Лабораторная работа № 2

2.1 Основные графические пакеты Java

Согласно концепции Java – Приложения Java должны работать в любых графических средах (MS Windows, Macintosh и др.). С этой целью были разработаны peer-интерфейсы (peer – родной), содержащие методы работы с графическими объектами в каждой такой среде отдельно.

Библиотека классов Java, основанных на peer-интерфейсах, получила название AWT (Abstract Window Toolkit – абстрактный набор инструментов). При выводе объекта, созданного в приложении Java при помощи peer-интерфейса, на экране создается парный (peer-to-peer) ему объект, который соответствует графической среде операционной системы (например, MS Windows).

В версии JDK 1.1 библиотека AWT была переработана. В нее добавлена возможность создания компонентов, полностью написанных на Java и не зависящих от peer-интерфейсов. Такие компоненты стали называть "легкими" (lightweight). Была создана обширная библиотека "легких" компонентов Java, названная Swing. В ней

были переписаны все компоненты библиотеки AWT, так что библиотека Swing может использоваться самостоятельно, несмотря на то, что все классы из нее расширяют классы библиотеки AWT.

В Java 2 библиотека AWT значительно расширена добавлением новых средств рисования, вывода текстов и изображений, получивших название Java 2D, и средств, реализующих перемещение текста методом DnD (Drag and Drop).

Все эти средства Java 2: AWT, Swing, Java 2D, DnD, Input Method Framework и Accessibility составили библиотеку графических средств Java, названную JFC (Java Foundation Classes).

На рис.14 показана иерархия основных классов AWT. Основу ее составляют готовые компоненты: Button, Canvas, Checkbox, Choice, Container, Label, и др. Если этого набора не хватает, то от класса Canvas можно породить собственные "тяжелые" компоненты, а "легкие" – от класса Container.

Целый набор классов помогает размещать компоненты, задавать цвет, шрифт, рисунки и изображения, реагировать на сигналы от мыши и клавиатуры. На рис. показаны и начальные классы иерархии библиотеки Swing – классы JComponent, JWindow, JFrame, JDialog, JApplet.

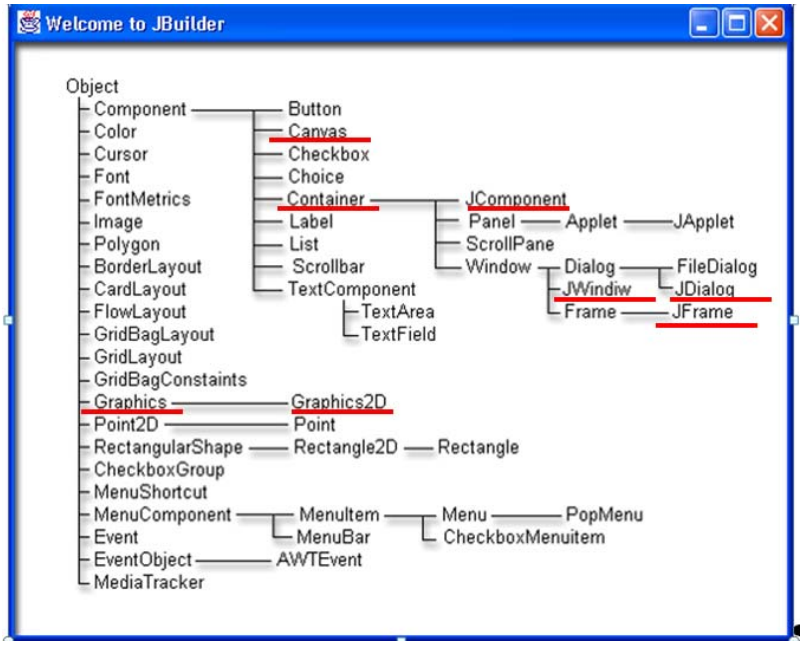


Рисунок 14 – Иерархия классов AWT

Итак, в JDK есть две основные графические библиотеки – это Awt и Swing. Они находятся в пакете Java. Данные библиотеки содержат классы, представляющие компоненты графического интерфейса и графики, а также полный набор методов для работы с ними. Перед их использованием необходимо эти библиотеки импортировать в программу.

Графический интерфейс начинается с создания объекта – Окна класса Frame (из Awt) или JFrame (из Swing).

Класс Frame представляет контейнер верхнего уровня, в котором размещаются другие графические компоненты. Каждый такой компонент перед выводом на экран помещается в свой контейнер. Класс Container является наследником класса Component, являющегося прямым наследником класса Object — вершины иерархии Java классов. От класса Container наследованы 3 основных класса – панель (Panel), панель с прокруткой (ScrollPane) и окно (Window). И уже от класса Window наследуется класс Frame. Таким образом, наследование классов выглядит следующим образом (рис. 15).

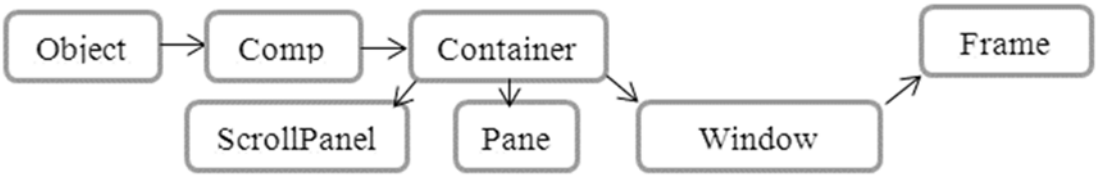


Рисунок 15 – Наследование классов

2.2 Работа с фреймовыми окнами

Класс Frame поддерживает два конструктора:

Frame () – создает стандартное окно, которое не содержит заголовка;

Frame (String заголовок) – создает окно с заголовком, указанным в параметре заголовок.

Методы класса Frame

1. Метод setSize () используется для установки размеров окна. Существует две формы этого метода (с разными списками параметров):

void setSize (int newWidth, int newHeight),
void setSize (Dimension newSize).

Новый размер окна определяется параметрами newWidth и newHeight (первая форма), или полями width и height объекта класса Dimension, передаваемыми параметру newSize. Размеры задаются в пикселах. Если не задать размер окна, то на экране появится окно минимального размера – только строка заголовка.

2. Метод getSize () используется для получения текущего размера окна. Его сигнатура:

Dimension getSize ()

Данный метод возвращает текущий размер окна в полях width и height объекта класса Dimension.

3. Метод setVisible () – показ окна. Сигнатура этого метода имеет вид:

void setVisible (boolean visibleFlag).

Компонент становится видимым, если параметр этого метода получает значение true, иначе он остается скрытым (невидимым).

4. Метод setTitle() – установка заголовка окна. Он имеет следующий формат:

void setTitle(string newTitle), где newTitle – новый заголовок окна.

В качестве примера напишем программу создания окна, которое будет пока пустым.

Задание 1. Создание пустого окна

Создать проект с именем Project_1 и объявите в нем класс с именем TooSimpleFrame (программный код представлен ниже). Он обладает всеми свойствами класса Frame библиотеки Awt, являясь его расширением. В программе создается экземпляр класса TooSimpleFrame, который сохраняется в переменной fr. Устанавливаются размеры окна – 400*150 пикселей методом setSize().

```
import java.awt.*; //импортируем пакет Awt
class TooSimpleFrame extends Frame {

    public static void main(String[] args){ //объявляем метод main
    Frame fr = new TooSimpleFrame();
    fr.setSize(500, 400);
    fr.setVisible(true);
     }
}
```

В программе объявляем класс с именем TooSimpleFrame, являющийся расширение класса Frame (компонент окно) библиотеки Awt. Создаем экземпляр класса TooSimpleFrame, сохранив его данные в переменной fr. Указываем размеры окна fr, используя метод setSize класса Frame. Делаем видимым окно на экране монитора методом setVisible класса Frame (рис. 16).

Рисунок 16 – Результат работы программы. Незакрывающееся окно

Для принудительного закрытия окна применяются средства операционной системы, например, комбинация клавиш <Ctrl>+<C> или диспетчер задач.

Задание 2. Создание закрываемого окна с помощью класса Frame библиотеки Awt

Создать класс FrameTest, позволяющий выводить на экран монитора закрываемое окно пользователя. Класс обладает всеми свойствами класса Frame библиотеки Awt, являясь его расширением. Установить размеры окна – 600*500 пикселей методом setSize ():

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

    public class FrameTest extends Frame {
    FrameTest (String str){
    super(str);
    setSize (600,500); //указываем размеры окна, метод setSize класса Frame
    setVisible(true); //делаем видимым окно, метод setVisible класса Frame
    addWindowListener (new WindowAdapter(){
    public void windowClosing (WindowEvent ev){
    System.exit(0);}});
    }

    public static void main(String [] args) { //объявляем метод main
    new FrameTest ("Моё первое окно"); //создаем экземпляр класса FrameTest со значением строкового параметра Мое первое окно
}
}
```

Импортируем библиотеку Awt и класс Event этой библиотеки, который содержит необходимые методы для обработки событий.

Описываем новый класс с именем FrameTest, который является расширением родительского класса Frame (суперкласса).

Для того, чтобы не переписывать весь сложный код конструктора окна, в программу добавлен конструктор, в котором прописывается инструкция обращении к конструктору суперкласса Frame со строковым параметром str.

Добавлен метод addWindowListener (), реагирующий на действия с окном. В качестве аргумента этому методу передается экземпляр класса WindowAdapter. Этот класс реализует метод закрытие окна windowClosing (), используя статический метод exit () класса System. Результат выполнения программы (рис. 17).

Рисунок 17 – Результат выполнения программы. Закрывающееся окно

2.3 Графические примитивы и методы класса Graphics

Класс Graphics имеет очень большой набор методов для работы с графикой. Наиболее часто используемые методы:

1. drawLine (int x1, int y1, int x2, int y2) – рисует линию из точки с координатами (x1, y1) до точки с координатами (x2, y2).

2. void drawRect (int top, int left, int width, int height) и void fillRect (int top, int left, int width, int height) – отображают соответственно рисованный и заполненный прямоугольник. Координаты левого верхнего угла прямоугольника – в параметрах top и left.

3. void drawRoundRect (int top, int left, int width, int height, int xDiam, int yDiam) и void fillRoundRect (int top, int left, int width, int height, int xDiam, int yDiam) – округленный прямоугольник. Диаметр округляющейся дуги по оси X определяется в xDiam. Диаметр округляющейся дуги по оси Y определяется в yDiam.

4. void drawOval (int top, int left, int width, int height) и void fillOval (int top, int left, int width, int height) – рисование эллипса.

Эллипс рисуется в пределах ограничительного прямоугольника, чей левый верхний угол определяется параметрами top и left, а ширина и высота указываются в width и height. Чтобы нарисовать круг, в качестве ограничительного прямоугольника указывайте квадрат.

5. void drawArc (int top, int left, int width, int height, int начало, int конец) и void fillArc (int top, int left, int width, int height, int начало, int конец) – рисование дуг.

Дуга ограничена прямоугольником. Дуга рисуется от начала до углового расстояния, указанного в конец. Углы указываются в градусах и отчитываются от горизонтальной оси против часовой стрелки. Дуга рисуется против часовой стрелки, если конец положителен, и по часовой стрелке, если конец отрицателен. Например, чтобы нарисовать дугу от двенадцатичасового до шестичасового положения, начальный угол должен быть 90^o и угол развертки 180^o.

6. void drawPolygon (int x[], int y[], int numPoints) и void fillPolygon (int x[], int y[], int numPoints) – фигуры произвольной формы.

Оконечные точки многоугольника определяются координатными парами, содержащимися в массивах x[] и y[]. Число точек в этих массивах, указывается параметром numPoints. Имеются альтернативные формы этих методов, в которых многоугольник определяется объектом класса Polygon.

7. setColor (Color c) – устанавливает текущий цвет, c – объект типа Color.

Для установки текущего цвета нужно в качестве аргумента метода setColor () указать объект типа Color. Можно предварительно создать такой объект конструкцией Color clr = new Color ().

Можно реализовать 7 конструкторов цвета (http://www.realcoding.net/articles/glava-9-graficheskie-primitivy.html?destination=node%2F14164#1). Наиболее удобным является конструктор:

Color (int r, int g, int b), где r – целое число типа int – составляющая красного цвета, g – зеленого, b – голубого. Каждая составляющая имеет длину 1 байт или 256 вариаций (0-255).

Инструкции:

Color pureRed = new Color (255, 0, 0);

Color pureGreen = new Color (0, 255, 0);

определяют чистый ярко-красный pureRed и чистый ярко-зеленый pureGreen цвета.

8. Прямоугольник с закругленными краями: RoundRect (int x, int y, int width, int height, int arcWidth, int arcHeight) http://study-and-dev.com/blog/java_swing_1/ (рис. 18).

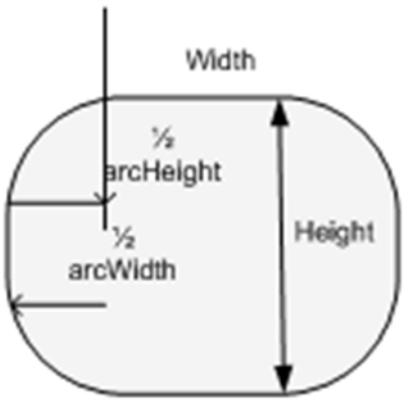


Рисунок 18 – Прямоугольник с закругленными краями

Задание 3. В код программы задания 2 добавьте ниже рассмотренные инструкции, изменяющие цвет фона области рисования

Если вы хотите изменить цвет фона области рисования, то установите новый текущий цвет и начертите им заполненный прямоугольник величиной во всю область фрейма:

```
public void paint(Graphics g){ //объявляем метод paint класса Graphics, g – объектная графическая переменная класса

Color initColor = g.getColor(); //сохраняем исходный цвет фона
g.setColor(new Color(0, 0, 255)); //задаем новый синий цвет фона
g.fillRect(0, 0, getSize().width - 1, getSize().height - 1); //заливаем синим цветом область рисования
/g.setColor(initColor); //восстанавливаем исходный цвет
}
```

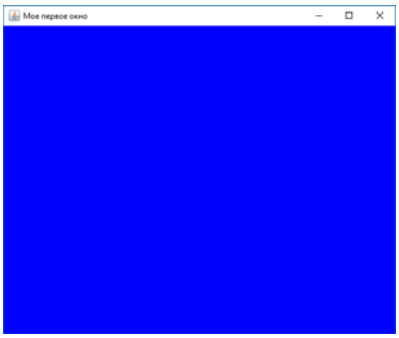


Рисунок 19 – Синий цвет фона

Рассмотрим на примере процесс рисования.

Задание 4. Создание графического объекта

Создайте проект Project_4 и новый класс с именем Gr1. Этот класс будет запускаться при запуске приложения, прорисовывать окно и элементы на нём. Для создания окна рисования используется класс JFrame.java. Программный код класс представлен ниже:

```
import java.awt.*; //импортируем библиотеку Awt
import javax.swing.*; //импортируем библиотеку swing

public class Gr1 extends JFrame {
    Gr1(String s) {//создаем конструктор класса Gr1
    super(s);
    setLayout(null); //отключение менеджера компонентов окна
    setSize(300,500); //указываем размеры окна
    setVisible(true); //делаем окно видимым
    this.setDefaultCloseOperation(EXIT_ON_CLOSE); //закрытие окна
```

```
    }
        public void paint(Graphics my_picture){ //метод paint()
        my_picture.setColor(Color.LIGHT_GRAY);
        my_picture.fillRect(0, 0, 300, 800);
        my_picture.setColor(Color.BLACK);

        my_picture.drawOval(90, 50, 100, 100); //рисуем человечка
        my_picture.drawLine(140, 150, 140, 300);
        my_picture.drawLine(140, 300, 100, 400);
        my_picture.drawLine(140, 300, 180, 400);
        my_picture.drawLine(140, 200, 75, 250);
        my_picture.drawLine(140, 200, 205, 250);
        }
    public static void main(String[] args) {
    new Gr1(""); //создаем экземпляр класса Gr1
    }
}
```

Объявляем класс Gr1, как расширение суперкласса JFrame. Создаем конструктор с именем класса и строковым параметром s – этот параметр используется при обращении к конструктору суперкласса jFrame, который записывает свой аргумент s в строку заголовка окна. Объявляем метод обработки события «закрытие окна». В методе paint() устанавливаем цвет заливки. Заливаем рабочую область светло-серым цветом. Устанавливаем черный цвет чернил и рисуем фигуру человечка.

Графический объект с именем my_picture мы создали с помощью аргумента метода paint**()**, затем в самом методе использовали данный объект для рисования. Объект my_picture наследует все методы класса Graphics, вызывая которые можно рисовать. На рисунке 20 представлен пример кода рассмотренной программы в IDE Eclipse.

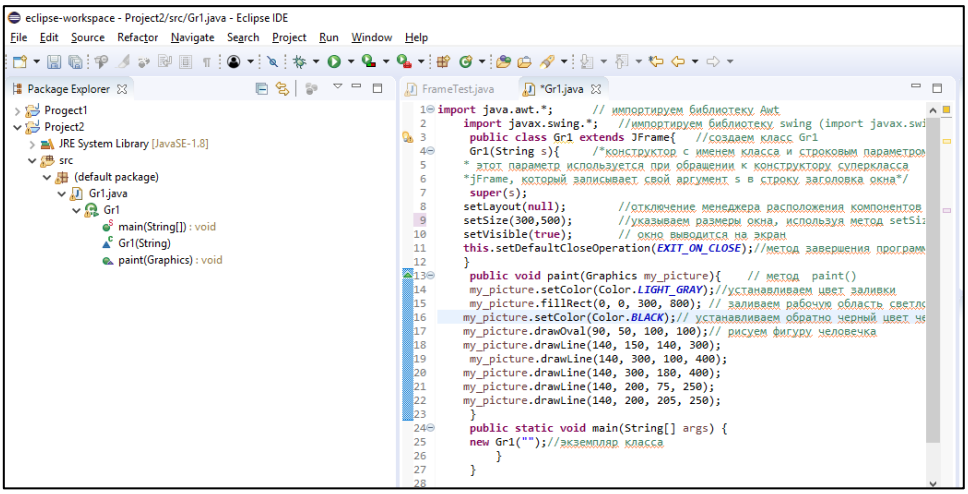


Рисунок 20 – Код рассмотренной программы в IDE Eclipse

Самостоятельное задание 1

Создать Java класс, выводящий на экран, предложенный ниже рисунок (рис. 20).

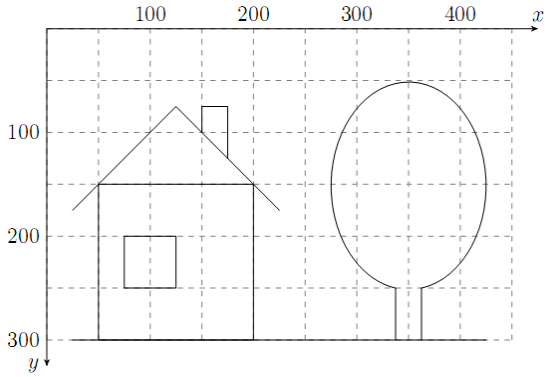


Рисунок 21 – Самостоятельное задание

Лабораторная работа № 3

3.1 Структура и содержание класса Java

Программы на языке Java создаются из классов. Класс представляет собой дискретный модуль со своими атрибутами и методами. Класс определяет базовую структуру объекта и во время выполнения программы обычно создается экземпляр этого класса.

Модификаторы класса

В объявлении класса может употребляться ряд служебных слов модификаторов, придающих классу дополнительные свойства. Эти ключевые слова указываются перед словом class при объявлении:

1. Модификатор public помечает класс признаком общедоступности.
2. Класс, обозначенный модификатором abstract, трактуется как неполный. Создавать экземпляры такого класса запрещено.
3. Класс, определенный как final, не допускает наследования.

Ниже представлена общая структура класса Java:

```
package Имя Пакета;
import Импортируемые Классы;

  доступ class Имя Класса {
  доступ тип Имя Переменной [=начальное значение]; //поля класса

    доступ Имя Класса ([список аргументов]) {//конструктор класса
      утверждения Конструктора
```

```
    }
        доступ Тип Возвращаемого Значения Имя Метода ([аргументы]) {
            <Тело Метода>
        }
}
```

Класс содержит перечень членов (members) как правило, трех типов: поля (fields), методы (methods) и вложенные типы. Вложенные типы – это объявления классов или интерфейсов, размещенные в контексте объявлений других классов или интерфейсов.

Поля класса

Поля (fields) класса – переменные, относящиеся к классу и его объектам. Важно понимать, что значения переменных класса различаются в различных экземплярах этого класса, определяя его состояние. Формат полей (переменных) класса определяется следующим образом:

<Доступ> <Тип данных> <Имя Переменной> [= <Начальное_значение>];

Ниже приведен пример объявления класса Body, объекты которого описывают небесные тела (планеты):

```
class Body { //объявление класса Body

//объявление и инициализация переменных класса

public long idNum; // номер объекта
public String name; //имя объекта
public Body orbits; //переменная, хранящая ссылку на орбиту
public static long nextId = 0; //текущий номер орбиты
 }
```

В объявлении класса содержится ключевое слово class. Объявление класса создает новый тип Body.

Ключевое слово static

Ключевое слово static используется для объявления статической переменной (метода или блока), которая не определяются для конкретного объекта класса. Например, объявленная переменная static int x = 0; означает, что есть только одна переменная x, независимо от того, сколько экземпляров класса существует.

Методы (methods)

Методы (methods) – именованные фрагменты исполняемого кода класса, обуславливающие особенности поведения объектов класса.

Формат объявления методов класса:

```
<Модификатор доступа> <Имя Класса> ([<аргументы>]) {
тело метода
}
```

Существуют две категории методов:

1) *методы-конструкторы*, которые используется только для создания экземпляра класса;

2) *другие методы*, которые используются для реализации какой-либо функции.

3.2 Конструктор класса

Конструктор – это блок выражений, которые используются для инициализации объекта (определения значений переменных). Конструктор указывает, каким должен быть объект.

Конструкторы обладают тем же именем, что и класс, в составе которого они объявляются. За именем следует список (возможно, пустой) параметров и тело конструктора – блок выражений, заключенный в фигурные скобки.

Рассмотрим особенности объявления конструкторов класса

Явное определение конструктора класса без аргументов

Ниже представлен пример инициализации класса Body () с помощью явного объявления конструктора без аргументов и выражений инициализации полей класса:

```
class Body {
public long idNum; //инициализация полей класса Body
public String name = "<Без имени>";
public Body orbits = null;
private static long nextId = 0;

    Body () {//конструктор класса Body без аргументов
    idNum = nextId++;//выражение конструктора
     }
 }
```

Если значения переменных класса не определено, в этом случае они по умолчанию получают: числовые поля класса получают нулевые значения, логические поля – значение false, ссылки – значение null.

Конструктор по умолчанию

Если конструктор не определен в классе, компилятор создает конструктор по умолчанию. Конструктор по умолчанию осуществляет вызов конструктора суперкласса.

Суперкласс

Суперклассом (superclass) для текущего класса (подкласса) (subclass) является исходный для него класс. В Java все классы являются потомками класса Object.

Определение конструктора класса с аргументами

Ниже представлен пример инициализации класса Body () с помощью явного объявления конструктора с аргументами:

```
class Body {
public long idNum; //инициализация полей класса Body
public String name = "<Без имени>";
public Body orbits = null;
private static long nextID = 0;
    //конструктор с аргументами
    Body (String bodyName, Body orbitsAround) {
    name = bodyName;
    orbits = orbitsAround;
    }
}
```

Применение ключевого слова this и super() в конструкторе

Тело конструктора может начинаться с вызова одного из конструкторов суперкласса (родителя), для этого записывается слово super() с параметрами конструктора суперкласса в скобках. Или с вызова другого конструктора этого же класса, для этого используется слово this():

```
Body (String name, Body orbits) {
super();//вызов конструктора суперкласса
}
//вызов другого конструктора этого же класса
Body (String name, Body orbits) {
this ();
this.name = name;
this.orbits = orbits;
}
```

Конструктор – это метод особого назначения с функцией создания экземпляра класса – объекта с определенными значениями переменных. Все экземпляры одного

класса (объекты, порожденные от одного класса) имеют один и тот же набор свойств (атрибутов) и поведение, определяемое методами.

Создание объектов при помощи конструктора без параметров

Создавая объект с помощью оператора new, мы задаем имя соответствующего класса и конструктор без аргументов. Конструкторы указывают, каким должен создаваться экземпляр класса – объект. Сначала объявляется переменная sun, предназначенная для хранения ссылки на объект типа Body. Объект, на который ссылается переменная sun, создается посредством оператора new. В этом случае для объекта выделяется память, а ссылка сохраняется в переменной sun.

```
//создание объекта sun (солнце) класса Body
Body sun = new Body();
```

Пример создания объектов при помощи конструктора с параметрами

```
Body sun = new Body("Солнце", null) ;
Body earth = new Body("Земля", sun);
```

Задание 1. Создание класса, его конструктора и объекта класса

Составить программу в соответствии с пунктами, представленными ниже. Программа представляет собой пример ООП, в котором создается класс, его конструктор и объект класса для вывода на консоль конкретных значений переменных класса. В классе создаются и применяются следующие методы: статический метод *print* для вывода значений переменных объекта класса, методы *Get* с целью получения значения переменных объекта класса, метод *main* для организации вывода значений переменных объекта класса на консоль.

Последовательность действий:

1. Создать проект и присвоить ему имя *ProjectOne* (выберите из главного меню File> New> Java Project ...).

2. Создать пакет. Убедитесь, что в проекте выделена папка src (папка хранения всех файлов проекта) и выберите File> New> Package. В поле Name диалогового окна New Java введите ru.ifmo.intro (имя пакета), выберите папку хранения (src) и нажмите кнопку Finish (рис. 22).

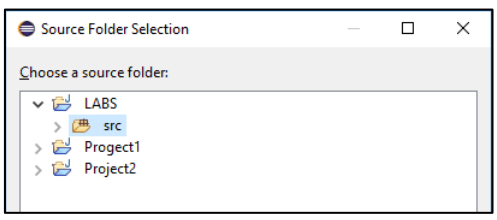


Рисунок 22

3. Создать класс *Person*. Выбрать правой кнопкой мыши образованный ранее пакет и из появившегося контекстного меню выбрать New> Class .... Отметьте public static void main(String [] args), для того, чтобы Eclipse в составе создаваемого класса определил метод main() и нажмите кнопку Finish.

4. Добавление переменных класса. Под заголовком класса добавьте следующий код java для определения переменных класса:

```
private String name;
private int age; //возраст
private int height; //рост
private int weight; //вес
private String eyeColor; //цвет глаз
private String gender; //пол
```

5. Добавить к классу Person два *конструктора*. Напомним, что конструктор – это специальный метод, который используется для создания объектов класса. Введите под строкой public class Person () следующий код, определяющий конструктор:

```
public Person(String name, int age, int height, int weight, String eyeColor,
String gender) {
this.name = name;
this.age = age;
this.height = height;
this.weight = weight;
this.eyeColor = eyeColor;
this.gender = gender;
}
```

Далее в классе введите конструктор по умолчанию (без параметров) следующего вида:

```
public Person() {
super();
}
```

6. Ниже введите код статического метода print следующего вида:

```
public static void print(String str) {
System.out.println(str);
}
```

В коде метода выполняется единственный оператор языка Java, который выводит на экран строковое содержимое, полученное в аргументе String str. Поскольку сигнатура этого метода содержит ключевое слово static, то в дальнейшем из класса Person мы сможем обращаться к нему, просто указывая имя метода (print), а из других

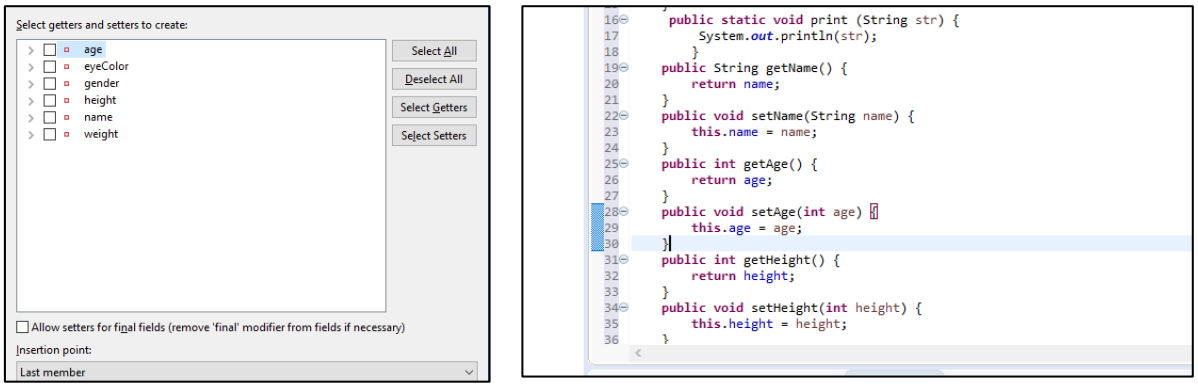
классов, которых пока нет в нашем проекте, по имени класса и имени метода (Person.print).

7. Генерация методов getter и setter.

В ООП не рекомендуется применять прямой доступ к свойствам какого-либо класса из методов других классов. Для этого используется набор методов объекта – интерфейс, посредством которого осуществляется все взаимодействие с этим объектом.

Метод getter используется для получения значений переменных объекта. Метод setter для установки значений переменных объектов.

Eclipse содержит средства создания геттеров и сеттеров переменных, описанных в классе. Для этого выделите созданные переменные класса. Щелкните правой кнопкой мыши в любом месте редактора кода Person и в открывшемся контекстном меню выберите Source> Getters and Setters ... Когда откроется диалоговое окно, нажмите кнопку Select Getters. Выберите Last member (Последний член) в поле со



списком Insertion Point (Точка внедрения) и нажмите кнопку OK.

Рисунок 23 – Создания геттеров и сеттеров переменных объекта

8. Кодирование метода main() – создать экземпляр класса Person, присвоить значение переменным экземпляра, а затем вывести их значения на консоль. Введите код и дополните его:

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

//создается объект с именем p с заданными значениями переменных экземпляра класса
Person p = new Person("James Bond",42,183,82,"Brown", "MALE");

//вывод текущих значений переменных объекта p с помощью метода print
print("Имя: " + p.getName());
print("Возраст :" + p.getAge());
```

```
//вывод значения переменной Height
…
//вывод значения переменной Weight
…
//вывод значения переменной EyeColor
…
//вывод значения переменной Gender
…
}
```

В качестве параметра метода *print* используется конкатенация из литерала типа String в двойных апострофах и значения соответствующей переменной объекта p, полученного с помощью метода getXXX, например,

```
print("Name:"+p.getName());
```

для переменной name.

9. В результате выполнения класса Person будет выведена информация на Console, представленная на рис. 24.

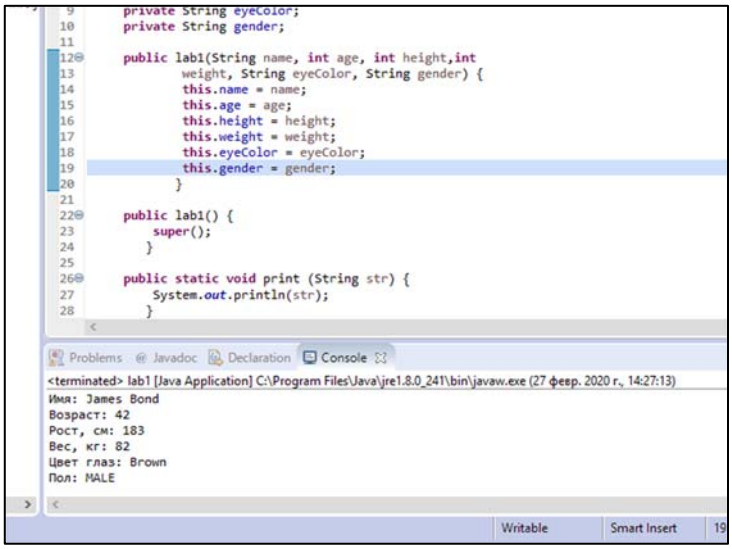


Рисунок 24 – Результат реализации класса Person

Краткая последовательность этапов программирования представлена ниже:

//объявление пакета

//объявление класса

//объявление переменных или полей класса

//объявление конструктора класса с параметрами и присвоение им значений переменных класса

//объявление второго конструктора класса с обращением к конструктору суперкласса

//создание статического метода print с аргументом строкового типа и выражения вывода на экран содержимого строковой переменной str

//применение методов Getters and Setters с целью получения значения переменных класса

//создание метода main

//вызов метода print, выводящего на консоль значения строковой переменной str. Переменная формируется с помощью конкатенации литерала типа String в двойных апострофах и значения соответствующей переменной объекта p, полученного с помощью нестатического метода get.

Лабораторная работа № 4

4.1 Графический пакет AWT

Самой первой графической Java-библиотекой была библиотека AWT (Abstract Window Toolkit). Она была включена в первую версию JDK 1.0. Затем библиотека AWT была дополнена библиотекой Java 2D API, расширяющей возможности работы с двухмерной графикой и изображениями. Архитектура AWT устроена таким образом, что компоненты размещаются в контейнерах (суперкласс java.awt.Container) Помимо графических компонентов, библиотека AWT содержит классы и интерфейсы, позволяющие обрабатывать различные типы событий, генерируемые AWT-компонентами.

Суперклассом, представляющим все AWT-события, является класс java.awt.AWTEvent. Для обработки событий компонента необходимо создать класс-слушатель, реализующий интерфейс java.awt.event.ActionListener, и присоединить его к данному компоненту. Кроме пакетов java.awt и java.awt.event библиотека AWT включает в себя:

Пакет java.awt.color используется для создания цвета.

Пакет java.awt.datatransfer используется для передачи данных внутри приложения и между приложениями.

Пакет java.awt.dnd реализует технологию Drag-and-Drop.

Пакет java.awt.font обеспечивает поддержку шрифтов.

Пакет java.awt.geom реализует двухмерную геометрию.

Пакет java.awt.im обеспечивает поддержку нестандартных методов ввода текста.

Пакет java.awt.image используется для создания и редактирования графических изображений.

Пакет java.awt.print обеспечивает поддержку печати.

Иерархия классов AWT

Класс Component находится наверху иерархии AWT. Component – это абстрактный класс, который инкапсулирует все атрибуты визуального компонента. Объект Component отвечает за запоминание текущих цветов переднего плана и фона, выбранного шрифта текста, а также размеров и местоположения (рис. 25).

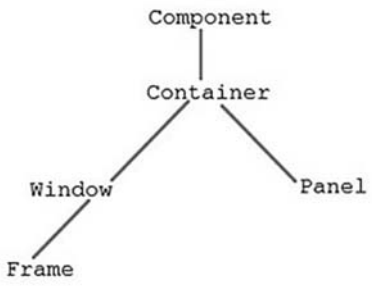


Рисунок 25 – Иерархия классов AWT

Container – это компонент AWT, который содержит другие компоненты, такие как кнопка, текстовое поле, таблицы и т. д. Контейнер является подклассом класса компонентов. Класс контейнера отслеживает и компонует добавляемые компоненты.

Класс Panel – это конкретный подкласс класса Container. Панель не содержит строку заголовка, строку меню или границу. Это контейнер, который используется для содержания компонентов. Класс Window создает окно верхнего уровня. Окно не имеет границ и меню.

Frame является подклассом класса Window и имеет заголовок и границы, а также изменяемый пользователем размер. Таким образом, для создания AWT приложения, в первую очередь, нужно создать объект Frame как окно верхнего уровня.

Мы создаем объект Frame, устанавливаем его размеры и делаем его видимым (см. лабораторную 2). В результате получаем окно с заголовком и кнопками минимизации и закрытия окна. Однако закрыть такое окно мы не сможем (рис. 26). Для этого мы должны добавить слушателя событий окна.

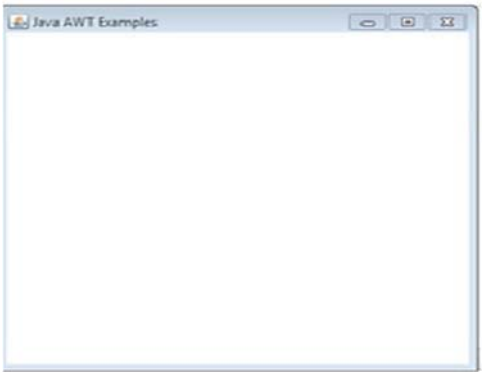


Рисунок 26 – Объект Frame

Так как Frame – это контейнер, мы можем добавлять в него меню и другие элементы графического интерфейса пользователя, кнопки, метки, поля и т. д. Frame-контейнер мы используем для работы с графикой.

Для работы с графикой в Java имеется класс Graphics пакета java.awt. Поскольку класс Graphics является абстрактным, т.к. графический контекст сильно зависит от конкретной графической платформы, создать экземпляры данного класса не удастся. Для простоты работы и снятия необходимости в самостоятельной реализации абстрактных методов, каждая виртуальная машина Java реализует эти методы и создает объекты данного класса методом getGraphics() или с помощью аргумента метода paint(). В лабораторной работе мы использовали метод paint().

4.2 Построение графических примитивов

Класс Frame

Класс Frame поддерживает два конструктора:
Frame() – создает стандартное окно, которое не содержит заголовка;
Frame(String заголовок) – создает окно с заголовком, указанным в параметре заголовок.

Методы класса Frame (см. *лаб. 2*)

1. Метод setSize() используется для установки размеров окна. Существует две формы этого метода (с разными списками параметров).

2. Метод getSize() используется для получения текущего размера окна.

3. Метод setVisible() – показ окна.

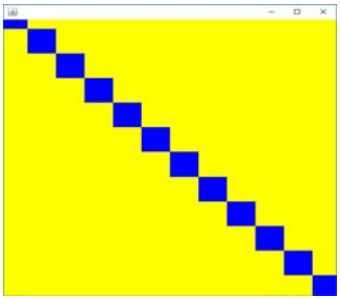
4. Метод setTitle() – установка заголовка окна.

Графические примитивы методы класса Graphics, которые используются для рисования фигур (*см. лаб. 2*).

Далее на примере рассмотрим процесс рисования.

Задание 1. Создание графического объекта

Создать класс статической отрисовки квадратов по диагонали окна Frame:



Этапы разработки программы:

1. Создать проект с именем ProjectFor. Импортировать графические пакеты java.awt и javax.swing.

2. Создать класс *Picture*, как расширение (наследник) класса JFrame пакета java.awt (см. лаб. №2, задание 3). Создать метод static void main(String[] args), формирующий объект класса Picture (), указав в качестве аргумента метода Название окна «Рисунок».

3. Создать конструктор с именем класса и строковым параметром s, который используется при обращении к конструктору суперкласса для записи параметра s в строку заголовка окна (значение параметра задается при создании объекта класса Picture ()).

4. В теле конструктора определить следующие инструкции (см. лаб. №2): 1) обращение к конструктору суперкласса, 2) отключение менеджера расположения компонентов в окне, 3) установление размера окна (600x600, метод setSize класса Graphics), 4) указание вывода на экран создаваемого окна, 5) вызов метода завершения программы после закрытия окна (указать конструкцию this перед методом, так как это метод суперкласса).

5. Создать метод paintS(Graphics g), который реализует условие задачи (статическое построение квадратов). В качестве аргумента метода используется переменная g типа Graphics (класс Graphics пакета awt). Эта переменная содержит графический контекст, включающий окно с панелью на которой имеются инструменты закрытия, свертывания, развертывания окна и с возможностью его названия:

```
g.setColor(Color.YELLOW); //установить желтый цвет (объект Color)
g.fillRect(0, 0, 600, 600); //залить окно установленным цветом
g.setColor(Color.blue);//установить синий цвет (объект Color)

int x1=0; //значение начальной точки по X оси
int y1=0; // значение начальной точки по Y оси
```

```
int x2=600; //размер окна по X
int y2=600; //размер окна по Y

g.drawLine(x1, y1, x2, y2); //прорисовка диагонали окна

int c=50; //длины стороны квадрата
int z= x2*x2+y2*y2; //расчет длины диагонали окна
System.out.println(z); // вывод значения переменной z в консоль

int s= c*c + c*c; //расчет длины диагонали квадрата 50x50
System.out.println(s); //вывод значения переменной s в консоль

int i=0; //счетчик квадратов
g.fillRect (x1, y1, c, c); //прорисовка первого квадратам в 0 точке окна

    while (s*++i<z) {пока сумма диагоналей прорисованных квадратов не
    станет больше длины диагонали окна
    g.fillRect (x1+(i*c), y1+(i*c), c, c);//рисуем следующий квадрат
```

6. Создать метод static void main(String[] args), создающий объект класса Picture ( ), указав в качестве аргумента метода Название окна «Рисунок».

Структура методов класса:

```
класс Picture
метод main (String [] args)
метод paintS (Graphics g)
```

Задание 2. Реализация динамики объекта

Создать на основе программы предыдущего задания класс с именем *PictureD*, реализующий процесс движения (динамики) заданного графического объекта по определенной в программе траектории (для рассмотренного примера – по диагонали). Для этого применить задержку рисования по времени с помощью Метода thread.sleep () (https://javarush.ru/groups/posts/isklyucheniya-java). Метод необходимо оформить следующим блоком:

```
try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        e.printStackTrace();        }
```

В этом случае создается впечатление, что объект (квадрат) движется (в рассмотренном примере по диагонали).

Метод прорисовки объектов назвать paintD (Graphics g).

Структура методов класса:

класс Picture
метод main (String [] args)
метод paintD (Graphics g)

Самостоятельное задание 1

Объединить в одном классе методы, созданные в предыдущих программах (статический и динамический) и реализовать вызов одного из них по условию: если ввести с клавиатуры число 1 – вызывается статический метод и реализуется статическая прорисовка заданного объекта, если ввести с клавиатуры число 2 – вызывается динамический метод и реализуется динамическая прорисовка заданного объекта. Этапы программы:

Создать класс с именем с именем *MyPicture*, импортировать необходимые пакеты. Сформировать метод public void paint(Graphics g), в котором реализуется вызов методов paintD или paintS по условию. Для этого необходимо: импортировать класс Scanner, создать экземпляр этого класса Scanner in = new Scanner(System.in), организовать вывод сообщения в консоль (введите 1 или 2), в зависимости от введенного числа вызвать один из методов (paintS(g) или paintD(g)).

Структура методов класса *MyPicture*:

класс Picture
метод main (String [] args)
метод paint (Graphics g)
метод paintS (Graphics g)
метод paintD (Graphics g)

Лабораторная работа № 5

5.1 Java 2D API

В исходной версии пакета JDK 1.0(Java Development Kit) механизм для рисования фигур выглядел очень просто. Можно было только выбирать необходимый цвет и режим рисования и вызывать методы класса Graphics, вроде drawRect() или fillOval(). API-интерфейс в Java 2D поддерживает гораздо больше возможностей.

Для рисования фигур требуется выполнять следующие действия:

1. Получить объект класса Graphics2D. Этот класс является подклассом класса Graphics. Начиная с версии Java SE 1.2, методы вроде paint() и paintComponent() автоматически получают объект класса Graphics2D. Поэтому остается лишь применить приведение типа, как показано ниже:

```
public void paintComponent(Graphics g){
```

2. Использовать метод setPaint() для указания расцветки(paint). Расцветка подразумевает закрашивание областей, вроде пути штриха или внутренней части фигуры. Она может состоять из одного сплошного цвета, нескольких меняющийся оттенков или мозаичных узоров:

```
Paint paint = ...;
g2.setPaint(paint);
```

В частности, класс Color, GradientPaint и TexturePaint реализуют этот интерфейс.

3. Использовать метод setStroke() для указания штриха(stroke), который должен применяться для прорисовки контура фигуры. Для этого штриха можно выбирать толщину, а также сплошную или пунктирную линию:

```
Stroke stroke =
g2.setStroke(stroke);
```

4. Рисовать или заливать фигуру. Под рисованием подразумевается очерчивание контуров фигуры, а под заливкой - закрашивание ее внутренней части:

```
g2.draw(shape);
g2.fill(shape);
```

Конструкторы класса GradientPaint:

1) GradientPaint(float Xl, float Yl, Color Cl, float X2, float Y2, Color C2)

В двух точках M и N устанавливаются разные цвета. В точке M задается цвет Cl, в точке N – цвет C2. Цвет заливки меняется от Cl к C2 вдоль прямой MN и одновременно прямых, перпендикулярных ей.

2) GradientPaint(float Xl, float Yl, Color cl, float x2, float y2, Color c2, boolean cyclic)

При задании параметра cyclic (циклический) == true цвет начинается с исходного, затем плавно переходит в конечный и в конце снова возвращается в первоначальный.

Конструктор класса TexturePaint:

```
TexturePaint(buffImage, new Rectangle(10,10))
```

Объект buffImage хранит изображение в качестве текстуры закрашиваемой фигуры. Второй параметр задает прямоугольную область 10x10 для объекта buffImage (экземпляр класса Rectangle), которая будет многократно использована для формирования текстуры (этапы создания представлены в примере программы).

5.2 Основные классы графических фигур Graphics2D

Эллипс – класс Ellipse2D.Double(x1, y1, w, h)

Прямоугольник – класс Rectangle2D.Double (x1, y1, w, h)

Линия – класс Line2D.Double (x1, y1, x2, y2)

Дуга – класс Arc2D.Double (x1, y1 w, h, z1, z2, c)

Первые четыре параметра задают верхнюю левую вершину (x,y), ширину и высоту (w, h) прямоугольника, ограничивающей дугу.

z1 – начальный угол, измеренный в градусах, z2 – угол дуги. Начальный угол и угол дуги измеряются относительно прямоугольника, ограничивающего фигуру.

c – константы для указания, как замыкается дуга. Константа Arc2D. PIE указывает, что дуга замыкается путем рисования двух линий. Одна линия соединяет начальную точку дуги и центр прямоугольника, вторая соединяет центр и конечную точку дуги. Константа Arc2D, CHORD приводит к рисованию линии от начальной точки к конечной точке дуги. Константа Arc 2D. OPEN указывает, что дуга не замыкается.

Дуга является частью полного эллипса, который вписывается в прямоугольник. Углы определяются относительно прямоугольника так, что, 45 градусов всегда составляет угол между линией, проведенной от центра эллипса к правой стороне и линией, проведенной от центра эллипса к правому верхнему углу прямоугольника:

Характеристика пера – класс BasicStroke (http://kek.kpfu.ru/EOS/Java/Tutorial_Java_AU/Glava9/Index13.htm)

Основной конструктор

BasicStroke(float w, int c, int join, float miter, float[] dash, float dashBegin)

w – толщина пера в пикселах;

c – оформление конца линии (CAP_ROUND — закругленный конец линии; CAP_SQUARE — квадратный конец линии; CAP_BUTT — оформление отсутствует);

join – определяет скругленность соединения линий (например, в углах прямоугольника) (JOIN_ROUND — линии сопрягаются дугой окружности, JOIN_MITER – линии просто стыкуются);

miter – расстояние между линиями, начиная с которого применяется сопряжение JOIN_MITER;

массив dash – длина штрихов и промежутки между штрихами; элементы с четными индексами задают длину штриха в пикселах, элементы с нечетными индексами — длину промежутка; массив перебирается циклически; индекс dashBegin, начиная с которого перебираются элементы массива dash.

Конструкторы задают некоторые характеристики по умолчанию:

BasicStroke (float w, int c, int join, float miter) – сплошная линия;

На примере рассмотрим применение описанных методов и классов Java2D.

Задание 1. Применение возможностей Java2D

Создать класс Shapes.java, который рисует фигуры Java2D (рис. 27) и демонстрирует возможности графического отображения линий различной толщины, закраску фигур узорами (шаблонами) и изображение пунктирных линий.

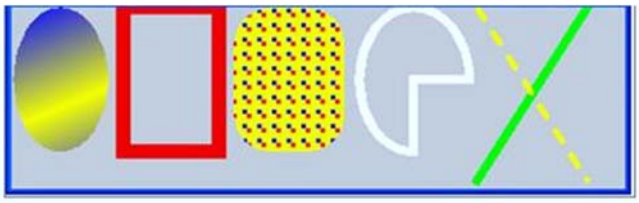


Рисунок 27 – Задание для лабораторной

Этапы разработки программы:

1. Создать проект с именем ProjectFive. Импортировать пакеты java.awt, java.awt.geom, java.awt.image и JFrame.

2. Создать класс Shape, как расширение класса JFrame пакета java.awt.

3. Создать конструктор класса Shape (), в котором создается окно JFrame, пакета javax.swing и выступающее основой для отрисовки заданных графических объектов.

4. В теле конструктора определить следующие инструкции:

1) обращение к конструктору базового класса super(s) (JFrame) для определения параметра s в качестве строки заголовка создаваемого окна. Уточним, что значение параметра s задается при создании ОБЪЕКТА (или экземпляра) класса Shape в главном методе созданного – main(String[] args);

2) отключение менеджера расположения создаваемых в окне объектов, setLayout(null);

3) установление размера создаваемого окна 425x160 – метод setSize класса Graphics;

4) вывод окна на экран – метод setVisible(true); Уточним, что, так как наш класс является расширением класса JFrame, создаваемое окно будет содержать строку заголовка, инструменты для свертывания, закрытия, изменения размеров окна;

5) процедура обработки события завершения программы при закрытии окна, this.setDefaultCloseOperation(EXIT_ON_CLOSE). Конструкция this перед методом указывает на его принадлежность к базовому классу – классу JFrame;

5. Создать метод, который реализует условие задачи – public void paint (Graphics g). В качестве аргумента метода указан графический контекст g (объект класса Graphics, который содержит все необходимое для рисования.

6. Осуществить вызов метода для правильной отрисовки дочерних легких компонентов пакета swing – super.paint(g).

7. Выполнить приведение типа Graphics графического контекста g к типу Graphics2D и присвоить его переменой ссылочного типа g2d для получения доступа к возможностям Java2D – Graphics2D g2d = (Graphics2D) g.

Рисуем овал с градиентной заливкой:

8. Вызвать метод setPaint (класс Graphics2D) для графического объекта g2d, указав в качестве аргумента объект (экземпляр) класса GradientPaint. Конструктор данного объекта определяет тип градиентной заливки.

9. Выполнить заливку овала градиентом: g2d.fill(new Ellipse2D.Double(5, 30, 65, 100));

Рисуем прямоугольник с красным контуром:

10. Вызвать метод setPaint для присвоения значения Color.red объекту Paint: g2d.setPaint(Color.red);

11. Вызвать метод setStroke (класса Graphics2D) для графического объекта g2d, указав в качестве аргумента объект (экземпляра) класса BasicStroke. Конструктор данного объекта определяет толщину линии в 10 пикселов (10.0f).

12. Нарисовать контур прямоугольника, вызвав метод draw для графического объекта g2d, указав в качестве аргумента объект (экземпляр) класса Rectangle2D.Double. Конструктор данного объекта определяет координаты прямоугольника (80, 30, 65, 100).

Рисуем прямоугольник со сглаженными углами, заполненный шаблоном

13. Создать буфер buffImage – объект класса BufferedImage из пакета java.awt.image размером 10x10 пикселей и цветовую схему RGB: BufferedImage buffImage = new BufferedImage(10,10, BufferedImage.TYPE_INT_RGB);

14. Создать объект gg типа Graphics2D, который можно использовать для занесения графики в buffImage – Graphics2D gg = buffImage.createGraphics();

15. Заполнить графический объект буфера фигурой, которая будет служить образцом текстуры:

```
gg.setColor( Color.yellow );
gg.fillRect( 0, 0, 10, 10 );
gg.setColor( Color.blue );
gg.fillRect( 1,1,3,3);
gg.setColor( Color.red) ;
gg.fillRect( 4,4,3,3) ;
```

16. Вызвать метод setPaint (класса Graphics2D) для графического объекта g2d, указав в качестве аргумента объект (экземпляр) класса TexturePaint. Первый параметр конструктора данного объекта определяет изображение, хранящееся в объекте buffImage, в качестве текстуры закрашиваемой фигуры. Второй параметр задает прямоугольную область 10x10 для объекта buffImage (экземпляр класса Rectangle), которая будет многократно использована для формирования текстуры – g2d.setPaint(new TexturePaint(buffImage, new Rectangle(10,10)));

17. Выполнить заливку прямоугольника со сглаженными углами полученной текстурой, указав следующие координаты (155,30,75, 100,50,50) (см. п.9);

Рисуем белый сектор

18. Вызвать метод setPaint для присвоения значения Color.White объекту Paint (см. п. 10).

19. Вызвать метод setStroke (класса Graphics2D) для графического объекта g2d, указав в качестве аргумента объект (экземпляр) класса BasicStroke. Конструктор данного объекта определяет толщину линии в 6 пикселов (6.0f).

20. Нарисовать контур сектора, вызвав метод draw для графического объекта g2d, указав в качестве аргумента объект (экземпляр) класса Arc2D.Double. Конструктор данного объекта определяется координатами дуги (240, 30, 75, 100, 0, 270, Arc2D.PIE).

Рисуем зеленую линию

21. Вызвать метод setPaint для присвоения значения Color.Green объекту Paint (см. п. 10).

22. Нарисовать линию, вызвав метод draw для графического объекта g2d, указав в качестве аргумента объект (экземпляр) класса Line2D.Double. Конструктор данного объекта определяет координаты линии (395, 30, 320, 150).

23. Определить двухэлементный массив float dashes[]={10}; Этот массив описывает длину штрихов и расстояний между штрихами в пикселах.

Рисуем желтую линию

24. Вызвать метод setPaint для присвоения значения Color.yellow объекту Paint (см. п. 10).

25. Вызвать метод setStroke (класса Graphics2D) для графического объекта g2d, указав в качестве аргумента объект (экземпляр) класса BasicStroke. Конструктор объекта:

```
g2d.setStroke(new BasicStroke(4, BasicStroke.CAP_ROUND,
BasicStroke.JOIN_ROUND, 10, dashes,0));
```

28. Нарисовать линию, вызвав метод draw для графического объекта g2d, указав в качестве аргумента объект (экземпляр) класса Line2D.Double. Конструктор данного объекта определяет координаты линии (320, 30, 395, 150).

29. Создать метод public static void main(String[] args), формирующий объект класса Shape (), указав в качестве аргумента метода Название окна «Изображение двухмерных фигур».

Лабораторная работа № 6

6.1 Линейный и радиальный градиент, заливка градиентом, текстура

Принцип формирования простого градиента заключается в следующем:

1. В точке A (x1, y1) задается цвет C1. В точке B (x2, y2) задается цвет C2. Цвет заливки плавно меняется вдоль отрезка AB, по направлению от A к B. Направление изменения цвета и граничные точки отрезка образуют вектор градиента.

2. Вне отрезка AB цвет не меняется. До точки A остается цвет C1, после точки B – цвет C2. Такую заливку создает конструктор:

```
GradientPaint (float x1, float y1, Color c1, float x2, float y2, Color c2)
```

Для заливки фигуры необходимо: создать объект класса GradientPaint, установить градиент в качестве текущего цвета, залить фигуру градиентом. Например,

```
//создание grd-объекта класса GradientPaint с определенным цветовым
градиентом
```

```
GradientPaint grd = new GradientPaint (20.0f, 20.0f, Color.BLUE, 200.0f, 20.0f, Color. YELLOW);
```

формируется переход цвета из синего в желтый, начало вектора градиента (20.0f, 20.0f), конец вектора – (200.0f, 20.0f), значения координат – вещественные числа

```
//установка градиента в качестве текущего цвета графического контекста g2d фреймового окна
g2d.setPaint (grd);
//заливка созданным градиентом прямоугольной области g2d-графического контекста фреймового окна (или проще - прямоугольника)
g2d.fill (new Rectangle2D. Float (20.0f, 20.0f, 200.0f, 100.0f));
```

или другой вариант инструкций (без объявления и инициализации переменной grd типа GradientPaint:

```
g2d.setPaint (new GradientPaint (20.0f, 20.0f, Color.BLUE, 200.0f, 20.0f, Color. YELLOW));
g2d.fill (new RoundRectangle2D.Float (20.0f, 20.0f, 200.0f, 100.0f));
```

Технически переход из одного цвета в другой выполняется так.

Создается слой, залитый первым цветом. Поверх него накладывается слой, залитый вторым цветом. Альфа (прозрачность) первого слоя линейно уменьшается от 1 до 0 в направлении вектора градиента. Альфа второго слоя линейно возрастает в этом же направлении.

Таким образом, на всем протяжении вектора сумма альфы двух слоев всегда равна единице, но происходит смешение цветов в разной пропорции на протяжении вектора. Следует понимать, что происходит именно переход из одного цвета в другой через смешение, а не перебор участка спектра между этими цветами. Поэтому переход от белого цвета к черному цвету будет выполнен градациями серого, а не спектром всех цветов.

Конструктор циклического градиента:

```
GradientPaint (float x1, float y1, Color c1, float x2, float y2, Color c2, boolean cyclic)
```

создает заливку, которая повторяется циклически по всей фигуре, если параметр cyclic имеет значение true, а длина фигуры больше, чем длина вектора градиента.

```
//создание grd-объекта класса GradientPaint с определенным цветовым градиентом
grd = new GradientPaint (20.0f, 170.0f, Color. YELLOW, 106.0f, 170.0f, Color. BLUE, true);
```

```
//установка градиента в качестве текущего цвета графического контекста
g2d фреймового окна
g2d.setPaint (grdPaint);
//заливка созданным градиентом прямоугольника
g2d.fill (new Rectangle2D. Float (20.0f, 170.0f, 440.0f, 50.0f));
```

формируется переход цвета из синего в желтый, 20,170 и 106,170 – координаты вектора циклического градиента (т.е. длина вектора 86 пикселов), 420 пикселов длина прямоугольника, т.е. в прямоугольнике реализуется почти 5 циклов градиентной заливки)

или другой вариант инструкций (без объявления и инициализации переменной grd типа GradientPaint:

```
g2d.setPaint (new GradientPaint (20.0f, 170.0f, Color. YELLOW, 106.0f, 170.0f, Color. BLUE, true));
g2d.fill (new Rectangle2D. Float (20.0f, 170.0f, 440.0f, 50.0f));
```

Конструктор линейного градиента:

Класс LinearGradientPaint может создавать градиентную заливку из нескольких цветовых диапазонов, вектор градиента AB делится на несколько частей точками. Длина вектора считается равной единице, точки располагаются в диапазоне от 0.0f до 1.0f. Расположение этих точек заносят в массив. Во второй массив заносят набор соответствующих цветов, например,

```
//массив опорных точек линейного градиента
    float [] base = {0.0f, 0.5f, 1.0f};
//массив цветов
    color = {Color.YELLOW, Color.BLUE, Color.GREEN};
```

//создание lgrd-объекта класса GradientPaint с определенным линейным градиентом

```
LinearGradientPaint lgrd = new LinearGradientPaint (0.0f, 150.0f, 200.0f, 150.0f, base, color);
//установка линейного градиента в качестве текущего цвета
    g2d.setPaint (lgrd);
//заливка закругленного прямоугольника
    g2d.fill (new RoundRectangle2D.Float (20.0f, 20.0f, 200.0f, 100.0f, 30.0f, 30.0f));
```

или другой вариант установки линейного градиента в качестве текущего цвета (без объявления и инициализации переменной lgrd типа LinearGradientPaint):

g2d.setPaint (new LinearGradientPaint (0.0f, 150.0f, 200.0f, 150.0f, base, color));

В этом примере от начала градиента (0.0f) до середины (0.5f) задан переход от желтого цвета к синему, а от середины до конца (1.0f) задан переход от синего к зеленому.

Аналогичным образом на основе массивов опорных точек и цветов можно создать радиальную заливку.

Конструктор радиального градиента

Класс RadialGradientPaint создает радиальную заливку, по направлению вдоль радиуса окружности от центра к краю. Вектор градиента AB делится на несколько частей точками, например,

RadialGradientPaint rgrd = new RadialGradientPaint (295.0f, 225.0f, 75.0f, base, color);

Параметрами конструктора радиальной заливки являются координаты центра окружности, длина радиуса окружности, массив точек, массив цветов.

Конструктор текстурной заливки представлен в предыдущей лабораторной работе.

Напомню, что текстурная заливка формируется при помощи следующих приемов:

– для хранения одиночного элемента текстуры создается объект буфера класса BufferedImage (буферное изображение) и его графический контекст.

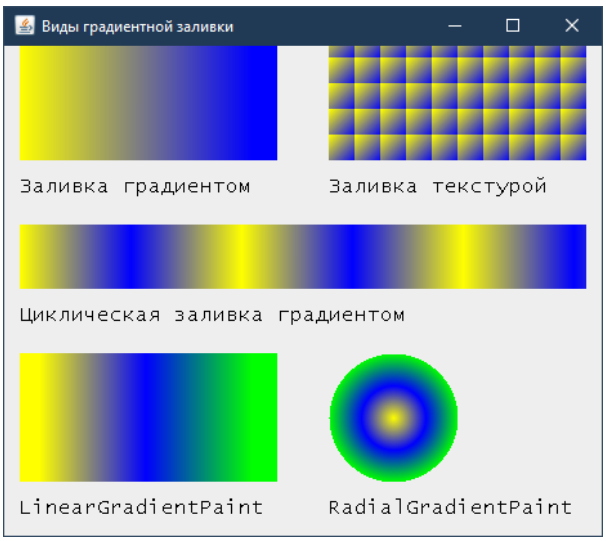
– создается рисунок текстуры буферного изображения (которая не отражается в окне фрейма).

– создается объект класса TexturePaint, в конструкторе которого указывается ссылка на буферное изображение, задается размер области буфера, которая будет многократно использована для формирования текстуры исходного объекта. При этом размеры области могут отличаться от размера буферного изображения.

– выполняется заливка фигуры полученной текстурой.

Задание 1. Применение градиента и текстуры

Создать проект ProjectSix. Написать и отладить программу GradientFrame.java, которая создает различные виды градиентной заливки Java2D:



Этапы разработки программы:

1. Импортировать пакеты java.awt, java.awt.geom, java.awt.image и JFrame.

2. Создать класс GradientFrame, как расширение класса JFrame пакета java.awt

3. Создать конструктор класса (см. предыдущую работу), который осуществляет: обращение к конструктору базового класса super(s) для передачи параметр s в качестве строки заголовка окна, отключение менеджера расположения компонентов в окне, установку размера фреймового окна (480x420), визуализацию окна, метод обработки события завершения программы и закрытия выполним заливку прямоугольника со сглаженными углами полученной текстурой. окна.

4. @Override //для переопределения методов класса JFrame, программа работает и без него, так как мы ничего не переопределяем.

6. Создать метод paint класса Graphics, который автоматически формирует графический контекст (g) окна JFrame, позволяя реализовать условие задачи (прорисовку и заливку заданных объектов).

7. Добавить метод super.paint (g), необходимый для правильной отрисовки легких графических компонентов пакета swing.

8. Преобразовать тип Graphics g-контекста к типу Graphics2D и присвоить ссылку на него переменой g2d (g2d-контекст).

9. Выполнить заливку 1 прямоугольника простым градиентом от желтого цвета к синему, для этого:

– создаем градиент как grd-объект класса GradientPaint с указанием следующих значений параметров – (20.0f, 20.0f, Color.YELLOW, 200.0f, 20.0f, Color.BLUE). 20,20 и 200,20 – координаты вектора градиента, см. рис. 28;

– с помощью метода setPaint устанавливаем текущий цвет заливки g2d-контекста окна, указав в качестве параметра grd-градиент;

– заливаем этим цветом прямоугольную область g2d-контекста окна (метод fill), указав в качестве параметра метода объект класса Rectangle2D.Float со следующими значениями конструктора – (20.0f, 20.0f, 200.0f, 100.0f);

10. Выполнить заливку 2 прямоугольника циклическим градиентом от желтого цвета к синему, для этого:

1) повторяем выше рассмотренные пункты, но для создаваемого градиента указываем следующие значения параметров – (20.0f, 170.0f, Color.YELLOW, 106.0f, 170.0f, Color. BLUE, true). 20,170 и 106,170 – координаты вектора циклического градиента, см. рис. 28;

2) заливаем созданным градиентом прямоугольную область g2d-контекста окна со следующими значениями конструктора объекта класса Rectangle2D.Float – (20.0f, 170.0f, 440.0f, 50.0f):

– снова создаем градиент как grd-объект класса GradientPaint с указанием следующих значений параметров – (20.0f, 170.0f, Color.YELLOW, 106.0f, 170.0f, Color. BLUE, true). 20,170 и 106,170 – координаты вектора циклического градиента, см. рис 28;

– снова с помощью метода setPaint устанавливаем текущий цвет заливки g2d-контекста окна, указав в качестве параметра grd-градиент;

– заливаем этим цветом прямоугольную область g2d-контекста окна (метод fill), указав в качестве параметра метода объект класса Rectangle2D.Float со следующими значениями конструктора – (20.0f, 170.0f, 440.0f, 50.0f);

11. Выполнить заливку 3 прямоугольника текстурным шаблоном в виде градиента, для этого:

– создаем буферное изображение (которое не отображается в окне) размером 20x20 пикселей – как b-объект класса BufferedImage со значениями параметров его конструктора (20, 20, BufferedImage.TYPE_INT_RGB);

– с помощью метода createGraphics () для b-буфера создаем графический контекст, преобразовав Graphics-тип к типу Graphics2D. Присваиваем ссылку на него переменой gb (gb-контекст буфера) – Graphics2D gb = b.createGraphics ();

– создаем нужный нам grd1-градиент – как объект класса GradientPaint со значениями параметров (0.0f, 0.0f, Color.YELLOW, 20.0f, 20.0f, Color.BLUE), направление вектора из точки с координатой 0,0 в точку с координатой 20,20 (диагональный градиент);

– при помощи метода setPaint устанавливаем для gb-контекста буфера текущую заливку с указанием в качестве параметра созданный grd1-градиент;

– в буфере создаем область текстуры размером 20x20, как tp-объект класса TexturePaint– TexturePaint tp = new TexturePaint (b, new Rectangle2D. Float (0.0f, 0.0f, 20.0f, 20.0f));

– устанавливаем текущий цвет g2d-контекста окна в соответствии с созданной текстурой – g2d.setPaint (tp);

– заливаем этим цветом прямоугольную область g2d-контекста окна (метод fill), указав в качестве параметра метода объект класса Rectangle2D.Float со следующими значениями конструктора – (260.0f, 20.0f, 200.0f, 100.0f);

12. Создаем многоцветную линейную градиентную заливку, для этого:

– создаем массив опорных точек {0.0f, 0.5f, 1.0f};

– создаем массив цветов {Color. YELLOW, Color. BLUE, Color.GREEN};

– создаем lgrd-объект класса LinearGradientPaint с указанием значений параметров конструктора (35.0f, 270.0f, 200.0f, 270.0f, base, color);

– методом setPaint устанавливаем текущий цвет заливки g2d-контекста окна, указав в качестве параметра градиент lgrd;

– заливаем этим цветом прямоугольную область g2d-контекста окна (метод fill), указав в качестве параметра метода объект класса Rectangle2D.Float со следующими значениями конструктора – (20.0f, 270.0f, 200.0f, 100.0f);

13. Создаем многоцветную радиальную градиентную заливку, для этого повторяем выше рассмотренные пункты, указав: 1) в качестве значений параметров конструктора rgrd-объекта класса RadialGradientPaint значения – (310.0f, 320.0f, 50.0f, base, color); 2) в качестве параметра метода fill (заливка) объект класса Ellipse2D. Float со следующими значениями конструктора – (20.0f, 270.0f, 200.0f, 100.0f);

14. Создаем подписи рисунков:

```
g2d.setPaint (Color. BLACK);
        g2d.setFont (new Font ("Lucida Console", Font.PLAIN, 16));
        g2d. drawString ("Заливка градиентом", 20, 145);
        g2d. drawString ("Заливка текстурой", 260, 145);
        g2d. drawString ("Циклическая заливка градиентом", 20, 245);
        g2d. drawString ("LinearGradientPaint", 20, 395);
        g2d. drawString ("RadialGradientPaint", 260, 395);
```

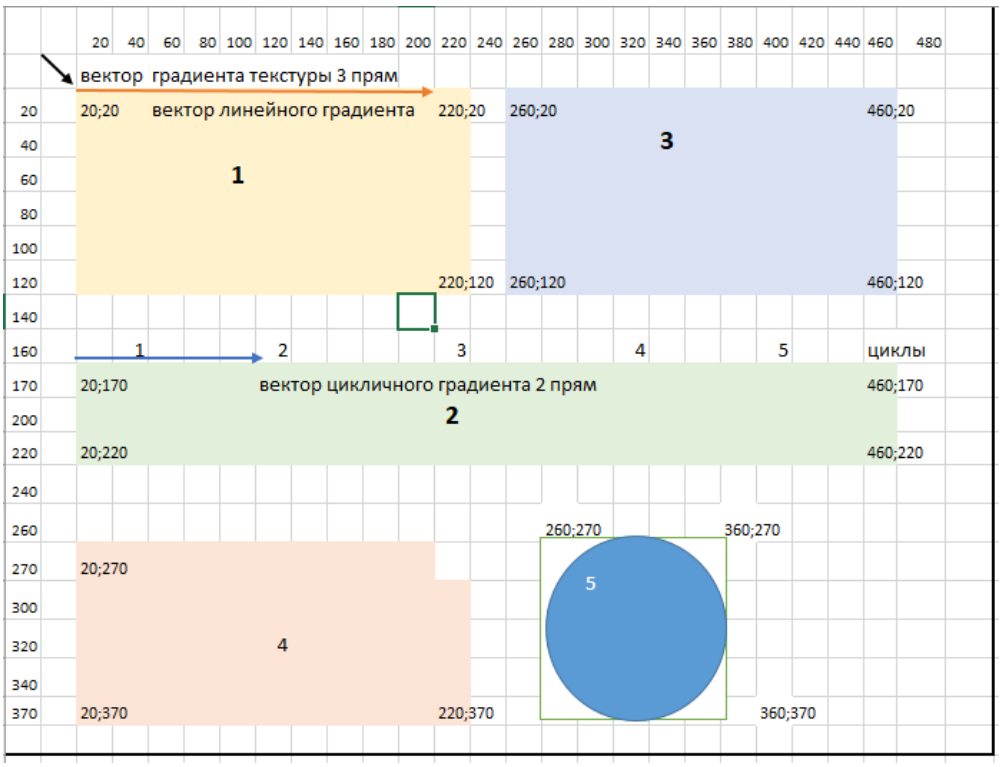


Рисунок 28 – Координаты фигур