции и конкретного интервала по x. Необходимо предусмотреть методы для задания координат прямоугольной области, в которой должен отображаться график, и для отображения самого графика. Создать апплет, отображающий в окне 300x300 пикселей график кривой f(x) = sin(x) на интервале x от 0 до 2*pi с помощью объекта класса Graph.

Литература

1. К.Арнольд, Дж.Гослинг, Д.Холмс "Язык программирования Java. 3-е изд.". – М.: Вильямс. – 2001, 624с.

2. М.Холл, Л.Браун "Программирование для Web. Библиотека профессионала". – М.: Вильямс. – 2002, 1264с.

3. Б.Эккель "Философия Java. Библиотека программиста". – СПб.: Питер. – 2001, 880с.

4. П.Ноутон, Г.Шилдт "Java 2". – СПб.:BHV-Петербург. – 2001, 1072с.

5. Д.Бишоп "Эффективная работа: Java 2". – СПб.:Питер; К.:BHV. – 2002, 592с.

ЛАБОРАТОРНАЯ РАБОТА № 4
КЛАССЫ-ОБОЛОЧКИ, ПАКЕТЫ В JAVA

1. Классы-оболочки.

Для каждого простого встроенного типа в Java существует соответствующий класс, позволяющий создавать объекты, содержащие в себе значение какой-либо переменной или константы этого примитивного типа. Такие классы дают возможность преобразовать, например, целочисленную переменную в объект класса Integer, или, напротив, записать в целочисленную переменную значение объекта класса Integer. Преобразование переменной или константы в объект называют "заворачиванием", а соответствующие классы - классами-оболочками (wrapper-classes).

Для числовых типов определены следующие классы-оболочки:

Тип	**Класс**
byte	*Byte*
short	*Short*
int	*Integer*
long	*Long*
float	*Float*
double	*Double*

Например,

```
double d;
d = 1.73;
Double dObj = new Double(d);
Integer iObj = new Integer(-10);
```

Классы-оболочки содержат статические поля-константы для максимального (MAX_VALUE) и минимального (MIN_VALUE) возможного значений переменных соответствующего типа и, кроме того, набор методов для преобразования чисел в строки и наоборот. Для преобразования значений числовых типов в строки используются методы классов-оболочек с именем *type*Value, где вместо *type* задается имя соответствующего типа. Например,

```
Byte bObj = new Byte( (byte)65 );
byte b = bObj.byteValue();
…
b = Byte.MAX_VALUE;          // b = 127
// или (менее предпочтительное)
b = bObj.MAX_VALUE;
```

Объект класса-оболочки можно преобразовать в строку с помощью метода toString, например,

```
String str = bObj.toString();    // s = "65"
```

А переменную простого типа с помощью статического метода toString(), например,

```
str = Byte.toString(b);
```

ЗАДАНИЕ 1.

Напишите программу, выводящую на экран таблицу, содержащую минимальное и максимальное значения для всех простых числовых типов.

В каждом классе-оболочке предусмотрены также перегруженные (статические в отличие от *type*Value) методы parse*Type*.

Например,

```
int i = Integer.parseInt(str);
```

Статический метод toString() в классах Integer и Long имеет также форму с двумя параметрами: первый - значение, преобразуемое в строку и второй, целочисленный, основание системы счисления (значение из диапазона 2 ÷ 32) в которой будет представлено это значение. Например,

```
String sBin = Integer.toString((int)b,2);
```

ЗАДАНИЕ 2.

Напишите программу, получающую через командную строку целое десятичное число и отображающую на экране само это число и его представление в двоичной, восьмеричной и шестнадцатеричной системах счисления.

Классы-оболочки не содержат методов для изменения своих объектов.

ЗАДАНИЕ 3.

Используйте справочную систему Java для того, чтобы определить, какие еще методы реализованы в классе Integer.

2. Пакеты.

Java позволяет группировать логически связанные классы в отдельные библиотеки - пакеты. Все стандартные классы и классы расширений в Java собраны в отдельные пакеты. Пакеты могут быть вложенными друг в друга. Например, стандартные классы, обеспечивающие создание и работу с апплетами содер-

жатся в пакете java.applet. Т.е. пакет applet является вложенным в пакет java. Между именами пакетов и структурой каталогов файловой системы существует прямая связь. Например, классы пакета applet должны располагаться в подкаталоге applet каталога java. Для доступа к классу пакета необходимо явно указывать и имя класса, и имя пакета. Например,

```
class MyApplet extends java.applet.Applet {
…
}
```

Можно сделать напрямую видимым имя класса из какого-либо пакета в другом пакете с помощью команды import. Например,

```
import java.applet.Applet;
class MyApplet extends Applet {
…
}
```

Команда import позволяет сделать видимыми в пакете также все имена из другого апплета. Например,

```
import java.applet.*;
class MyApplet extends Applet {
…
}
```

Пакет java.lang, содержащий, например, классы System и Math, доступен любой программе без использования import.

Чтобы разместить классы в собственном пакете нужно ис-

пользовать конструкцию

package имя_пакета;

Эта конструкция должна размещаться в файле до определения классов и до конструкций import. Например, если нужно поместить классы A, B и C в пакет mypackage, то, если все классы размещены в одном исходном файле A.java, в его начало нужно поместить package mypackage, если же каждый класс содержится в своем собственном файле (A.java, B.java, C.java), то такую конструкцию нужно поместить в начало каждого из этих файлов.

Файл A.java:

```
package mypackage;
public class A {
...
}
```

Файл B.java:

```
package mypackage;
public class B {
...
}
```

Файл C.java:

```
package mypackage;
public class C {
...
}
```

В любом случае, после компиляции (успешной) получим три файла A.class, B.class, C.class. Необходимо разместить эти файлы в подкаталоге с именем, в точности совпадающим с именем пакета. В примере выше, если файлы с исходными текстами расположены в каталоге C:\tmp\my, то необходимо создать каталог C:\tmp\my\mypackage и, после трансляции, переместить туда class-файлы. Запускать интерпретатор JVM нужно так, чтобы текущим был каталог C:\tmp\my. При этом имя класса, содержащего метод main должно указываться полностью, вместе с именем пакета, т.е.

c:\tmp\my> java **mypackge.A**

(если main() объявлена в классе A).

ЗАДАНИЕ 4.

Преобразуйте приложение из предыдущего задания так, чтобы его реализация принадлежала пакету converter.

Литература

1. К.Арнольд, Дж.Гослинг, Д.Холмс "Язык программирования Java. 3-е изд.". – М.: Вильямс. – 2001, 624с.

2. М.Холл, Л.Браун "Программирование для Web. Библиотека профессионала". – М.: Вильямс. – 2002, 1264с.

3. Б.Эккель "Философия Java. Библиотека программиста". – СПб.: Питер. – 2001, 880с.

4. П.Ноутон, Г.Шилдт "Java 2". – СПб.:BHV-Петербург. – 2001, 1072с.

5. Д.Бишоп "Эффективная работа: Java 2". – СПб.:Питер; К.:BHV. – 2002, 592с.

ЛАБОРАТОРНАЯ РАБОТА № 5
КЛАСС ARRAYLIST, ОРГАНИЗАЦИЯ ПРОСТЕЙШЕГО ТЕКСТОВОГО МЕНЮ

I. Класс ArrayList.

В пакете java.util содержатся классы, предназначенные для работы со стандартными структурами данных. В частности, класс ArrayList обеспечивает работу с динамическими массивами произвольных объектов (т.е. элементами массива являются ссылки на объекты классов, производных от суперкласса Object). Так как ArrayList (и другие классы из java.util) способен манипулировать только объектами, в программе, с его помощью, нельзя создать динамически изменяющийся массив элементов примитивных встроенных типов (например, int). В подобных случаях используют классы-оболочки, например, Integer. В процессе работы с объектом класса ArrayList можно добавлять новые элементы в произвольное место массива, удалять произвольные элементы массива и заменять элементы в массиве на какие-либо другие. Как и для обычных массивов, для ArrayList определено понятие индекса элемента. Индексирование проводится естественным образом (от нуля).

Основные методы класса ArrayList:

boolean add(Object elem) - добавляет элемент, доступный по ссылке elem в

конец массива;

void add(int index, Object elem) - добавляет элемент elem в позицию index;

boolean contains(Object elem) - проверяет, содержится ли элемент elem в массиве;

Object get(int index) - возвращает элемент списка с индексом index;

Object set(int index, Object elem) - заменяет элемент с индексом index на elem, возвращает ссылку на прежний (заменённый) элемент;

Object remove(int index) - удаляет элемент с индексом index и возвращает ссылку на него;

void clear() - удаляет все элементы из массива;

int indexOf(Object elem) - возвращает индекс первого вхождения элемента elem в массив;

boolean isEmpty() - возвращает true, если в массиве нет элементов;

int size() - возвращает значение, равное количеству элементов в списке;

Object[] toArray() - создает и возвращает массив неизменяемого размера, содержащий все элементы объекта ArrayList.

Более подробную информацию можно найти в документации к Java SDK.

Стандартная техника работы с ArrayList такова:

1. Используем директиву import для того, чтобы имя ArrayList из пакета java.util было напрямую доступно в нашем пакете:

```
import java.util.ArrayList;
```

2. Создаем объект класса ArrayList (операция new) и сохраняем ссылку на него в некоторой переменной (например, array):

```
ArrayList array = new ArrayList();
```

3. Добавляем элементы:

```
array.add( new Integer(12) );
array.add( new Integer(-100) );
```

4. Получаем значение последнего элемента массива:

```
int value = 0;  // локальные переменные методов не инициализируются по умолчанию
if( !array.isEmpty() ) // если в массиве есть элементы
{
        int size = array.size(); // число элементов в массиве
        Integer i = (Integer)array.get( size-1 ); // последний объект массива
        value = i.intValue(); // значение последнего элемента массива
// можно использовать эквивалентную, но более короткую запись:
//        value = ((Integer)array.get( size - 1 )).intValue();
}
```

ЗАДАНИЕ 1.

Протестируйте класс java.util.ArrayList с помощью программы, которая проводит базовые операции с динамическим массивом (добавление элемента, удаление элемента, определение количества элементов в списке, доступ к элементу списка). В качестве элементов списка используйте целые значения.

II. Стандартный ввод.

Работа с файлами и устройствами в Java организована с помощью множества классов. Реализация стандартного ввода (для консольного приложения) оказывается более сложной, чем в других языках программирования. Например, в классе InputStream, имеется только метод read(), позволяющий вводить коды отдельных символов (коды нажатых клавиш). Класс System содержит статическое поле in - объект класса InputStream. Поэтому, в простейшем случае, можно получить код нажатой клавиши (величину int) так:

```
int code = 0;
code = System.in.read();
```

В действительности все несколько сложнее. В процессе работы со стандартным устройством ввода могут возникать ошибочные ситуации. В Java реакцией на такую ситуацию будет выбрасывание исключения. Исключения мы будем рассматривать позже, но Java не позволит обращаться к read() до тех пор, пока мы не организуем обработчик возможной ошибочной ситуации (обработчик исключения). Пока нужно просто запомнить, что все действия с вводом данных должны размещаться внутри следующей конструкции:

```
int code = 0;
try {
        code = Syatem.in.read();
}
catch(java.io.IOException e) {}
```

Вы знаете, что ввод с клавиатуры буферизован, т.е. вво-

димые символы помещаются в буфер в ОЗУ, а программа считывает их из этого буфера. Предположим, что в буфер помещено 10 символов, а считано лишь 5. Здесь не возникает проблем. Но, если вновь обратиться к функции чтения символов, введенных с помощью клавиатуры (например, read()), то функция вернет не вновь введенный символ, а тот, что остался в буфере. Поэтому стандартный прием при вводе данных со стандартного устройства - это очистка буфера и, лишь затем, запрос на чтение символов. Класс InputStream содержит два метода: available(), возвращающий количество байт, содержащихся на данный момент в буфере, и skip(), удаляющий указанное количество байт из буфера. Итог:

```
int code = 0;
try {
System.in.skip( System.in.available() );    // очистить буфер клавиатуры
code = System.in.read();                    // получить код нажатой клавиши
}
catch(java.io.IOException e) {}
```

Теперь можно написать собственный метод, позволяющий вводить с клавиатуры целые числа из диапазона, например, от -99 999 999 до 999 999 999. Для хранения таких чисел достаточно иметь массив из девяти байт. Кроме того, удобно использовать коды символов, '0' - 48, '1' - 49, …, '9' - 57. Обычно ввод данных заканчивается после нажатия на клавишу Enter (коды 13 и 10), кроме того, число не может содержать пробельный символ (код 32).

```
    private static int getIntValue()    {
      int       val = 0,          // введенное значение
      len = 0;          // реальное количество введенных символов
      byte b[]=new byte[9]; // массив для хранения кодов введенных символов
      int by=0; // вспомогат. переменная для ввода кода очередного символа
      int sign = 1;     // знак числа
      try {
            System.in.skip(System.in.available());
            while(true)       // число может содержать несколько символов
            {
                  by = System.in.read(); // код очередного символа
                  // первым символом (len == 0) может быть '-'
                  if( len == 0 && (char)by == '-' ) { sign = -1; continue; }
                  // символ, завершающий ввод числа:
                  // Enter, пробел или не цифра
                  if( by == 13 || by == 32 || by<48 || by>57 ) break;
                  b[len++]=(byte)(by - 48);       // превращаем код символа в число
            if(len>8) break;//слишком много символов в буфере клавиатуры
```

```
                }
            }
            catch( java.io.IOException e) {}
            if( len <1 ) return -1; // сразу нажата Enter или клавиша - не цифра
            int ten=1; // для преобразования последовательности цифр в число
            // 10 в степени, соответствующей позиции цифры в числе
            for(int i=len-1; i>=0; i--)          // от последней цифры - к первой
            {
                val += b[i] * ten;
                ten *= 10;
            }
            return val*sign;
        }
```

III. Организация простейшего текстового меню.

Одним из общепринятых способов взаимодействия программы с ее пользователем является организация меню. На экран выводятся пронумерованные строки - пункты меню, и пользователю предлагается ввести с клавиатуры целое положительное число - номер пункта меню. Тогда общая структура класса, поддерживающего меню может выглядеть так:

```
package lab7;
public class Lab7 {
  boolean quit = false;    // признак окончания работы про-
```

граммы

```
  String[] menu = {"1. Punkt1.", "2. Quit."}; // пункты меню
  // другие поля-данные
  …
private int getIntValue() {…}
private void showMenu() {…} //выводит на экран пункты меню и приглашение  // на ввод номера выбранного пункт меню
private void processMenu(int choice)      // выполнение выбранного пункта меню
{
  …
  switch( choice )
  {
        …
        case 2: // Quit
              quit = true;
              break;
  }
}
public static void main(String[] arg)
{
  Lab7 lab = new Lab7();
  while( ! lab.quit ) // пока флажок сброшен повторять
  {
        lab.showMenu();
        lab.processMenu(getIntValue());
  }
}
```

}

ЗАДАНИЕ 2.

Протестируйте класс java.util.ArrayList с помощью программы, которая взаимодействует с пользователем посредством простого меню:

1. Add first.
2. Add last.
3. Remove first.
4. Remove last.
5. Show array.
6. Quit.

Your choice (1 - 6) ?: _

В качестве элементов списка используйте целые значения.

ЗАДАНИЕ 3.

С помощью композиции создайте класс Queue, использующий класс java.util.ArrayList для представления структуры данных "очередь". Очередь функционирует следующим образом: новый элемент добавляется только в конец очереди, удаляется всегда первый элемент очереди. Протестируйте класс Queue с помощью программы, которая взаимодействует с пользователем посредством простого меню:

1. Add element.
2. Remove element.
3. Show queue.
4. Quit.

Your choice (1 - 4) ?: _

В качестве элементов списка используйте целые значения.

ЗАДАНИЕ 4.

Реализуйте задачу из ЗАДАНИЯ 3 с помощью наследования.

Литература

1. К.Арнольд, Дж.Гослинг, Д.Холмс "Язык программирования Java. 3-е изд.". – М.: Вильямс. – 2001, 624с.

2. М.Холл, Л.Браун "Программирование для Web. Библиотека профессионала". – М.: Вильямс. – 2002, 1264с.

3. Б.Эккель "Философия Java. Библиотека программиста". – СПб.: Питер. – 2001, 880с.

4. П.Ноутон, Г.Шилдт "Java 2". – СПб.:BHV-Петербург. – 2001, 1072с.

5. Д.Бишоп "Эффективная работа: Java 2". – СПб.:Питер; К.:BHV. – 2002, 592с.

ЛАБОРАТОРНАЯ РАБОТА № 6

При реализации программного проекта обычно используют подход, основанный на разделении обязанностей. Это значит, что решение задачи получают путем создания нескольких классов, выполняющих частные задачи, а не одного сверхбольшого специализированного класса, решающего всю задачу сразу. Например, нужно создать программу, отображающую на экране графики различных функций. Эту задачу можно разбить на несколько подзадач. Во-первых, на графике нужно разместить оси координат с метками и значениями аргумента и функции в граничных и некоторых промежуточных точках. Во-вторых, один график может содержать несколько кривых (обычно разного цвета) - графиков различных функций. В-третьих, на графике можно разместить пояснение (legend), указывающее какой функции соответствует кривая заданного цвета. Тогда можно создать следующие классы:

- класс Axis для представления одной оси координат (может быть вертикальная или горизонтальная ось);

- класс Curve для представления графика одной функции (может содержать метод setData() подобный используемому в лаб. 4b);

- класс Legend для представления пояснений к графику;

- класс Graph - объемлющий класс, полями которого являются две ссылки на Axis (ось абсцисс и ось ординат), массив ссылок на Curve (в упрощенном случае можно зафиксировать максимальное возможное число кривых, например, значением 10), ссылку на Legend.

Класс Graph должен содержать методы, позволяющие на-

страивать параметры осей координат, задавать пояснения и таблицы значений для отображаемых кривых и метод для отображения всего графика. Этот класс должен быть доступен тому, кто собирается работать с графиками функций. Об остальных классах (Axis, Curve, Legend) он может и не догадываться.

ЗАДАНИЕ. Модифицируйте лаб. 4b так, чтобы задача рисования графика решалась набором классов, указанных выше. Разместите эти классы в пакете, например, lab6.graph. Создайте аплет-тест, расположенный в другом пакете, например, lab6, который использует пакет lab6.graph для отображения (одновременного) графиков двух функций, например, sin(x) и cos(x), на интервале изменения аргумента [-pi, pi].

ЛАБОРАТОРНАЯ РАБОТА № 7

В пакете java.util содержатся классы, предназначенные для работы со стандартными структурами данных. В частности, класс ArrayList обеспечивает работу с динамическими массивами произвольных объектов (т.е. элементами массива являются ссылки на объекты классов, производных от суперкласса Object). Так как ArrayList (и другие классы из java.util) способен манипулировать только объектами, в программе, с его помощью, нельзя создать динамически изменяющийся массив элементов примитивных встроенных типов (например, int). В подобных случаях используют классы-оболочки, например, Integer. В процессе работы с объектом класса ArrayList можно добавлять новые элементы в произвольное место массива, удалять произвольные элементы массива и заменять элементы в массиве на какие-либо другие. Как и для обычных массивов, для ArrayList определено понятие индекса элемента. Индексирование проводится естественным образом (от нуля).

Основные методы класса ArrayList:

boolean add(Object elem) -добавляет элемент, доступный по ссылке elem в конец массива;

void add(int index, Object elem) - добавляет элемент elem в позицию index;

boolean contains(Object elem) - проверяет, содержится ли элемент elem в массиве;

Object get(int index) - возвращает элемент списка с индексом index;

Object set(int index, Object elem) - заменяет элемент с ин-

дексом index на elem, возвращает ссылку на прежний (заменен-ный) элемент;

Object remove(int index) - удаляет элемент с индексом index и возвращает ссылку на него;

void clear() - удаляет все элементы из массива;

int indexOf(Object elem) - возвращает индекс первого вхож-дения элемента elem в массив;

boolean isEmpty() - возвращает true, если в массиве нет элементов;

int size() - возвращает значение, равное количеству элемен-тов в списке;

Object[] toArray() - создает и возвращает массив неизме-няемого размера, содержащий все элементы объекта ArrayList.

Более подробную информацию можно найти в документа-ции к Java SDK.

Стандартная техника работы с ArrayList такова:

1. Используем директиву import для того, чтобы имя ArrayL-ist из пакета java.util было напрямую доступно в нашем пакете:

```
import java.util.ArrayList;
```

2. Создаем объект класса ArrayList (операция new) и сохра-няем ссылку на него в некоторой переменной (например, array):

```
ArrayList array = new ArrayList();
```

3. Добавляем элементы:

```
array.add( new Integer(12) );
array.add( new Integer(-100) );
```

4. Получаем значение последнего элемента массива:

```
int value = 0;   // локальные переменные методов не инициализируются по умолчанию
if( !array.isEmpty() ) // если в массиве есть элементы
{
        int size = array.size(); // число элементов в массиве
        Integer i = (Integer)array.get( size-1 ); // последний объект массива
        value = i.intValue(); // значение последнего элемента массива
// можно использовать эквивалентную, но более короткую запись:
//        value = ((Integer)array.get( size - 1 )).intValue();
}
```

***

ЗАДАНИЕ 1. Протестируйте класс java.util.ArrayList с помощью программы, которая проводит базовые операции с динамическим массивом (добавление элемента, удаление элемента, определение количества элементов в списке, доступ к элементу списка). В качестве элементов списка используйте целые значе-

ния.

***

Работа с файлами и устройствами в Java организована с помощью множества классов. Реализация стандартного ввода (для консольного приложения) оказывается более сложной, чем в других языках программирования. Например, в классе InputStream, имеется только метод read(), позволяющий вводить коды отдельных символов (коды нажатых клавиш). Класс System содержит статическое поле in - объект класса InputStream. Поэтому, в простейшем случае, можно получить код нажатой клавиши (величину int) так:

```
int code = 0;
code = System.in.read();
```

В действительности все несколько сложнее. В процессе работы со стандартным устройством ввода могут возникать ошибочные ситуации. В Java реакцией на такую ситуацию будет выбрасывание исключения. Исключения мы будем рассматривать позже, но Java не позволит обращаться к read() до тех пор, пока мы не организуем обработчик возможной ошибочной ситуации (обработчик исключения). Пока нужно просто запомнить, что все действия с вводом данных должны размещаться внутри следующей конструкции:

```
int code = 0;
try {
        code = Syatem.in.read();
}
catch(java.io.IOException e) {}
```

Вы знаете, что ввод с клавиатуры буферизован, т.е. вводимые символы помещаются в буфер в ОЗУ, а программа считывает их из этого буфера. Предположим, что в буфер помещено 10 символов, а считано лишь 5. Здесь не возникает проблем. Но, если вновь обратиться к функции чтения символов, введенных с помощью клавиатуры (например, read()), то функция вернет не вновь введенный символ, а тот, что остался в буфере. Поэтому стандартный прием при вводе данных со стандартного устройства - это очистка буфера и, лишь затем, запрос на чтение символов. Класс InputStream содержит два метода: available(), возвращающий количество байт, содержащихся на данный момент в буфере, и skip(), удаляющий указанное количество байт из буфера. Итог:

```
int code = 0;
try {
        System.in.skip( System.in.available() );    //
очистить буфер клавиатуры
        code = System.in.read();                    // по-
лучить код нажатой клавиши
}
catch(java.io.IOException e) {}
```

Теперь можно написать собственный метод, позволяющий вводить с клавиатуры целые числа из диапазона, например, от -99 999 999 до 999 999 999. Для хранения таких чисел достаточно иметь массив из девяти байт. Кроме того, удобно использовать коды символов, '0' - 48, '1' - 49, …, '9' - 57. Обычно ввод данных заканчивается после нажатия на клавишу Enter (коды 13 и 10),

кроме того, число не может содержать пробельный символ (код 32).

```
private static int getIntValue()
{
        int     val = 0,        // введенное значение
                len = 0;        // реальное количество вве-
        денных символов
        byte b[] = new byte[9]; // массив для хранения ко-
дов введенных символов
        int by = 0;     // вспомогательная переменная для
ввода кода очередного символа
        int sign = 1;   // знак числа
        try {
                System.in.skip(System.in.available());
                while(true)     // число может содержать
несколько символов
                {
                        by = System.in.read(); // код оче-
редного символа

                        // первым символом (len == 0) мо-
жет быть '-'
                        if( len == 0 && (char)by == '-' ) {
sign = -1; continue; }

                        // символ, завершающий ввод чис-
ла:
```

```
                                    // Enter, пробел или не цифра
                                    if( by == 13 || by == 32 || by<48 ||
by>57 ) break;

                                    b[len++] = (byte)(by - 48);      //
превращаем код символа в число
                                    if( len > 8 ) break; // слишком много
символов в буфере клавиатуры
                                }
                        }
                        catch( java.io.IOException e) {}
                        if( len <1 ) return -1; // сразу нажата Enter или кла-
виша - не цифра
                        int ten = 1;      // для преобразования последова-
тельности цифр в число
                                    // 10 в степени, соответствующей
позиции цифры в числе
                        for(int i=len-1; i>=0; i--)        //     от     последней
цифры - к первой
                        {
                                val += b[i] * ten;
                                ten *= 10;
                        }
                        return val*sign;
                }
```

Одним из общепринятых способов взаимодействия программы с ее пользователем является организация меню. На экран

выводятся пронумерованные строки - пункты меню, и пользователю предлагается ввести с клавиатуры целое положительное число - номер пункта меню. Тогда общая структура класса, поддерживающего меню может выглядеть так:

```
package lab7;

public class Lab7 {

  boolean quit = false;    // признак окончания работы программы
  String[] menu = {"1. Punkt1.", "2. Quit."}; // пункты меню
  // другие поля-данные
  …

  private int getIntValue() {…}

  private void showMenu() {…} // выводит на экран пункты меню и приглашение
                              // на ввод номера выбранного пункт меню
  private void processMenu(int choice)    // выполнение выбранного пункта меню
  {
      …
      switch( choice )
      {
          …
```

```
                case 2: // Quit
                        quit = true;
                        break;
                }
        }

        public static void main(String[] arg)
        {
                Lab7 lab = new Lab7();
                while( ! lab.quit ) // пока флажок сброшен повто-
рять
                {
                        lab.showMenu();
                        lab.processMenu(getIntValue());
                }
        }
}
```

***

ЗАДАНИЕ 2. Протестируйте класс java.util.ArrayList с помощью программы, которая взаимодействует с пользователем посредством простого меню:

1. Add first.
2. Add last.
3. Remove first.
4. Remove last.

5. Show array.

6. Quit.

Your choice (1 - 6) ?: _

В качестве элементов списка используйте целые значения.

***

***

ЗАДАНИЕ 3. С помощью композиции создайте класс Queue, использующий класс java.util.ArrayList для представления структуры данных "очередь". Очередь функционирует следующим образом: новый элемент добавляется только в конец очереди, удаляется всегда первый элемент очереди. Протестируйте класс Queue с помощью программы, которая взаимодействует с пользователем посредством простого меню:

1. Add element.

2. Remove element.

3. Show queue.

4. Quit.

Your choice (1 - 4) ?: _

В качестве элементов списка используйте целые значения.

***

***

ЗАДАНИЕ 4. Реализуйте задачу из ЗАДАНИЯ 3 с помощью наследования.

***

ЛАБОРАТОРНАЯ РАБОТА № 8

Функции, рисующие графические примитивы реализованы как методы класса java.awt.Graphics (см. документацию к Java SDK). В пакете java.awt содержится также класс Color для представления цветов. В этом классе есть статические константы red, green, yellow, black и т.д. (см. документацию к Java SDK)

ЗАДАНИЕ 1.

Спроектируйте и реализуйте пакет для отображения рисунков, построенных из графических примитивов. Как минимум, должны быть реализованы классы для работы с окружностями, кругами (закрашенными "окружностями"), прямоугольниками и закрашенными прямоугольниками. Обязательно должен присутствовать (по крайней мере, один) абстрактный класс, расположенный на вершине иерархии. Отображение конкретной фигуры возможно как в графическом, так и в текстовом виде. Т.е. нужно разработать для каждого (конкретного) класса свою форму "командного" представления, например, для окружности с центром в точке (100,100) и радиусом 50 пикселей, красного цвета:

CIRCLE (100,100) - 50 red

Для такого же круга (Filled CIRCLE):

FCIRCLE (100,100) - 50 red

Для прямоугольников можно использовать, например, обозначения RECT и FRECT.

Для графического вывода каждый класс должен иметь один метод (например, show), а для текстового - другой (например, print).

Используйте какую-нибудь обобщенную структуру данных (например, java.util.ArrayList) для представления рисунка в целом (списка графических примитивов, образующих при отображении рисунок).

ЗАДАНИЕ 2.

Протестируйте созданный в Задании 1 пакет. Для этого постройте и отобразите в графическом и текстовом виде рисунок, состоящий из следующих примитивов (в указанном порядке!):

FRECT (7,24) - 100, 100 red

CIRCLE (155,155) - 50 red

FCIRCLE (100,100) - 50 green

FRECT (210,24) - 100, 100 red

FCIRCLE (153,170) - 35 yellow

FCIRCLE (210,100) - 50 green

FCIRCLE (120,120) - 20 black

FCIRCLE (190,120) - 20 black

FCIRCLE (155,155) - 20 red

ЗАДАНИЕ 3.

Реализуйте новый примитив - отрезок прямой и создайте несложный (но привлекательный!) рисунок из реализованных примитивов (если необходимо, добавьте новые примитивы).

ЛАБОРАТОРНАЯ РАБОТА № 9

I. Форматирование данных.

Под форматированием данных понимают преобразования чисел и более сложных данных (например, календарных дат, временных интервалов, содержащих часы, минуты, секунды) в строки, либо обратные преобразования. При этом часто, одни и те же значения после преобразования могут иметь различный вид. Например, число 3.14159265 можно преобразовать в строку "3.1416" или в строку "3.14" и т.д.

1) Числа. Для форматирования чисел можно использовать класс DecimalFormat из пакета java.text.

Символы, используемые для форматирования, более подробно см. в документации к Java SDK [1]:

Символ	Описание
0	цифра
#	если незначащий ноль - игнорируется, иначе цифра
.	позиция десятичной точки
,	позиция разделителя групп разрядов (например, число 1000000.5 выводится как 1,000,000.5)
-	вывод знака числа
E	позиция, после которой размещается степень числа
%	значение отображается как процент

Например:

```
import java.text.*;
…
DecimalFormat df = new DecimalFormat("#.##E00");
```

```
double d1 = 123.75, d2 = 1.7e3, d3 = 0.12345;
System.out.println("d1="+d1+"\td2="+d2+"\td3="+d3);
System.out.println("d1="+df.format(d1)+"\td2="+
df.format(d2)+"\td3="+ df.format(d3));
…
```

Получаем на экране:

d1=123.75 d2=1700.0 d3=0.12345

d1=1.24E02 d2=1.7E03 d3=1.23E-01

В экспоненте можно использовать только символ '0'. В этом случае количество нулей определяет минимальное число цифр в представлении экспоненты. Например,

```
import java.text.*;
…
DecimalFormat df = new DecimalFormat("0.0000E00");
double d1 = 15.5e123, d2 = 15.5;
System.out.println("d1="+df.format(d1)+"\td2="+
df.format(d2));
…
```

Получаем на экране:

d1=1.5500E124 d2=1.5500E01

ЗАДАНИЕ 1. Создайте приложение, отображающее на экране таблицу значений некоторой функции на заданном интервале с выровненными столбцами значений аргумента и функции. Используйте DecimalFormat.

2) Календарные даты и время. Изначально для представ-

ления дат и времени в Java использовался класс java.util.Date, но теперь многие его методы являются устаревшими и их не рекомендуются использовать. Вместо этого нужно пользоваться возможностями, заложенными в классах Calendar, DateFormat и их наследниках [2]. Но для преобразований часто необходимо иметь объект класса Date. Обычно дата и время в Java упаковываются в величину long.

Например, статический метод класса System currentTimeMillis() возвращает текущее время как величину long. Она содержит число миллисекунд, прошедших с момента "сотворения мира" в операционной системе Unix - полуночи 1 января 1970 года. Используя это значение, *можно создать* объект класса Data. Для преобразования объекта Data в строку *можно воспользоваться* классом SimpleDateFormat, производным от DateFormat. Конструктору SimpleDateFormat *можно передать* строку, содержащую, в закодированном виде, описание формата представления даты и времени (см. таблицу ниже). Саму эту отформатированную строку возвращает метод format() класса SimpleDateFormat. Метод format() имеет один параметр - объект класса Data.

Некоторые символы, используемые для форматирования (более подробно см. в документации к Java SDK):

Символ	Значение
y	год
M	месяц
d	день
h	часы (12 часовое представление)
H	часы (24 часовое представление)

m	минуты
s	секунды
S	тысячные доли секунды
a	am или pm для 12-часового представления часов

Например, чтобы вывести на консоль текущие дату и время так, чтобы номер дня в месяце, номер месяца, часы (в 24-часовом представлении), минуты и секунды содержали две цифры, а год - 4 цифры, прежде выводилась дата, а, затем, через пробел, время, чтобы разделителем для даты был знак '.', а для времени - ':':

```
import java.util.*;
…
SimpleDateFormat sdf = new SimpleDateFormat ("dd.MM.yyyy HH:mm:ss");
String s = sdf.format( new Date(System.currentTimeMillis()) );
System.out.println("Current data/time: "+s);
…
```

Более простой вариант:

```
System.out.println("Current data/time: "+ new Date(System.currentTimeMillis());
```

отображает время и дату в фиксированном, зависящем от выбранных в данный момент в JVM национальных установок (locale).

II. Потоки ввода/вывода.

Содержимое файлов можно рассматривать как последовательность байт или текст (последовательность символов). В этом смысле файл представляет собой поток байт или символов. Саму идею потока элементов расширяют и на устройства, подобные клавиатуре (стандартный поток ввода) или монитору (стандартный поток вывода). Java содержит иерархии классов для байтовых и символьных потоков ввода и вывода (пакет java.io). Более того, часто нужно проводить преобразование потока байт в поток символов (или наоборот). Поэтому существуют также классы для этих преобразований. Байтовые потоки ввода представлены классами, производными от InputStream, вывода - от OutputStream. Символьные потоки - от классов Reader и Writer соответственно. Классы преобразований содержат в своих именах и Stream и Reader (или Writer). Стандартные потоки ввода/вывода по историческим причинам не символьные, а байтовые. Поэтому для работы с ними часто объект, представляющий байтовый поток (например, System.in) "заворачивают" в объект класса, представляющего символьный поток. Еще одной особенностью ввода/вывода является буферизация. В памяти создается буфер и все операции ввода из файла (или устройства) или вывода в файл (на устройство) проводятся через этот буфер. Программа читает (или записывает) данные из буфера, который, по мере необходимости, подгружается средствами исполняющей системы Java. Имена классов для буферизованных потоков содержат слово Buffered (например, BufferedReader или BufferedWriter). В классе BufferedReader определен метод readLine(). Этот метод извлекает символы из потока до тех пор, пока не встретит признак перехода

на новую строку ('\n' или '\r', или "\r\n"), создает новый объект класса String, содержащий эти символы и возвращает ссылку на него. Наконец, еще один важный момент связан с тем, что при выполнении операций ввода/вывода могут возникать исключительные ситуации. Поэтому необходимо перехватывать исключения, связанные с вводом/выводом (try/catch). Например, запросить с клавиатуры целое число, можно так:

```
import java.io.*;
…
try {

BufferedReader console =
        new BufferedReader(    new InputStreamReader( System.in ) );
                    |                                          |
|
                    |                          |          стандартный поток ввода (байтовый)
                    |                          |
                     |            символьный поток ввода на основе байтового
                    |
 буферизованный символьный поток ввода

String str = console.readLine();
int value = Integer.parseInt( str );
```

```
catch( IOException e) {
        System.out.println( "Something wrong!!!" );
}
...
```

Символьный поток ввода для файла можно создать с помощью класса FileReader. Тогда ввод целого числа из текстового файла "data.txt" будет выглядеть так:

```
...
try {

BufferedReader file =
        new BufferedReader(    new FileReader( "data.txt" ) );
String str = file.readLine();
int value = Integer.parseInt( str );

catch( IOException e) {
        System.out.println( "Something wrong!!!" );
}

...
```

Если метод readLine() не может прочитать строку (достигнут конец файла?), то он возвращает пустую ссылку (null). Если файловый поток больше не нужен, его следует закрыть:

```
...
file.close();
...
```

В классе BufferedWriter (буферизованный символьный поток вывода) определены методы write() и newLine(). Первый их них помещает указанное число символов их указанной строки в поток, второй помещает в поток признак перехода на новую строку. Поток символьного файлового вывода можно создать с помощью FileWriter. Например, переписать все содержимое текстового файла "prices.us" в файл "prices.ru", заменяя все вхождения символа '$' на символ 'p' (латинская буква!), можно так:

```
import java.io.*;
…
try {

BufferedReader inFile =
        new BufferedReader(    new FileReader( " prices.us" )
);
BufferedWriter outFile =
        new BufferedWriter(     new FileWriter( " prices.ru" )
);

String str1 = null;  // ссылка на считываемую строку
String str2 = null;  // ссылка на записываемую строку

while( (str1 = inFile.readLine()) != null )          // пока не
достигнут конец файла
{
        str2 = str1.replace( '$', 'p' ); // метод класса String
создает новую строку
```

```
            outFile.write( str2, 0, str2.length() );
                                |     |     |
                                |       |   количество записываемых
символов
                                |   индекс начального записываемо-
го символа строки
                    строка-источник символов

            outFile.newLine();  // поместить в поток '\n'
        }
        inFile.close();
        outFile.close();

        catch( IOException e) {
            System.out.println( "Something wrong!!!" );
        }

        …
```

Помимо классов для работы с потоками в Java определен класс java.io.File. Этот класс не предназначен для ввода/вывода данных. С его помощью можно получать (изменять) различную информацию о файле (или каталоге). Например,

```
import java.io.*;
…
File f = new File("myfile.txt");
if( f.exists() )            // если файл существует
{
```

```
        …
    }
    …
```

В частности, в классе File объявлены следующие методы:

boolean canRead() - возвращает true, если файл доступен для чтения;

boolean canWrite()- возвращает true, если файл доступен для записи;

boolean isHidden()- возвращает true, если файл является скрытым;

boolean isFile()- возвращает true, если объект связан с файлом;

boolean isDirectory()- возвращает true, если объект класса File связан с каталогом (папкой);

boolean exists() - возвращает true, если файл существует;

String getCanonicalPath()- возвращает полный путь и имя файла;

long length() - возвращает размер файла в байтах;

long lastModified() - возвращает дату и время последней модификации файла;

String[] list() - **только если объект связан с каталогом!**; возвращает массив строк, содержащих имена файлов и подкаталогов каталога, связанного с объектом, для которого вызван этот метод;

File[] listFiles()- **только если объект связан с каталогом!**; возвращает массив объектов класса File, соответствующих

файлам и подкаталогам каталога, связанного с объектом, для которого вызван этот метод;

ЗАДАНИЕ 2. Напишите приложение, находящее в текущем каталоге файл с самой старой датой модификации и отображающее информацию об этом файле на экране. Информация должна включать полный путь и имя файла, дату и время последней модификации, размер файла и, в закодированном виде, атрибуты (доступ на чтение, на запись, является ли скрытым). Закодировать атрибуты можно, например, так:

rwh // доступен для чтения, записи и скрытый

rw- // доступен для чтения, записи, но не скрытый

r-- // доступен для чтения, но не для записи и не скрытый

-w- // доступен для записи, но не для чтения, и не скрытый

ЗАДАНИЕ 3. Создайте приложение (не апплет!), записывающее в файл информацию обо всех файлах и подкаталогах (папках) текущего каталога, а также, обо всех файлах и подкаталогах подкаталогов и т.д., т.е. обо всей ветви файловой подсистемы, начинающейся от текущего каталога. (Удобно использовать рекурсию). Информация должна включать в себя атрибуты (для каталога - строку [dir]), размер (только для файлов), дату и время последней модификации и имя. Кроме того, перед выводом информации о содержимом каталога нужно печатать строку, содержащую его полное (включая весь путь от корневого каталога) имя.

Ссылки:

[1] Холл М., Браун Л. Программирование для Web. Библиотека профессионала. - М.: "Вильямс", 2002. - 1264 с.

[2] Арнольд К., Гослинг Д., Холмс Д. Язык программирования Java. 3-е изд. - М.: "Вильямс", 2001. - 624 с.

ЛАБОРАТОРНАЯ РАБОТА № 10

I. Использование NetBeans.

На текущий момент организация NetBeans (www.NetBeans.org) предлагает интегрированную среду (IDE) для создания, отладки и сопровождения программных проектов, организованную по модульному принципу (встраиваемые модули - plug-in). NetBeans была куплена компанией Sun Microsystems, которая предлагает основанный на разработке NetBeans продукт Forte for Java. Сама же организация NetBeans работает по принципу открытого кода (Open Source).

Продукт NetBeans подобно другим популярным IDE содержит набор "колдунов" (wizards) для быстрого макетирования приложений стандартных типов (например, приложение или апплет, сервлет и т.д.). Для создания нового проекта выбираем File/New (Ctrl-N), затем template/Java Classes/Applet или main и т.д. На следующем шаге (Step 2. Target location) "маг" запросит имена класса и пакета. Не забудьте запомнить каталог, в который будут помещены файлы проекта.

ЗАДАНИЕ 1. Создайте в NetBeans апплет, отображающий текущее время. Можно создать эффект тени ("объемное" изображение), выводя строку, содержащую время дважды, различным цветом (например, черным, затем белым), сместив одну строку относительно другой на 1 пиксель вниз и вправо.

II. Движущиеся изображения.

Для решения задачи можно использовать различные техники. Один из простейших подходов - задержка потока, отображающего изображение на заданный промежуток времени. Этого можно достичь с помощью статического метода класса Thread с именем sleep. Этот метод может выбрасывать исключение InterruptedException, поэтому обращение к нему должно располагаться в блоке try. Например, задержка на 500 миллисекунд:

```
…
  try {
        …
        Thread.sleep(500);
        …
  }
  catch(InterruptedException ie)
  {
  }
…
```

ЗАДАНИЕ 2. Создайте апплет, в котором оранжевый шарик (файл orange.gif) перемещается по заданному изображению (файл bg.jpg, размер 200x160 пикселей). Для этого создайте в апплете дополнительный поток. Этот поток должен периодически (с задержкой, например, 200 миллисекунд) менять координаты шарика и перерисовывать весь апплет (метод repaint()). При перерисовке апплета на экран выводится фоновая картинка (bg.jpg), затем шарик. Загрузить изображение (в программе представлено как объект класса Image) можно так:

```
…
import java.awt.*;
…
private Image bg;
…
                    bg = getImage(getCodeBase(),"bg.jpg");
…
```

Отобразить изображение - так:

```
…
  public void paint(Graphics g)
  {
            …
            g.drawImage(bg, x, y, this);
// x, y - координаты левого верхнего угла изображения, для фона 0, 0.
            …
  }
…
```

При работе апплета должно быть заметно мерцание. Причина его в том, что прорисовка происходит медленно. Есть несколько способов борьбы с проблемой, самый простой заключается в предотвращении прорисовки фона апплета. Суть в следующем: когда вызывается метод repaint(), вначале перерисовывается фон, для этого repaint() вызывает метод update() и лишь затем, метод update() вызывает собственно paint(). Если перегрузить update() так, чтобы он лишь вызывал paint() перерисовки фона не

произойдет.

ЗАДАНИЕ 3. Внесите изменения в апплет из Задания 2 для предотвращения перерисовки фона. Добавьте также вывод текущего времени.

ЛАБОРАТОРНАЯ РАБОТА № 11

I. Двойная буферизация.

Более совершенной техникой, которая, в частности позволяет избежать мерцания изображения при его перерисовке, является двойная буферизация. Идея проста - все рисование проводится в буфере, организованном в ОЗУ, а отображение на экране по существу сводится к одной пересылке битов изображения из буфера в ОЗУ в видеопамять. Схема действий такова:

1) Создаем пустое изображение нужного размера в ОЗУ. Для этого вызываем метод класса java.awt.Component (родительский класс, в частности, для java.awt.Applet) - **createImage**(). Метод вернет ссылку на объект Image.

2) С помощью этой ссылки можно получить графический контекст, связанный с изображением. Для этого вызывается метод **getGraphics**(). Последний возвращает ссылку на контекст, т.е. ссылку на объект класса Graphics. Эту ссылку используют для всех изображений апплета.

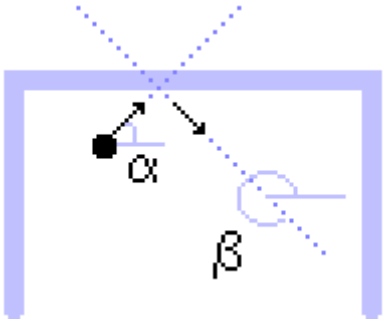
3) Результат (изображение, содержащееся в буфере) пересылается в графический контекст апплета с помощью метода класса Graphics **drawImage**().

Например, для красного квадрата, перемещающегося по фоновому изображению (координаты левого верхнего угла квадрата - sqrX и sqrY задаются отдельным потоком):

```
…
  public void paint(Graphics g)
  {
```

```
            // 1)
            Image buffer = createImage(width, height);
            // 2)
            Graphics gr = buffer.getGraphics();
            …
            gr.drawImage(bgImage,0,0,this);        //     фоновое
изображение
            gr.setColor(Color.red);
            gr.fillRect(sqrX,sqrY,10,10);          //     красный
квадрат 10x10
            …
            // 3)
            g.drawImage( buffer, 0, 0, this );
        }
```

ЗАДАНИЕ 1. Используя двойную буферизацию, создайте апплет, отображающий фоновое изображение и непрерывно и равномерно перемещающийся по нему квадрат. Начальные координаты и вектор скорости квадрата выбирается случайным образом. Квадрат не может покинуть область апплета. Достигнув края, упруго отражается от него и продолжает движение.



Необходимо отдельно рассматривать отражение от каждой "стенки" для разных углов падения фигуры. Например, если углы отсчитываются в направлении против часовой стрелки, то на рисунке ниже, угол, образуемый вектором скорости с положительным направлением оси абсцисс, до отражения равен α, а после отражения $\beta = 360° - \alpha$. Напротив, если бы частица налетала на границу справа, угол изменился бы на +90° ($\beta = \alpha + 90°$). Наконец, если частица движется вверх вдоль оси ординат ($\alpha = 90°$), после отражения угол изменяется на +180° ($\beta = 270°$). Подобное же рассмотрение нужно провести и для остальных границ. Несмотря на то, что движется не точка, а квадрат, можно выбрать отдельную, характерную, точку этого квадрата и следить за нею. Например, если в качестве характерной, выбрана точка, задающая левый верхний угол квадрата, условие нахождения квадрата у правого края разрешенной для движения области будет заключаться в равенстве x-координаты этой точки значению x-координаты границы минус значение размера стороны квадрата.

II. Таймеры.

Java поддерживает программные конструкции, позволяющие организовать циклическое оповещение некоторых компонентов программы об истечении заданного интервала времени - таймеры. Для этого используются возможности, заложенные в классе

javax.swing.Timer и в механизме обработки сообщений Java. Программист создает таймер и указывает интервал времени (в миллисекундах), по истечении которого будет циклически посылаться сообщение об этом событии тем компонентам программы, которые зарегистрированы как "слушатели" (Listeners) такого типа сообщения. Посылка сообщений заключается в вызове специального открытого метода класса-"слушателя". Регистрация "слушателя" - в запоминании ссылки на него в таймере. Различные программные компоненты могут генерировать разные события. Каждое событие имеет свое имя. Например, таймер генерирует событие с именем Action (действие). Это имя используется в названиях всех классов и интерфейсов, отвечающих за обработку события. Во-первых, информация о событии заключена в классе ***имя_события*Event** (например, ActionEvent). Во-вторых, класс, способный обрабатывать событие, должен реализовывать интерфейс ***имя_события*Listener**, в этом интерфейсе объявлен метод ***имя_события*Performed**, и, в-третьих, класс, способный генерировать события (например, Timer), должен получить ссылку на "слушателя", т.е. объект класса, реализующего ActionListener.

Тем самым при работе с таймером используются следующие классы/интерфейсы:

1) тип события - **ActionEvent**; ссылка на объект этого класса передается обработчику события;

2) интерфейс **ActionListener**; класс, содержащий обработчик события должен реализовывать этот интерфейс;

3) собственно обработка события - это вызов метода

public void actionPrformed(ActionEvent ae), объявленного в интерфейсе ActionListener;

4) для таймера должен ссылка на класс реализующий ActionListener gg может быть передана с помощью конструктора.

Например,

```
…
import java.applet.*;
import javax.swing.*;              // класс Timer
…
class TimerTest extends Applet implements ActionListener {
  …
  private Timer timer;
  private int period = 500;          // два раза в секунду
  private final int LIMIT = 1000;
  private int counter = 0;
  …
  public void init()
  {
            …
  // создается таймер, для генерации сообщений
  // через каждые period миллисекунд
  //  "слушатель" сообщения - объект самого класса TimerTest
            timer = new Timer( period, this );
// запускается таймер
            timer.start();
            …
  }
  …
```

```
// Обработчик событий от таймера
  public void actionPerformed( ActionEvent ae )
  {
        if( counter < LIMIT-1 )
        {
              …
              counter++;
              …
        }
        else
        {
              …
              timer.stop();
              …
        }
        …
  }
  …
};
…
```

Для таймеров Java создает отдельный поток вычислений. Кроме того, обработка событий, в нашем случае ведущая к перерисовке апплета, проводится в другой специально созданной нити (AWT Event Queue). Поэтому, даже если мы не создаем потоки явно, работа с таймерами неизбежно является многопоточной.

Начиная с версии 1.3, Java содержит еще один класс Timer в другом пакете - java.util. Этот класс обеспечивает несколько большую функциональность и другой подход к обработке собы-

тия.

ЗАДАНИЕ 2. Создайте класс Sprite для работы с небольшими изображениями, которые можно перемещать по экрану. Класс должен содержать таймер и включать его при запуске объекта этого класса (спрайта). Пользователь класса может задавать изображение (Image), представляющее спрайт, начальные координаты и скорость спрайта, а также область внутри которой спрайт может двигаться (упруго отражаясь от стенок). В самом классе Sprite по сообщениям таймера производится изменение координат спрайта и посылается сообщение владельцу спрайта о необходимости перерисовать себя (repaint()). Кроме того, имеется еще и открытый метод (например, public void show(Graphics g)) для отображения спрайта в текущей позиции. Сам этот метод вызывается из метода paint() владельца спрайта (значит, при создании спрайта владелец должен передать ему ссылку на себя, а спрайт должен сохранить ее в своем поле).

Создайте апплет, владеющий несколькими спрайтами, начальные координаты и скорости которых выбираются случайным образом. Задайте для спрайтов различные изображения (файлы b0.gif, b1.gif, …, b6.gif). Используйте в методе paint() апплета технику двойной буферизации для отображения фоновой картинки (bg.jpg) и всех спрайтов. С помощью HTML-тэга PARAM задайте максимальное время (в миллисекундах) выполнения апплета. Предусмотрите для этого параметра значение по умолчанию (если нет соответствующего параметра или его значение является недопустимым). По истечении указанного времени нужно останавливать таймеры (метод stop() класса Timer).

ЛАБОРАТОРНАЯ РАБОТА № 12.

I. События, связанные с мышью и клавиатурой.

а) MouseEvent (пакет java.awt.event).

Это событие связано с "мышиной активностью", причем причины, приводящие к возникновению события, разбивают на две группы: Mouse и MouseMotion. К первой группе относят события, вызванные изменением одного из следующих состояний мыши:

1) нажата какая-нибудь кнопка мыши (pressed),

2) отпущена кнопка (released),

3) "щелкнули" кнопкой (clicked)

и моменты

4) попадания (entered) курсора мыши в область компонента, "прослушивающего" событие (например, Applet)

и

5) выхода (exited) из этой области.

"Слушатель" (Listener) для первой группы - это MouseListener.

Ко второй группе относятся:

1) "перетаскивание" мышью (dragged), т.е. перемещение курсора мыши при нажатой (левой или правой) кнопке мыши,

и

2) собственно "перемещение" курсора (moved), когда никакая кнопка не нажата.

"Слушатель" (Listener) для этой группы - MouseMotionListener.

Так как причин, вызывающих возникновение события,

связанного с изменением состояния мыши много, то и обработчиков этих событий также много (ровно столько же). Тем не менее, каждый из этих обработчиков получает объект одного и того же класса MouseEvent. Этот объект содержит информацию о состоянии мыши в момент возникновения события. Подробности о том, какая информация сохраняется в объекте MouseEvent, см. в стандартной документации. Наиболее часто используют следующее:

1) информацию о положении курсора мыши. Для этого можно использовать методы класса MouseEvent - getX(), getY() или getPoint(). Первые два возвращают целочисленные значения - координаты курсора мыши соответственно по горизонтали или вертикали. Последний метод - ссылку на объект класса Point. Класс Point - это вспомогательный класс из пакета java.awt, объекты которого содержат два открытых целочисленных поля x и y

```
class Point {
  public int x;
  public int y;
}
```

Т.о. выражение getPoint().x дает то же самое значение, что и getX();

2) определение состояния кнопок мыши. Обычно для этого используют методы класса InputEvent - предка MouseEvent: getModifiers() или getModifiersEx(). Последний позволяет узнать состояние не только кнопок мыши, но и некоторых клавиш-переключателей (Alt, Ctrl, …). Оба метода возвращают целочисленное значение, а в классе InputEvent определены различные статические константы, представляющие значения приписанные различным состояниям кнопок мыши. Например, BUTTON1_MASK -

такое значение вернут методы getModifiers(), если нажата левая кнопка мыши, BUTTON2_MASK - средняя кнопка, BUTTON3_MASK - правая кнопка. Т.о., например, в обработчике события, возникшего при нажатии на кнопку мыши, можно проверить, нажата ли именно левая кнопка мыши:

```
…
  public void mousePressed( MouseEvent me )
  {
        …

        if(me.getModifiers() == MouseEvent.BUTTON1_MASK )
// или InputEvent.BUTTON1_MASK
        {
              // нажата левая кнопка мыши
              …
        }
        …
  }
…
```

b) "Слушатели" и адаптеры для мыши.

Как следует из выше изложенного, интерфейс MouseListener содержит описание пяти обработчиков события MouseEvent:

```
void mousePressed( MouseEvent me );
void mouseReleased( MouseEvent me );
void mouseClicked( MouseEvent me );
void mouseEntered( MouseEvent me );
```

```
  void mouseExited( MouseEvent me );
```

а интерфейс MouseMotionListener - двух обработчиков:

```
  void mouseDragged( MouseEvent me );
  void mouseMoved( MouseEvent me );
```

Соответствующие классы-адаптеры - MouseAdapter и MouseMoutionAdapter, содержат (пустые) реализации этих методов.

Безусловно, нужно зарегистрировать собственный объект - "прослушиватель" событий от мыши. Например,

```
…
import java.applet.*;
import java.awt.event.*;
…
public class Lab12 extends Applet {
  …
  public void init()
  {
        …
        addMouseListener( new MouseAdapter() {
              public void mouseClicked( MouseEvent me )
              {
                    …
              }
              public void mouseReleased( MouseEvent me )
              {
                    …
                    if( me.getModifiers() == MoiseEvent.BUTTON1_MASK )
```

```
                        {
                                ...
                        }
                        ...
                }
        });
        ...
        addMouseMotionListener( new MouseMotionA-
dapter() {
                public void mouseDragged( MouseEvent me )
                {
                        ...
                }
        });
        ...
    }
    ...
}
...
```

c) KeyEvent.

С клавиатурой связаны три события:

1) нажата клавиша (pressed);

2) клавиша отпущена (released);

3) введен символ (typed).

Наличие третьего вида события обусловлено тем, что часто нужно вводить с помощью клавиатуры отдельные Unicode-символы. Это не всегда можно сделать с помощью единичного

нажатия на клавишу (например, Shift+'a' дает 'A'). В общем, "ввод символа" означает, что были нажаты (и отпущены) клавиши, комбинация которых задает некоторый символ. В эту категорию не попадают, например, клавиши Alt, F1 и т.д. Значит нажатие, например, на F10 не приведет к вызову обработчика typed ("ввод символа"). Несмотря на то, что запоминание кодов всего лишь ста с небольшим клавиш клавиатуры и сочетаний некоторых из них с клавишами Alt, Shift, Ctrl и т.п., дело пустяковое, класс KeyEvent содержит объявления статических констант ("виртуальных клавиш") для кнопок клавиатуры. Имена этих констант начинаются с префикса VK_. Например, VK_1, VK_9, VK_A, VK_BACK_SLASH, VK_LEFT, VK_KP_LEFT - представляют коды клавиш, соответственно, '1' (не на цифровой, "numpad", части клавиатуры), '9', 'a', '\', 'стрелка влево' (не numpad), 'стрелка влево' (на цифровой, "numpad", части клавиатуры). См. документацию по KeyEvent.

Чтобы получить код клавиши в обработчике события от клавиатуры, нужно вызвать метод getKeyCode() класса KeyEvent. Сравнивая возвращаемое getKeyCode() целое значение с константами, задающими коды клавиш, определяем, какая клавиша нажата. Делать это есть смысл в обработчиках pressed и released. В обработчике typed метод getKeyCode() всегда возвращает величину, заданную константой VK_UNDEFINED. В этом обработчике обычно вызывают другой метод класса KeyEvent - getKeyChar(). Возвращается величина типа char - введенный Unicode-символ. Метод getKeyChar() можно вызывать в любом обработчике сообщений от клавиатуры. Например, проверяем, нажата ли клавиша 'r'

…

```
public void keyPressed( KeyEvent ke )
{
        …
        if( getKeyChar() == 'r' )
        {
                // нажата 'r'
                …
        }
        // или
        // if( getKeyCode() == VK_R ))
        …
}
…
```

Остальные возможности KeyEvent см. в документации.

d) "Слушатели" и адаптеры для клавиатуры.

Содержит объявление трех методов:

```
void keyPressed( KeyEvent ke );
void keyReleased( KeyEvent ke );
void keyTyped( KeyEvent ke );
```

Соответствующий класс-адаптер - KeyAdapter.

II. Рисование вне метода paint() (в обработчике сообщения).

Метод paint(Graphics g) получает графический контекст (g) извне и все рисование проводит в этом контексте. Любой другой метод апплета может получить такой контекст с помощью метода getGraphics(), например,

```
…
  public void mousePressed( MouseEvent me )
  {
        …
        Graphics g = getGraphics();
        g.setColor( Color.red );
        g.drawLine( 10, 10, 200, 100 );
        …
  }
…
```

Конечно же, все нарисованное таким образом видно до первой перерисовки апплета, т.е. этот способ можно применять для "временных" изображений. Часто приходится комбинировать "постоянное" изображение и изменяющуюся часть, рисуемую поверх него. Тогда в обработчике события вызывается перерисовка, а затем уже дорисовывается переменную часть. Обычный прием для перерисовки апплета - вызов repaint(). Но, этот метод помещает в очередь сообщение на перерисовку… Если при выполнении "задания" возникнут проблемы с перерисовкой, подумайте еще раз, в чем отличие между прямым вызовом метода paint() и обращением к repaint(). Можно ли из обработчика события вызвать paint() напрямую?

ЗАДАНИЕ. Создайте апплет для демонстрации техники рисования с помощью "резиновой нити". Предполагается, что существует некоторая выбранная (текущая) точка в пределах окна апплета. Такая точка задается щелчком (click) левой кнопки мыши. Если пользователь нажимает на левую кнопку мыши и, не отпус-

кая кнопки, перемещает курсор мыши (drag), то от текущей точки до курсора отображается отрезок. Этот отрезок ("резиновая нить") меняет свой наклон и длину, следуя за курсором. Когда пользователь отпускает кнопку мыши (release), отрезок остается в своем последнем положении, а новой текущей точкой становится точка, задающая конец отрезка, связанный прежде с курсором мыши. Таким образом, повторяя процесс, можно рисовать произвольные линии. Менять цвет отрезка можно с помощью клавиатуры. Например, клавиша g задает зеленый, r - красный, b - синий цвет и т.д.

ЛАБОРАТОРНАЯ РАБОТА № 13

I. Классы Frame, Applet и Panel.

Если приложение должно иметь собственное окно и выполняться в графическом режиме (а не в режиме эмуляции консольного терминала), то оно должно создать это окно верхнего уровня явно. Для этого обычно используют класс Frame (или наследник Frame):

```
...
  public static void main(String[} s)
  {
        ...
        Frame frame = new Frame();      // или new Frame("String in title");
        ...
  }
...
```

или

```
...
class MainFrame extends Frame {
  ...
}
...
  public static void main(String[} s)
  {
        ...
        MainFrame frame = new MainFrame();
        ...
```

```
  }
…
или
…
  public static void main(String[} s)
  {
        …
        Frame frame = new Frame() {
              …
        };
        …
  }
…
```

Чтобы обеспечить возможность нормального окончания работы приложения, т.е. дать пользователю возможность закрыть окно, необходимо добавить к фрейму блок прослушивания событий Window. Затем нужно поместить в окно фрейма необходимые компоненты (метод add()). И, напоследок, отобразить фрейм на экране - setVisible(true). Например,

```
…
  private Button button = new Button("Hello");
…
  public static void main(String[} s)
  {
        …
        Frame frame = new Frame("Lab14");
        frame.addWindowListener( new WindowAdapter()
```

```
{
                    public void windowClosing(WindowEvent we)
                    {
                        System.exit(0); // нормальное завершение
                    }
                });
                ...
                frame.add( button );
                ...
                frame.setVisible( true );
        }
    ...
```

Для апплета функции Frame выполняет окно браузера. Сам же класс Applet наследник класса Panel. Panel - это контейнер но не самостоятельный (не верхнего уровня) а вложенный в другой контейнер (например, Frame). Всегда можно рассматривать Applet как Panel, т.е. можно создать класс, расширяющий Applet и, с помощью add(), добавить его во фрейм.

Основные (технические!) отличия приложения от апплета:

1) точка входа в приложение - статический (!) метод main(), а в апплет - метод init();

2) в приложении нужно явно создавать и настраивать Frame, а в апплете ничего этого делать не нужно;

3) в приложении все содержимое программы нужно добавлять в окно фрейма, а в апплете - в окно апплета.

Если поставить задачу написания программы, способной

работать и как приложение и как апплет, то можно использовать подход, часто применяемый при использовании средств визуальной разработки программ. Решаем задачу, не думая о том, как она будет запускаться. Т.е. создаем отдельную "главную" панель (назовем ее, например, mainPanel) как класс, наследник Panel. Все необходимые компоненты размещаем на mainPanel. Создаем метод, например,

```
Panel getMainPanel()
{
        Panel mainPanel = new Panel() {
                ...
        };
        ...
        add(...);
        ...
        return mainPanel;
}
```

в котором создается и настраивается "главная" панель. Затем как в main() так и в init() используем add(getMainPanel()).

II. Кнопки, метки и текстовые поля.

Кнопки (Button), метки (Label) и текстовые поля (TextField) это стандартные компоненты (но не контейнеры) из пакета AWT (Animation Windowing Toolkit). Самый простой компонент - Label - представляет собой текстовую строку (метку), используемую для отображения пояснений к другим элементам управления. Этот компонент не способен реагировать на сообщения, но может ме-

нять ассоциированную с ним строку (метод setText(String msg)) и ее атрибуты (например, цвет текста - метод setForeground(Color color) или цвет фона - метод setBackground(Color color)). Например, в методе init() апплета

```
…
  Label label = new Label("This is label");
  label.setBackground( Color. black );
  label.setForeground( Color. red );
  add( label );
…
```

Все компоненты могут быть в активном или пассивном (недоступном) состоянии. Задавать состояние можно с помощью метода setEnabled(boolean enabled). Если значение параметра метода равно true - компонент переводится в активное состояние. В этом случае он способен реагировать на связанные с ним события (кроме label) и, кроме того, отображается более тусклым серым цветом. Текстовое поле можно использовать для ввода или только для отображения текста. В последнем случае, поле должно быть объявлено неизменяемым (не редактируемым) с помощью метода setEditable(false). В классе TextField определены различные конструкторы, например, TextFiled(String text) и TextField(int charCount) (резервирует место для вывода charCount символов одинаковой ширины) и методы void setText(String msg) и String getText(). Текст на кнопке можно задать с помощью метода setLabel(String text). Например,

```
…
  private Button b1 = new Button("Start");
  private Button b2 = new Button("Stop");
```

```
  private TextField text = new TextField(2);
// пустое поле, способное вмещать до двух символов
…
  private Label label = new Label("Count");
  private int counter;
…
  public void init()
  {
        …
        b1.addActionListener( new ActionListener() {
              public void actionPerformed(ActionEvent ae)
              {
                    …
                    counter++;
                    text.setText(""+counter);
                    …
              }
        });
        b1.addActionListener( new ActionListener() {
              public void actionPerformed(ActionEvent ae)
              {
                    …
              }
        });
        b2.setEnabled( false ); // b2 не может реагиро-
вать на сообщения
        add( b1 );
        add( b2 );
```

```
        text.setText(""+counter);
        text.setEditable( false );
        add( label );
        add( text );
        …
    }
```

Менеджер компоновки пдля определения размеров компонента обращается к его методам Dimension getPrefferedSize() и Dimension getMinimumSize(). Можно переопределить для своего компонента (например, Panel) метод getPrefferedSize() для управления его размером.

ЗАДАНИЕ. Написать программу, которую можно было бы запускать и как приложение и как апплет. Изначально изображение в окне - это фоновая картинка (Canvas) и набор элементов управления: кнопки (Button) "Add Sprite", "Remove Sprite", метка (Label) "Count of sprites" и текстовое поле, которое нельзя редактировать (TextField), содержащее количество спрайтов активных на данный момент. С помощью кнопки "Add Sprite" можно добавлять движущиеся (со скорость выбранной случайным образом) спрайты. Если число спрайтов равно, например, пяти, то кнопка "Add Sprite" становится неактивной (setEnsbled(false)). Подобным же образом, если число спрайтов равно нулю, то кнопка "Remove Sprite" должна стать неактивной. Фоновая картинка и спрайты - из лаб. 11.

ЛАБОРАТОРНАЯ РАБОТА № 14

I. Класс Choice.

Позволяет организовать поле выбора из списка. В поле отображается только одна строка из списка (выбранная строка), но в правой части есть элемент, позволяющий раскрыть весь список и выбрать новую строку. Необходимо создать объект класса Choice, добавить в его список нужное число строк (метод add() или addItem()) и блок прослушивания события Item.

Конструктор:

Choice()

Методы:

void addItem(String s) добавить новую строку к списку

void add(String s) - " -" -

String getSelectedItem() получить выбранную строку

int getSelectedIndex() получить индекс (от нуля) выбранной строки

int getItemCount() получить текущее количество строк в списке

void select(int i) программно выделить i-ю строку

void select(String s) программно выделить строку s

String getItem(int i) получить i-ю строку

Событие при выборе элемента списка:

ItemEvent

Необходимо реализовать интерфейс ItemListener, в котором объявлен один метод - void itemStateChanged(ItemEvent ie).

II. Классы Checkbox и CheckboxGroup.

Класс Checkbox позволяет создавать флажки, т.е. элементы управления, которые могут быть отмечены или нет. Если таких элементов несколько, то они могут быть отмечены независимо друг от друга. Если же несколько объектов Checkbox отнесены к одной группе (CheckboxGroup), то они ведут себя как переключатели (radio buttons), т.е. так, что выбранным (отмеченным) всегда является только один элемент группы.

1) CheckboxGroup.

Конструктор:

CheckboxGroup()

Методы:

Checkbox getSelectedCheckbox() получить выбранный элемент

void setSelectedCheckbox(Checkbox c) программно отметить указанный элемент

2) Checkbox.

Конструктор:

Checkbox()

Checkbox(String s) строка справа от элемента

Checkbox(String s, boolean on) если on == true, элемент отмечен

Checkbox(String s, boolean on, CheckboxGroup g) отнести к группе g

Checkbox(String s, CheckboxGroup g, boolean on)

- " - " -

Методы:

boolean getState() true - выбран, false - нет

void setState(boolean b)

String getLabel() получить метку справа от элемента

void setLabel(String s)

Событие при выборе выборе или отмене выбора элемента:

ItemEvent

Необходимо реализовать интерфейс ItemListener, в котором объявлен один метод - void itemStateChanged(ItemEvent ie).

Например,

```
…
// поля класса
Checkbox cb1, cb2;
Checkbox radio1, radio2;
…
  CheckboxGroup group = new CheckboxGroup();
// элементы с независимой фиксацией (флажки)
  cb1 = new Checkbox("X-Axis",true);
  cb2 = new Checkbox("Y-Axis");
  cb1.addItemListener( this );
  cb2.addItemListener( this );
…
// элементы с зависимой фиксацией (переключатели)
  radio1 = new Checkbox("Show", true, group);
```

```
  radio2 = new Checkbox("Hide", false, group);
…
  radio1.addItemListener( this );
  radio2.addItemListener( this );
…
  public void itemStateChanged(ItemEvent ie)
  {
            …
            Object o = ie.getSource();
            if( o == radio1 || o == radio2 )
            {
                  // событие связано с переключателем
                  …
            }
            if( o == cb1 || o == cb2 )
            {
                  // событие связано с флажком
                  …
            }
  }
…
```

ЗАДАНИЕ. Написать программу, которая позволяет отображать графики некоторых, выбираемых из списка функций f(x) на указанном интервале изменения аргумента x. Т.е. изображение в окне должно содержать две области: график и набор элементов управления: поле выбора функции (Choice) с поясняющей меткой

"Function", два переключателя "Show" и "Hide" (Radio Buttons), с поясняющей меткой "Labels", и два текстовых поля ввода с метками "Xmin" и "Xmax". Изменение параметров должно приводить к немедленному изменению графика функции.

ЛАБОРАТОРНАЯ РАБОТА №16.

Чтобы создать компоненты, необходимо определить класс, 1) реализующий интерфейс Serializable, 2) имеющий конструктор по умолчанию, 3) поддерживающий "правила имен" для свойств и, если обрабатываются сообщения, 4) модель делегирования событий. Класс должен быть упакован в архив (jar-файл) с файлом описания (манифестом), в котором для класса компонента должны быть указаны стандартные учетные записи. Например, если имя класса компонента MyBeanClass, в манифесте должны быть строки:

Name: MyBeanClass.class

Java-Bean: True

Чтобы компонент стал доступен из IDE, необходимо занести его в палитру инструментов. В NetBeans для этого нужно выбрать пункт меню Tools/Install new Java Bean … и, затем, указать jar-файл с компонентом. Для новых компонентов предназначена панель Beans. Теперь можно добавлять его на форму и настраивать так же, как и любой другой известный NetBeans компонент.

NetBeans содержит шаблоны для создания компонентов (в том числе и визуального). При этом сами компоненты автоматически не упаковываются в jar-файлы (чтобы упаковать, необходимо создать отдельный "узел" для архива). Такие компоненты можно добавить в палитру, щелкнув правой кнопкой в области окна редактирования и выбрав Tools/Add to Component Palette...

ЗАДАНИЕ 1. Создайте компонент, представляющий собой панель с тремя метками. Первая метка (upperLabel) занимает

верхнюю часть панели. Вторая (downLabel) - нижнюю часть. Третья (centerLabel) - центр панели. Текст меток upperLabel и downLabel имеет следующие атрибуты: name - "dialog", type - Font.BOLD | Font.ITALIC, size - 10, а метки centerLabel - "dialog", Bold, 18 (см. метод setFont(java.awt.Font f) и конструктор класса Font - Font(String name, int type, int size)). Для текста первой метки должно быть задано горизонтальное выравнивание (см. метод setHorizontalAlignment(int a)) влево, для второй вправо, а для третьей по центру. Предусмотрите такие методы, чтобы можно было задавать цвет фона панели и текст любой метки (все эти параметры - свойства компонента). После закрашивания фона окна в методе paint() не забудьте вызвать paint() базового класса для прорисовки компонентов (в данном случае меток) размещенных на панели. Используйте NetBeans для тестирования (и, быть может, создания) компонента.

Чаще всего компонент способен обрабатывать различные сообщения. Для этого нужно определять и добавлять к компоненту блоки прослушивания. Например,

```
…
  public MyBean extends JComponent {
        …
        public MyBean()
        {
              …
              addMouseListener( new MouseAdapter()
{
                    public void mouseClicked(MouseEvent me)
```

```
                                        {
                                                if( !enabled ) return; // если в
неактивном состоянии
                                                ...
                                        }
                                ...
                        }
                        });
                        ...
        }
    ...
```

Изменение значения свойства - это событие PropertyChangeEvent. Соответствующий интерфейс блока прослушивания - PropertyChangeListener (пакет java.beans). В интерфейсе объявлен один метод - propertyChange(PropertyChangeEvent pce). Событие генерируется в методе set соответствующего компонента. Для генерации используется метод firePropertyChange(String propertyName, Type oldValue, Type newValue), где Type это либо Object, либо примитивный тип. В обработчике события (методе propertyChange()) можно получить переданные при генерации (методом firePropertyChange()) параметры с помощью методов класса PeopertyChangeEvent: Object getNewValue(), Object getOldValue() и String getPropertyName(). Например,

```
...
import javax.swing.*;
import java.bean.*;
...
class SuperBean extends JComponent {
```

```
...
  // свойство
  private int value;
  ...
  public int getValue() { return value; }

  public void setValue(int v)
  {
        int old = value;
        value = v;
        // генерация события
        firePropertyChange(
              "SuperValue",  // имя не обязано совпадать
        с именем свойства
              old,
              value);
  }
  ...
  public SuperBean()
  {
        ...
        // создать и присоединить блок прослушивания со-
бытия PropertyChangeEvent
        addPropertyChangeListener( new PropertyChangeListener() {
              public void propertyChange(PropertyChangeEvent pce)
              {
```

```
                    String pName = pce.getPropertyName();
                    if( pName.compareTo("SuperValue") == 0 )
                    {
                        int oldV = ( (Integer)getOldValue() ).intValue();
                        int newV = ( (Integer)getNewValue() ).intValue();
                        ...
                    }
                    ...
                }
            });
            ...
        }
        ...
    }
    ...
```

ЗАДАНИЕ 2. Создайте компонент "игральная кость", выглядящий как квадрат, внутри которого могут размещаться от одного до шести закрашенных кружков. Цвет кружков должен быть отдельным свойством, цвет фона (области внутри рамки) - другим свойством, а число кружков - третьим. Компонент должен реагировать на щелчок мышью, меняя (случайным образом) значение свойства "число кружков". Кроме того, "кость" должна иметь два состояния: активное и неактивное. В первом обработка событий

(щелчок мышкой) обрабатываются, во втором - игнорируются. Используйте NetBeans для тестирования (и, быть может, создания) компонента.

Если программа строится из набора компонентов, то ее создание с помощью средств визуального программирования или быстрого создания программ выглядит как добавление компонентов на форму и настройка их взаимодействия.

ЗАДАНИЕ 3. С помощью NetBeans создайте игру со следующими правилами:

- играют двое;
- каждый из них по очереди бросает свою собственную игральную кость;
- выброшенное значение определяет число клеток игрового поля, на которое перемещается игрок;
- с каждой клеткой поля связано положительное или отрицательное значение (очки), либо одно из следующих действий - повтор хода, потеря всех набранных очков, окончание игры;
- при каждом перемещении игрока по полю он получает или теряет соответствующее число очков;
- игрок, первым достигший финишной клетки, получает дополнительно 50 очков;
- игра продолжается до тех пор, пока один из игроков не попадет на финишную клетку;
- победил тот, у кого на момент окончания игры больше очков (о чем должно появиться соответствующее сообщение).

Для создания игры используйте компоненты из Заданий 1 и

2. Внешний вид игры должен оказаться подобным следующему: