ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ В РАЗРАБОТКУ ПРОГРАММ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ JAVA

Язык Java — это объектно-ориентированный язык программирования. Программы написанные на Java могут выполняться на различных операционных системах при наличии необходимого ПО - Java Runtime Environment. Для того чтобы создать программу на языке Java необходимо следующее ПО:

- Java Development Kit (JDK).
- Java Runtime Environment (JRE).
- Среда разработки. Например, NetBeans или IDE IntelliJ IDEA.

Установка ПО

Для того, чтобы скачать ПО, можно воспользоваться следующими ссылками:

1. Программа “IntelliJ IDEA”:https://www.jetbrains.com/idea/download/#section=windows

2. Программа “NetBeans IDE”: https://netbeans.org/downloads/

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

По умолчанию, скаченный JDK установится в папку с таким адресом: C:\Program Files\Java

Начало работы с программой

После установки одной из сред разработки (“IntelliJ IDEA” или “NetBeans IDE”) можно начать создавать проекты. Далее будет показано, как начать новый проект на примере программы «IntelliJ IDEA».

1. В открытом окне программы выбираем “Create New Project”.

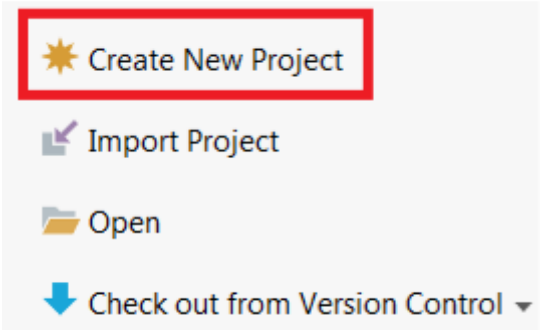

Рисунок 1

2. Щёлкаем «New», чтобы загрузить JDK.

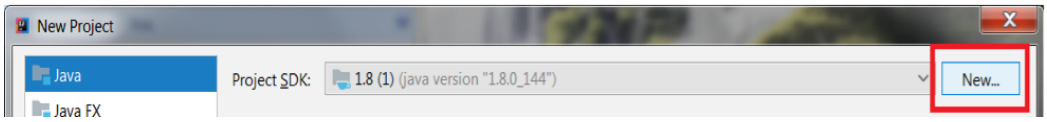

Рисунок 2

3. Из выпадающего списка папок выбираем «Program Files».

Рисунок 3

4. В «Program Files» выбираем папку «Java».

Рисунок 4

5. Далее выбираем папку «jdk...».

Рисунок 5

6. Затем дважды нажимаем «Next» в нижнем правом углу.

Рисунок 6

7. Выбираем название для будущего проекта и затем нажимаем кнопку «Finish».

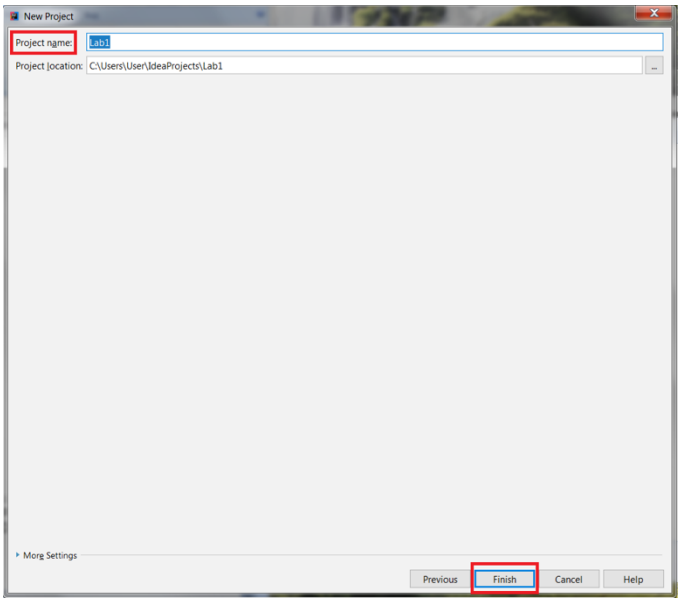

Рисунок 7

8. Щёлкаем правой кнопкой мыши по папке «src» и создаем новый пакет.

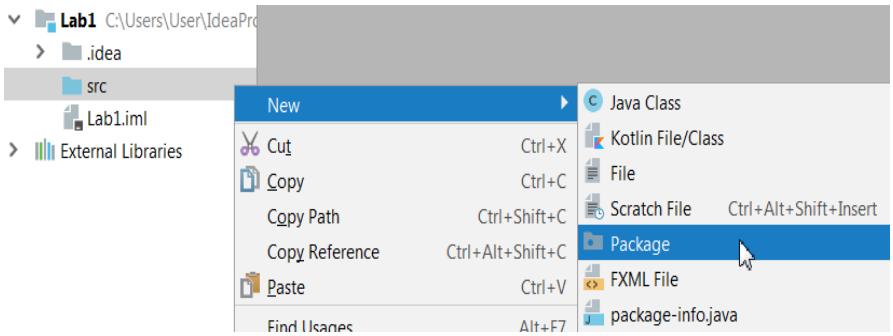

Рисунок 8

9. Вводим название пакета. «Package» – это оператор, который сообщает транслятору, в каком пакете должны определяться содержащиеся в данном файле классы.

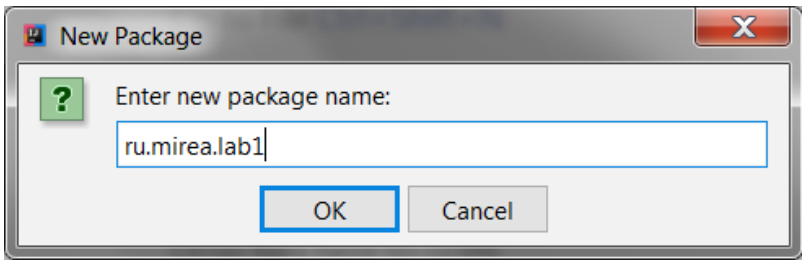

Рисунок 9

10. Щелкаем по созданному пакету правой кнопкой мыши и создаем новый класс.

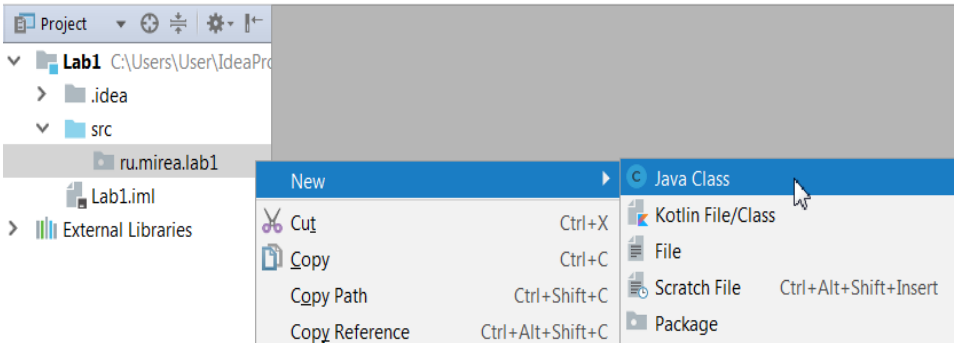

Рисунок 10

11. Новый проект создан. Теперь можно приступать к написанию кода.

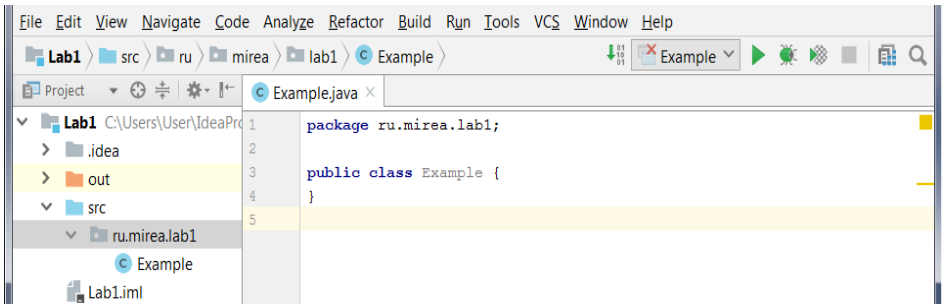
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Lab1 src ru mirea lab1 Example
Example
Project
Example.java
Lab1 C:\Users\User\IdeaPr
.idea
out
src
ru.mirea.lab1
Example
Lab1.iml
1 package ru.mirea.lab1;
2
3 public class Example {
4 }
5

Рисунок 11

ПРАКТИЧЕСКАЯ РАБОТА №1. КЛАССЫ, КАК НОВЫЕ ТИПЫ ДАННЫХ. ПОЛЯ ДАННЫХ И МЕТОДЫ

Цель работы

Цель данной практической работы – освоить на практике работу с классами на Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Понятие класса

В Java, класс является определением объектов одного и того же вида. Другими словами, класс — это тип данных, создаваемый программистом для решения задач. Он представляет из себя шаблон, или прототип, который определяет и описывает статические свойства и динамическое поведение, общие для всех объектов одного и того же вида.

Экземпляр класса - реализация конкретного объекта типа класс. Другими словами, экземпляр экземпляра класса. Все экземпляры класса имеют аналогичные свойства, как задано в определении класса. Например, вы можете определить класс с именем "Студент " и создать три экземпляра класса "Студент": " Петр", " Павел" и " Полина ". Термин "Объект " обычно относится к экземпляру класса. Но он часто используется свободно, которые могут относиться к классу или экземпляру.

Графически можно представить класс в виде UML[1] диаграммы как прямоугольник в виде как трех секций, в котором присутствует секция наименования класса, секция инкапсуляции данных и методов (функций или операций) класса. Пример общего представления диаграммы класса представлен на рисунке 1.1.

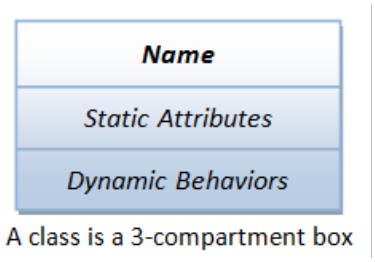

Рисунок 1.1 - Диаграмма класса. Общее представление.

Рассмотрим подробнее диаграмму класса. Имя (или сущность): определяет класс.

Переменные (или атрибуты, состояние, поля данных класса): содержит статические атрибуты класса, или описывают свойства класса (сущности предметной области).

Методы (или поведение, функции, работа с данными): описывают динамическое поведение класса. Другими словами, класс инкапсулирует статические свойства (данные) и динамические модели поведения (операции, которые работают с данными) в одном месте (“контейнере” или “боксе”), представленном на рисунке в виде прямоугольника.

На рисунке 1.2 показано несколько примеров классов. У каждого из них есть имя, переменные класса и методы.

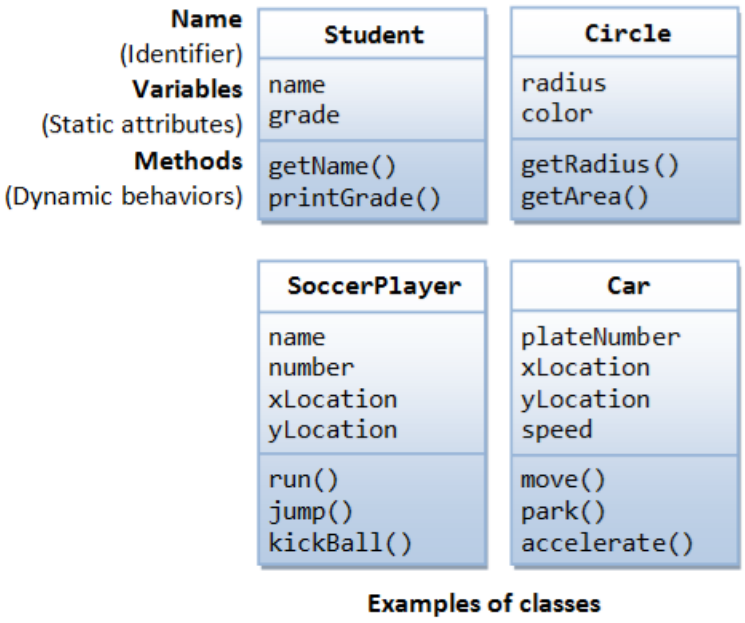

Рисунок 1.2 - Примеры классов

На рисунке 1.3 показаны два экземпляра класса типа Student "paul" и "peter" в виде UML диаграммы экземпляра класса.

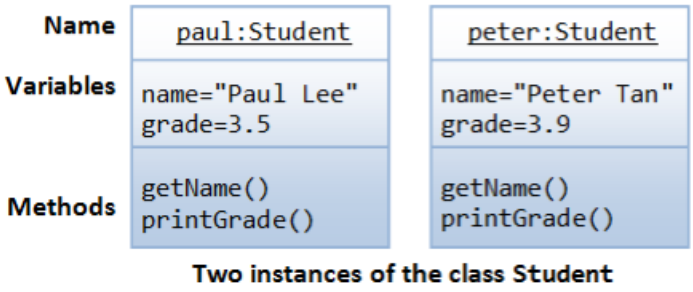

Рисунок 1.3 - Экземпляры класса Student.

Приведенные выше диаграммы классов описаны в соответствии с UML нотацией. Класс представляется в этой нотации как прямоугольник, разделенный на три отсека, один содержит название, два вторых переменные (поля данных класса) и методы, соответственно. Имя класса выделено жирным шрифтом и находится посредине. Экземпляр (объект класса) также представляется в виде прямоугольника, разделенного на три отсека, в первом отсеке, надпись с именем экземпляра, показанной в instanceName: Classname и выделенная подчеркиванием (название_экземпляра : имя_класса).

Кратко подведем итоги по определению класса:

1) Класс, тип данных, определяемый программистом, абстрактный тип данных, повторно-используемый программный объект, который имитирует реальные сущности предметной области. Класс можно представить графически в виде контейнера на UML диаграмме, который состоит из трех условных частей и содержит имя, переменные и методы.

2) Класс инкапсулирует статическое состояние объекта, его атрибуты или свойства данных в виде переменных класса и поведение объекта в виде методов, которые могут реализовывать определенные алгоритмы.

3) Значения переменных или поля данные составляют его состояние. Методы создает свои модели поведения.

Экземпляр класса — это представление (или реализация) конкретного представителя класса.

2. Определение класса.

В Java, мы используем слово class как ключевое или служебное слово, например, чтобы задать определение класса. Для примера:

```
public class Circle {    // class name
     double radius;      // variables
     String color;

     double getRadius() {...} // methods
     double getArea() {...}
}
public class SoccerPlayer {   // class name
     int number;                    // variables
     String name;
     int x, y;

     void run() {...}    // methods
     void kickBall() {...}
}
```

Давайте разъясним, что такое контроль доступа или спецификатор доступа, например, public и private, позже.

Конвенция кода для класса (Class Naming Convention).

Конвенцией кода называют соглашение между программистами о правилах написания кода. Соглашение содержит правила именования переменных и не только. Например, в соответствии с конвенцией кода на Java имя класса должно быть всегда существительным или словосочетанием из нескольких слов. Все слова должны с прописной буквы (так, называемая верблюжья нотация или camel notation). Совет: для имени класса всегда используйте существительное в единственном числе. Выберите значимое и самодостаточное имя для названия класса. Для примера, SoccerPlayer, HttpProxyServer, FileInputStream, PrintStream and SocketFactory будут подходящими именами в определенной предметной области, для которой вы пишете программу.

3. Создание экземпляров класса

Чтобы создать экземпляр класса, вы должны выполнить следующие действия:

• объявить идентификатор экземпляра (имя экземпляра) конкретного класса.

• Сконструировать экземпляр класса (то есть выделить память для

экземпляра и инициализировать его) с помощью оператора "new".

Например, предположим, что у нас есть класс с именем Circle , тогда мы можем создавать экземпляры класса Circle, следующим образом:

```
// Declare 3 instances of the class Circle, c1, c2, and c3
Circle c1, c2, c3;
// Allocate and construct the instances via new operator 4 c1 =
new Circle();
c2 = new Circle(2.0);
c3 = new Circle(3.0, "red");
// You can declare and construct in the same statement
Circle c4 = new Circle();
```

4. Операция получения доступа к компонентам класса.

Доступ к компонентам класса осуществляется с помощью операции получения доступа, а именно операции точка “.”

Переменные и методы, входящие в состав класса, формально называется переменные-поля данных класса и методы класса и являются компонентами класса. Для ссылки на переменную-поле данных класса или метод, вы должны:

- сначала создать экземпляр класса, который вам нужен;
- затем, использовать оператор точка “.” чтобы сослаться на элемент класса (переменную-поле данных или метод класса).

Предположим, что у нас есть класс с именем Circle, с двумя переменными (радиус и цвет) и двумя методами (getRadius () и GetArea ()). Мы создали три экземпляра класса Circle, а именно, C1, C2 и C3 . Чтобы вызвать метод GetArea (), вы должны сначала определить к какой именно сущности вы обращаетесь, об этом собственно говорит говорит c2, а затем использовать оператор точка, в виде c2.getArea (), для вызова метода GetArea () экземпляра c2. Например,

```
// Declare and construct instances c1 and c2 of the class Circle
Circle c1 = new Circle ();
Circle c2 = new Circle ();
// Invoke member methods for the instance c1 via dot operator
System.out.println(c1.getArea());
System.out.println(c1.getRadius());
// Reference member variables for instance c2 via dot operator
c2.radius = 5.0;
c2.color = "blue"
```

Вызов метода getArea () без указания экземпляра не имеет смысла , так как радиус неизвестно какого объекта (может быть много окружностей, у каждой из которых свой собственный радиус) .

В общем, полагают, есть класс, называемый AClass с переменной-полем данных под названием aVariable и способом доступа к полю методом aMethod(). Экземпляр называется anInstance и строится с использованием AClass.

Вы можете использовать для доступа к открытым полям и методам операцию точка ".", например - anInstance.aVariable и anInstance.aMethod().

5. Переменные - поля данных класса

Переменная-поле данных имеет имя (идентификатор) и тип, а также может иметь значение определенного типа, например базового или типа определенного программистом ранее. Переменная-поле данных может также быть экземпляром определенного класса (которые будут обсуждаться позже).

Конвенция об именах переменных гласит: имя переменной должно быть существительным или словосочетанием из нескольких слов. Первое слово в нижнем регистре, а остальные слова пишутся с прописной буквы (двугорбая нотация или camel notation), например, roomNumber, Xmax , Ymin и xTopLeft.

Обратите внимание, что имя переменной начинается с буквы в нижнем регистре, в то время как имя класса всегда начинается с заглавной буквы.

Формальный синтаксис для определения переменной в Java:

```
[модификатор_доступа] тип имя_перем [= иниц_знач];
[модификатор_доступа] тип имя_пер-1 [=иниц_знач-1], тип имя_пер-
2 [=иниц_знач-2] ...;
```

Например:

```
private double radius;
public int length = 1, width = 1;
```

6. Методы класса

Метод (способ как описано в предыдущем разделе):

- принимает параметры из вызова (как в функции);
- выполняет операции, описанные в теле метода, и;
- возвращает часть результата (или void) в точку вызова.

Формальный синтаксис объявления метода в Java:

```
1 [AccessControlModifier] returnType methodName ([argumentList])
{ 2 // method body or implementation
3 ......
4}
```

Например:

```
public double getArea() {
     return radius*radius*Math.PI; 3
}
```

Конвенция кода о правилах записи имен методов гласит следующее: имя метода должно быть глаголом или начинаться глаголом в виде фразы из нескольких слов, первое слово записывается в нижнем регистре, а остальные слова начинаются с прописной буквы (двугорбая запись). Например, getRadius (), getParameterValues ().

Обратите внимание, что имя переменной существительное (обозначающий статический атрибут), в то время как имя метода - глагол (обозначает действие). Они имеют те же наименования. Тем не менее, вы можете легко отличить их от контекста. Методы могут принимать аргументы в скобках (возможно, нулевой аргумент, в пустых скобках), ноне поля данных.

При записи, методы обозначаются парой круглых скобок, например, println(), getArea ().

7. Теперь соберем все вместе: Пример ООП

На рисунке ниже представлена диаграмма класса и его трех экземпляров.

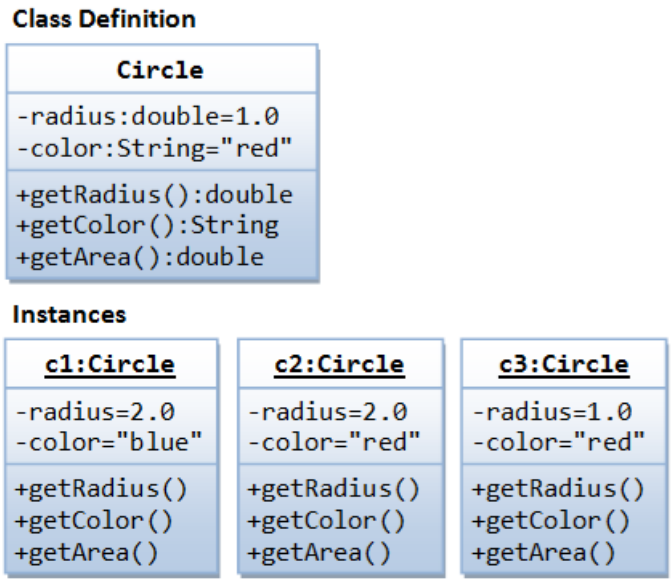

Рисунок 1.4 – Диаграмма класса и его экземпляров.

Класс называется Circle и должен быть определен, как показано на диаграмме классов выше.

Он содержит две переменные: radius (типа double) и color (типа String); и три метода: getRadius() , getColor(), и getArea () .

Три экземпляра Circle называются C1, C2, C3, и должны быть построены с учетом их соответствующих элементов данных и методов, как показано на схемах UML для экземпляров класса.

Исходные коды для изображенного на UML диаграмме класса Circle.java:

Circle.java

```
// Define the Circle class
public class Circle { // Save as "Circle.java"

     // Private variables
     private double radius;
     private String color;

     // Constructors (overloaded)
     public Circle() { // 1st Constructor
     radius = 1.0;
     color = "red";
     }
     public Circle(double r) { // 2nd Constructor
          radius = r;
          color = "red";
     }

     public Circle(double r, String c) { // 3rd Constructor
          radius = r;
          color = c;
     }
     // Public methods
     public double getRadius() { return radius;
     }
     public String getColor() {
     return color;
     }
     public double getArea() {
     return radius*radius*Math.PI;
     }
```

Компиляция "Circle.java" в "Circle.class". Обратите внимание, что в классе Circle нет метода main(). Следовательно, это не будет программой на Java, и вы не можете запустить класс Circle сам по себе. Класс Circle нужен, чтобы быть отдельным строительным блоком и использоваться в других программах.

Дополним нашу программу еще одним классом, который будет демонстрировать работу с нашим классом. Мы напишем TestCircle, в котором будем использовать Circle класс. Класс TestCircle содержит метод main(), теперь мы можем откомпилировать и запустить программу.

TestCircle.java

```
public class TestCircle { // Save as "TestCircle.java"
     public static void main(String[] args) { // Execution entry
point
          // Construct an instance of the Circle class called c1
          Circle c1 = new Circle(2.0, "blue"); // Use 3rd
constructor
          System.out.println("Radius is " + c1.getRadius() +
"Color is " + c1.getColor() + "Area is " + c1.getArea()); // use
dot operator to invoke member methods
     }
     // Construct another instance of the Circle class called c2
     Circle c2 = new Circle(2.0); // Use 2nd constructor
     System.out.println("Radius is " + c2.getRadius() + "Color is
" + c2.getColor() + "Area is " + c2.getArea());
     // Construct yet another instance of the Circle class called
c3
     Circle c3 = new Circle(); // Use 3rd constructor
     System.out.println("Radius is " + c3.getRadius() + "Color is
" + c3.getColor() + "Area is" + c3.getArea());
}
```

Запустим TestCircle и увидим результат:

```
Radius is 2.0 Color is blue Area is 12.566370614359172
Radius is 2.0 Color is red Area is 12.566370614359172
Radius is 1.0 Color is red Area is 3.141592653589793
```

8. Конструкторы

Конструктор – это специальный метод класса, который имеет то же имя, что используется в качестве имени класса. В приведенном выше класса Circle, мы определим три перегруженных версии конструктора Circle(...). Конструктор используется для создания и инициализации всех переменных-полей данных класса. Чтобы создать новый экземпляр класса, вы должны использовать специальный оператор "new" с последующим вызовом одного из конструкторов.

Например,

```
Circle c1 = new Circle();
Circle c2 = new Circle(2.0);
Circle c3 = new Circle(3.0, "red");
```

Конструктор отличается от обычного метода следующим:

- название метода-конструктора совпадает с именем класса, а имя класса по конвенции, начинается с заглавной буквы;
- конструктор не имеет возвращаемого значения типа (или неявно не возвращает), таким образом, нет объявления типа возвращаемого значения при объявлении;
- конструктор может быть вызван только через оператор «new», он может быть использован только один раз, чтобы инициализировать построенный экземпляр.
- вы не можете впоследствии вызвать конструктор в теле программы подобно обычным методам (функциям);
- конструкторы не наследуется (будет объяснено позже).

Конструктор без параметров называется конструктором по умолчанию, который инициализирует переменные-поля данных через их значения по умолчанию. Например, конструктор Circle() в рассмотренном выше примере.

9. Перегрузка методов

Перегрузка методов означает, что несколько методов могут иметь то же самое имя метод, но сами методы могут иметь различные реализации (версии). Тем не менее, различные реализации должны быть различимы по списку их аргументов (либо количество аргументов, или типа аргументов, или их порядок).

Пример: метод average() имеет три версии с различными списками аргументов. При вызове может использоваться соответствующий выбору вариант, в соответствии с аргументами.

```
public class TestMethodOverloading {
   public static int average(int n1, int n2) { // A
   return (n1+n2)/2;
   }

   public static double average(double n1, double n2) { // B
        return (n1+n2)/2;
```

```
    }

    public static int average(int n1, int n2, int n3) { // C
        return (n1+n2+n3)/3;
    }

    public static void main(String[] args) {
        System.out.println(average(1, 2)); // Use A
        System.out.println(average(1.0, 2.0)); // Use B
        System.out.println(average(1, 2, 3)); // Use C
        System.out.println(average(1.0,2)); //Use B - int 2
implicitly casted to double 2.0
        // average(1, 2, 3, 4); // Compilation Error - No
matching method
    }
  }
```

Рассмотрим перегрузку конструктора класса Circle.

Приведенный выше класс Circle имеет три версии конструктора, которые отличаются списком их параметров, следовательно:

```
Circle()
Circle(double r)
Circle(double r, String c)
```

В зависимости от фактического списка аргументов, используемых при вызове метода, будет вызван соответствующий конструктор. Если ваш список аргументов не соответствует ни одному из определенных методов, вы получите ошибку компиляции.

10. Модификаторы контроля доступа- public или private.

Контроль за доступом осуществляется с помощью модификатора, он может быть использован для управления видимостью класса или переменных – полей или методов внутри класса. Мы начнем со следующих двух модификаторов управления доступом:

public: класс / переменная / метод доступным и для всех других объектов в системе.

private: класс / переменная / метод доступным и в пределах только этого класса.

Например, в приведенном выше определении Circle, radius переменная-поле данных класса объявлена private. В результате, radius доступен внутри класса Circle , но не внутри класса TestCircle . Другими словами, вы не можете использовать "c1.radius" по отношению к радиусу C1 в классе TestCircle . Попробуйте вставить текст "System.out.println (c1.radius) ;" в TestCircle и понаблюдать за сообщением об ошибке.

Попробуйте изменить radius на public, и повторно запустить пример.

С другой стороны, метод getRadius() определяется как public в классе Circle . Таким образом, он может быть вызван в классе TestCircle.

В нотации UML на диаграмме классов обозначается: общедоступные (public) элементы обозначены со знаком "+", в то время как закрытые (private) элементы со знаком " - ".

11. Информация по сокрытию реализации и инкапсуляции.

Класс инкапсулирует имя, статические атрибуты и динамическое поведение в "виде трех частей целого". Можно себе представить класс в виде некоторой коробки, на которой написано имя, внутри нее есть некоторое содержимое. После того, как класс определен, то есть вы туда сто-то положили, написали название на коробке, то можно закрыть "крышку" у этой коробки и поставить ее на полку. В дальнейшем как вы сами можете использовать эту коробку, так и можете предоставить возможность использовать ее всем желающим. Другими словами, любой желающий может открыть "крышку" этой коробки и использовать ее содержимое, то есть использовать ваш класс в своем приложении. Этого нельзя сделать, программируя в традиционном процедурном-ориентированном стиле, например на языке Си, так как статические атрибуты (или переменные)

разбросаны по всей программе и в файлах, заголовки которых вы включаете в код. Вы не сможете "вырезать" куски из части программы на Си, включить их в другую программу и ожидать что такая программа запуститься без внесения в код серьезных изменений.

Переменные-поля данных класса, как правило, скрыты от внешнего слоя (то есть, недоступны другим классам), для этого осуществляется разграничение доступа или контроль доступа с использованием модификатора доступа - private. Доступ к закрытым переменным - полям класса предоставляются через методы, объявленные с модификатором public, например, getRadius () и getColor().

Использование такого подход соответствует принципу сокрытия информации – принципу инкапсуляции. То есть, объекты могут общаться друг с другом, только через хорошо определенные интерфейсы (публичные методы). Объектам не позволено знать детали реализации других объектов. Детали реализации всегда скрыты извне или инкапсулированы внутри класса. Сокрытие информации облегчает повторное использование класса.

Основное правило при создании определения классов: не объявляйте переменные общедоступными – с модификатором доступа public, если у вас на то нет веских оснований, не нарушайте принцип инкапсуляции.

ЗАДАНИЕ.

Необходимо реализовать простейший класс на языке программирования Java. Не забудьте добавить метод toString() к вашему классу. Так-же в программе необходимо предусмотреть класс-тестер для тестирования класса и вывода информации об объекте.

Упражнение 1.

Реализуйте простейший класс «Собака».

Упражнение 2.

Реализуйте простейший класс «Мяч».

Упражнение 3.

Реализуйте простейший класс «Книга».

ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ 1.

Задание: Реализуйте класс «Собака, который содержит данные экземпляра, которые представляют имя собаки и ее возраст. Определить конструктор собаки, чтобы принять и инициализировать данные экземпляр. Включите методы получения и установки для имени и возраста. Включите метод вычисления и возвращает возраст собаки в "человеческих" лет (возраст семь раз собаки). Включите метод toString(), который возвращает описание на одну строку собаки. Создание класса драйвера под названием питомника, основной метод конкретизирует и обновляет несколько объектов собаки.

Код программы:

Dog.java

```
import java.lang.*;
public class Dog {
     private String name;
     private int age;
     public Dog(String n, int a){
          name = n;
          age = a;
     }
     public Dog(String n){
          name = n;
          age = 0;
     }
     public Dog(){
          name = "Pup";
          age = 0;
     }
     public void setAge(int age) {
          this.age = age;
```

```
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getName(String name){
        return name;
    }
    public int getAge() {
        return age;
    }
    public String toString(){
        return this.name+", age "+this.age;
    }
    public void intoHumanAge(){
        System.out.println(name+"'s age in human years is
"+age*7+" years");
    }
}
```

TestDog.java

```
import java.lang.*;
public class TestDog {
    public static void main(String[] args) {
        Dog d1 = new Dog("Mike", 2);
        Dog d2 = new Dog("Helen", 7);
        Dog d3 = new Dog("Bob"); d3.setAge(1);
        System.out.println(d1);
        d1.intoHumanAge();
        d2.intoHumanAge();
        d3.intoHumanAge();
    }
}
```

ПРАКТИЧЕСКАЯ РАБОТА №2. ИСПОЛЬЗОВАНИЕ UML ДИАГРАММ В ОБЪЕКТНО- ОРИЕНТИРОВАННОМ ПРОГРАММИРОВАНИИ

Цель работы: работа с UML-диаграммами классов.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Язык моделирования Unified Modeling Language (UML) является стандартом де-факто с 1998 года для проектирования и документирования объектно-ориентированных программ. Средствами UML в виде диаграмм можно графически изобразить класс и экземпляр класса.

Графически представляем класс в виде прямоугольника, разделенного на три области – область именования класса, область инкапсуляции данных и область операций (методы).

Имя (или сущность) : определяет класс.

Переменные (или атрибуты, состояние, поля данных класса): содержит статические атрибуты класса, или описывают свойства класса (сущности предметной области). На рисунке 2.1 приведен общий вид UML диаграммы класса.

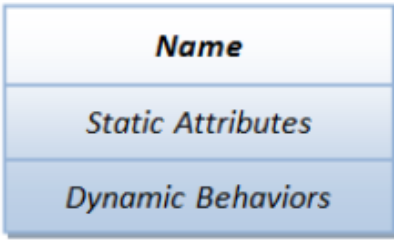

Рисунок 2.1 - Представление класса.

Методы (или поведение, функции, работа с данными): описывают динамическое поведение класса. Другими словами, класс инкапсулирует

статические свойства (данные) и динамические модели поведения (операции, которые работают с данными) в одном месте (“коробке” или прямоугольнике).

На рисунке 2.2 показано несколько примеров классов:

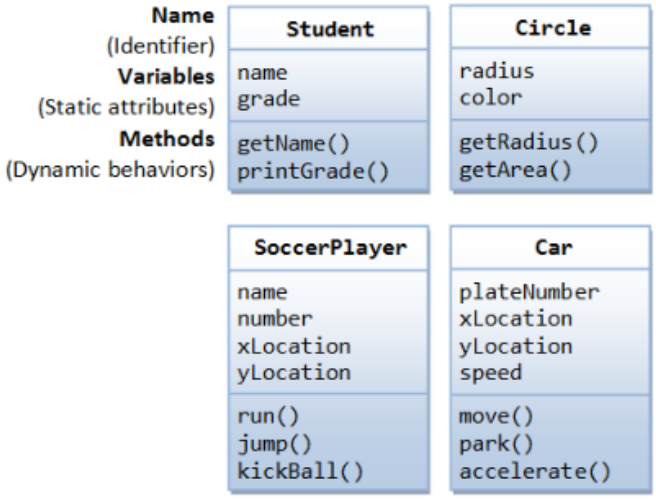

Рисунок 2.2 - Примеры экземпляров классов.

На рисунке 2.3 показаны два экземпляра класса типа Student "paul" и "peter".

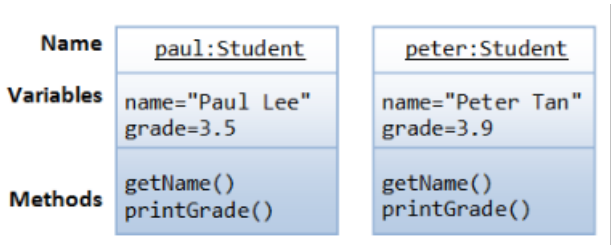

Рисунок 2.3 - Экземпляры класса Student.

Приведенные выше диаграммы классов описаны в соответствии с UML нотацией. Класс представляется в этой нотации как прямоугольник, разделенный на три области, одна содержит название, две вторых содержат переменные (поля данных класса) и методы класса, соответственно. Имя класса выделено жирным шрифтом и находится посредине. Экземпляр (объект

класса) также представляется в виде прямоугольника, разделенного на три части, в первой части помещается надпись с именем экземпляра, например в instanceName:Classname и выделенная подчеркиванием (название_экземпляра : имя_класса).

ЗАДАНИЯ

Упражнение 1.

По диаграмме класса UML описывающей сущность Автор. Необходимо написать программу, которая состоит из двух классов Author и TestAuthor. Класс Author должен содержать реализацию методов, представленных на диаграмме класса на рисунке 2.4.

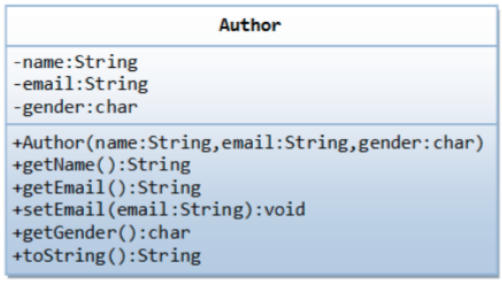

Рисунок 2.4 - Диаграмма класса Author.

Упражнение 2.

По UML диаграмме класса, представленной на рисунке 2.5. написать программу, которая состоит из двух классов. Один из них Ball должен реализовывать сущность мяч, а другой с названием TestBall тестировать работу созданного класса. Класс Ball должен содержать реализацию методов, представленных на UML. Диаграмма на рисунке описывает сущность Мяч написать программу.

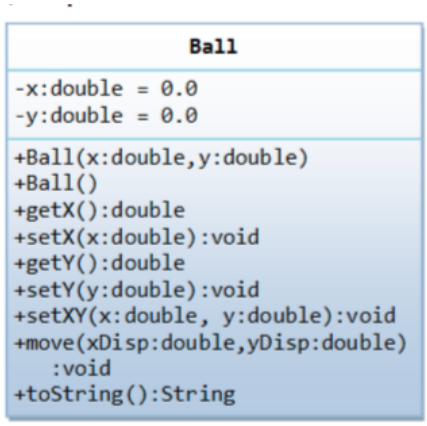

Рисунок 2.5 - Диаграмма класса Ball.

Класс Ball моделирует движущийся мяч. В состав класса входят:

- Две переменные с модификатором private (поля данных класса): x, y, которые описывают положение мяча на поле.
- Конструкторы, public методы получения и записи значений для private переменных.
- Метод setXY (), который задает положение мяча и метод setXYSpeed(), чтобы задать скорость мяча
- Метод move() , позволяет переместить мяч, так что что увеличивает x и y на данном участке на xDisp и yDisp, соответственно.
- Метод toString(), который возвращает "Ball @ (x , y) " .

ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ.

Класс, называемый Author (с англ. Автор), как показано на диаграмме классов, моделирует сущность предметной области – автор книги.

Он содержит:

- три private переменных-поля данных класса: name (типа String) , email (типа String) , и gender (типа char, которая может принимать три

значения либо 'M', если автор книги мужчина, 'F' – если автор книги женщина, или 'U' если пол автора неизвестен, - вы можете также использовать для реализации логическую переменную под названием male для обозначения пола автора, которая будет принимать значение истина или ложь);

- один конструктор для инициализации переменных name, email и gender с заданными значениями. (здесь не будет использоваться конструктор по умолчанию, так как нет значений по умолчанию: ни для имени, ни для электронной почты или пола).
- стандартные методы класса, геттеры/сеттеры: getName(), должны быть объявлены с модификатором public;
- методы getEmail(), setEmail(), and getGender(), нужно упомянуть, что класс не содержит методов сеттеров для полей данных - имени и пола, так как эти атрибуты не могут изменяться;
- метод toString(), который должен возвращать следующий текст "автор - имя (пол) на адрес электронной почты, например, "Ivan Popov(m) at ivPopov@somewhere.com", или "Anna Ivanova(ms) at anIvanova@somewhere.com ", то есть в строке должно быть записано имя[пробел](пол)[пробел]at[пробел]емайл

Ball.java

```
package balls;
public class Ball {
     private double x = 0.0;
     private double y = 0.0;

     public Ball(){}
     public Ball(double x, double y){
          this.x = x;
          this.y = y;
     }
     public double getX() {
          return x;
     }
     public double getY() {
          return y;
     }
     public void setX(double x) {
          this.x = x;
     }
     public void setY(double y) {
          this.y = y;
     }

     public void setXY(double x, double y){
          this.x = x;
          this.y = y;
     }
     public void move( double xDisp, double yDisp){
          x+=xDisp;
          y+=yDisp;
     }
     @Override
     public String toString() {
          return "Ball @ ("+this.x+", "+this.y+").";
     }
}
```

TestBall.java

```
package balls;
public class TestBall {
     public static void main(String[] args) {
          Ball b1 = new Ball(100, 100);
          System.out.println(b1);
          b1.move(30, 15);
          System.out.println(b1);
     }
}
```

ПРАКТИЧЕСКАЯ РАБОТА №3. НАСЛЕДОВАНИЕ. АБСТРАКТНЫЕ СУПЕРКЛАССЫ И ИХ ПОДКЛАССЫ В JAVA.

Цель работы: освоить на практике работу с абстрактными классами и наследованием на Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Класс, содержащий абстрактные методы, называется абстрактным классом. Такие классы при определении помечаются ключевым словом abstract.

Абстрактный метод внутри абстрактного класса не имеет тела, только прототип. Он состоит только из объявления и не имеет тела:

abstract void yourMethod();

По сути, мы создаём шаблон метода. Например, можно создать абстрактный метод для вычисления площади фигуры в абстрактном классе Фигура. А все другие производные классы от главного класса могут уже реализовать свой код для готового метода. Ведь площадь у прямоугольника и треугольника вычисляется по разным алгоритмам и универсального метода не существует.

Если вы объявляете класс, производный от абстрактного класса, но хотите иметь возможность создания объектов нового типа, вам придётся предоставить определения для всех абстрактных методов базового класса. Если этого не сделать, производный класс тоже останется абстрактным, и компилятор заставит пометить новый класс ключевым словом **abstract**.

Абстрактный класс не может содержать какие-либо объекты, а также абстрактные конструкторы и абстрактные статические методы. Любой

подкласс абстрактного класса должен либо реализовать все абстрактные методы суперкласса, либо сам быть объявлен абстрактным.

```
public abstract class Swim {
     // абстрактный метод плавать()
     abstract void neigh();

     // абстрактный класс может содержать и обычный метод
     void run() {
          System.out.println("Куда идешь?");
     }
}
class Swimmer extends Swim { ....
}
```

ЗАДАНИЯ

Упражнение 1.

Создайте абстрактный родительский суперкласс Shape и его дочерние классы (подклассы).

Упражнение 2.

Перепишите суперкласс Shape и его подклассы, так как это представлено на диаграмме Circle, Rectangle and Square рисунка 3.1.

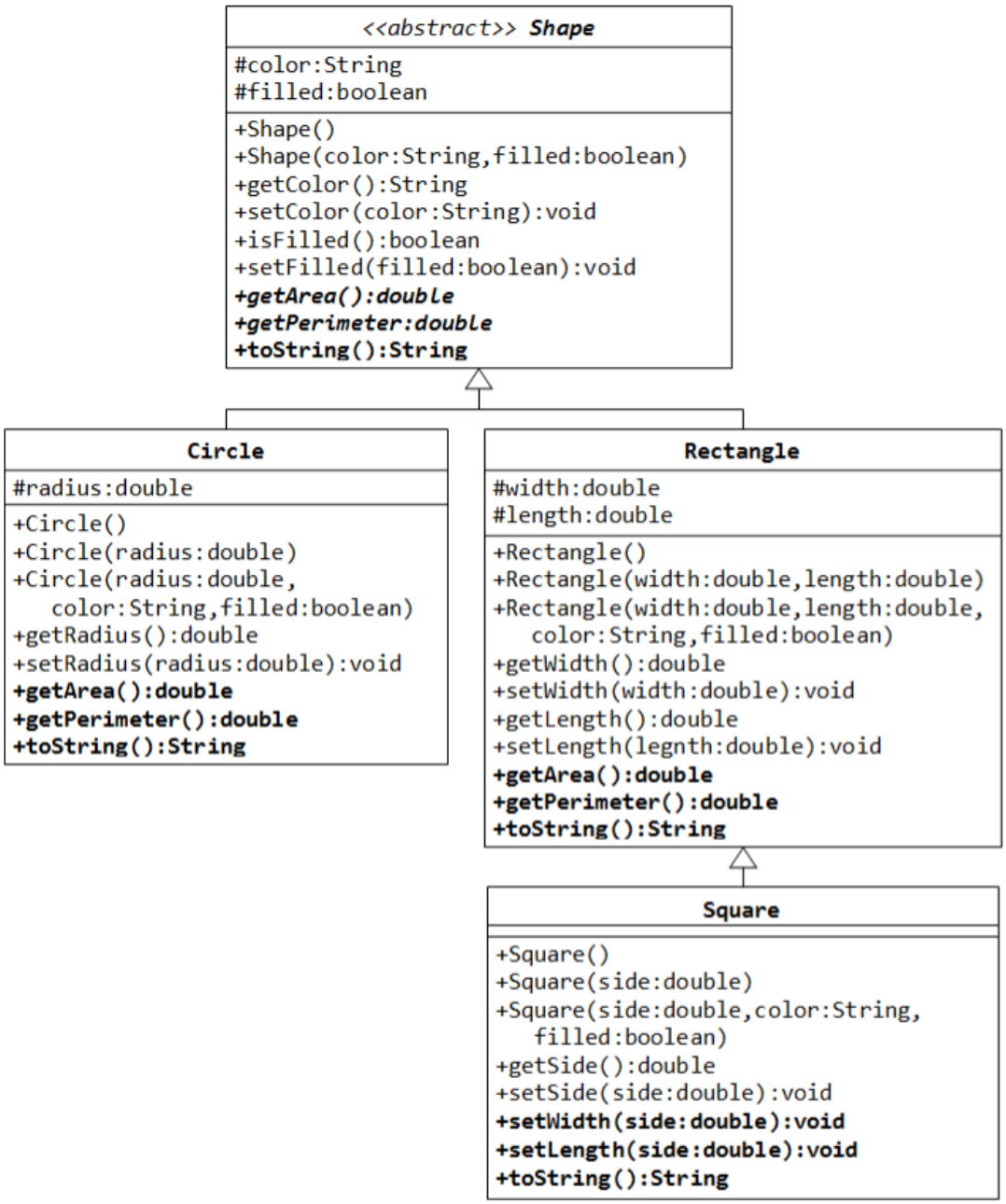

Рисунок 3.1 – Диаграмма суперкласса Shape.

В этом задании, класс Shape определяется как абстрактный класс, который содержит:

• Два поля или переменные класса, объявлены с модификатором ***protected*** color (тип String) и filled (тип boolean). Такие защищенные переменные могут быть доступны в подклассах и классах в одном пакете. Они обозначаются со знаком “#” на диаграмме классов в нотации языка UML.

- Методы геттеры и сеттеры для всех переменных экземпляра класса, и метод toString().
- Два абстрактных метода getArea() и getPerimeter() выделены курсивом в диаграмме класса).

В подклассах Circle (круг) и Rectangle (прямоугольник) должны переопределяться абстрактные методы getArea() и getPerimeter(), чтобы обеспечить их надлежащее выполнение для конкретных экземпляров типа подкласс. Также необходимо для каждого подкласса переопределить toString().

Упражнение 3.

Вам нужно написать тестовый класс, чтобы самостоятельно это проверить, необходимо объяснить полученные результаты и связать их с понятием ООП - полиморфизм. Некоторые объявления могут вызвать ошибки компиляции. Объясните полученные ошибки, если таковые имеются.

```
Shape s1 = new Circle(5.5, "RED", false);   // Upcast Circle to
Shape
System.out.println(s1);                     // which version?
System.out.println(s1.getArea());           // which version?
System.out.println(s1.getPerimeter());      // which version?
System.out.println(s1.getColor());
System.out.println(s1.isFilled());
System.out.println(s1.getRadius());

Circle c1 = (Circle)s1; // downcast back to Circle
System.out.println(c1);
System.out.println(c1.getArea());
System.out.println(c1.getPerimeter());
System.out.println(c1.getColor());
System.out.println(c1.isFilled());
System.out.println(c1.getRadius());

Shape s2 = new Shape();

Shape s3 = new Rectangle(1.0, 2.0, "RED", false); // upcast
System.out.println(s3);
System.out.println(s3.getArea());
System.out.println(s3.getPerimeter());
System.out.println(s3.getColor());
```

```
System.out.println(s3.getLength());

Rectangle r1 = (Rectangle)s3; // downcast
System.out.println(r1);
System.out.println(r1.getArea());
System.out.println(r1.getColor());
System.out.println(r1.getLength());

Shape s4 = new Square(6.6); // Upcast
System.out.println(s4);
System.out.println(s4.getArea());
System.out.println(s4.getColor());
System.out.println(s4.getSide());

// Take note that we downcast Shape s4 to Rectangle,
// which is a superclass of Square, instead of Square
Rectangle r2 = (Rectangle)s4;
System.out.println(r2);
System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());

// Downcast Rectangle r2 to Square
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());
System.out.println(sq1.getLength());
```

Упражнение 4.

Напишите два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable.

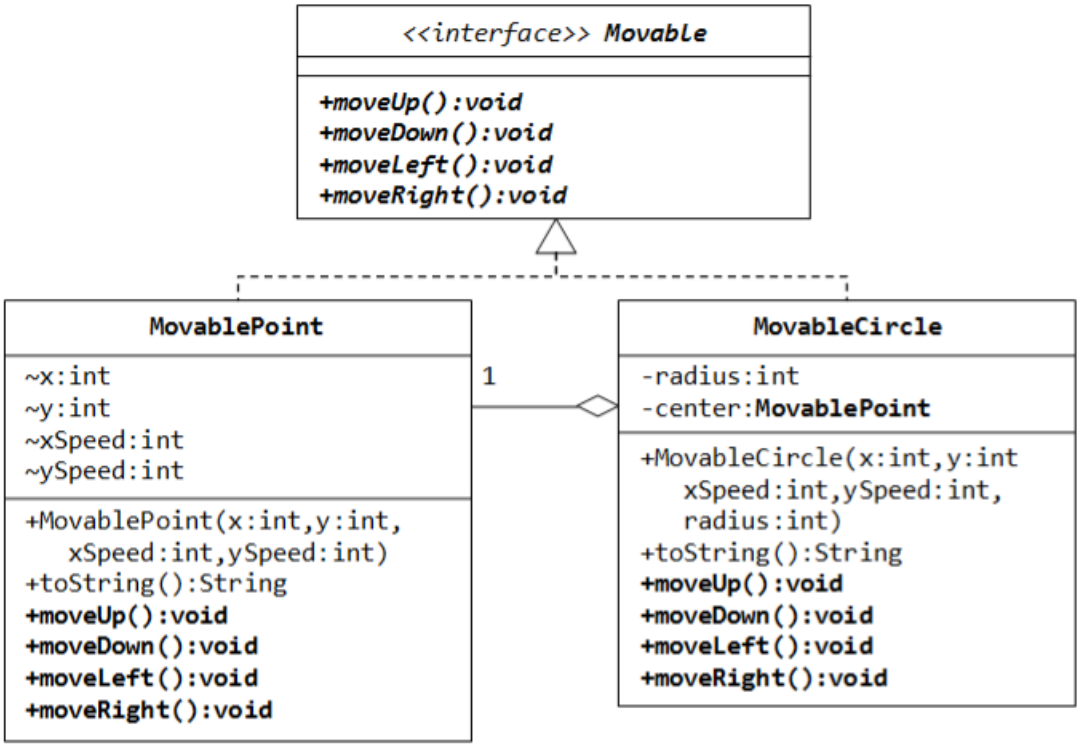

Рисунок 3.2. – Диаграмма реализации итерфейса Movable.

```
public interface Movable {
// saved as "Movable.java"
public void moveUp();
......
}
```

Упражнение 5.

Напишите новый класс MovableRectangle (движущийся прямоугольник). Его можно представить как две движущиеся точки MovablePoints (представляющих верхняя левая и нижняя правая точки) и реализующие интерфейс Movable. Убедитесь, что две точки имеет одну и ту же скорость (нужен метод это проверяющий).

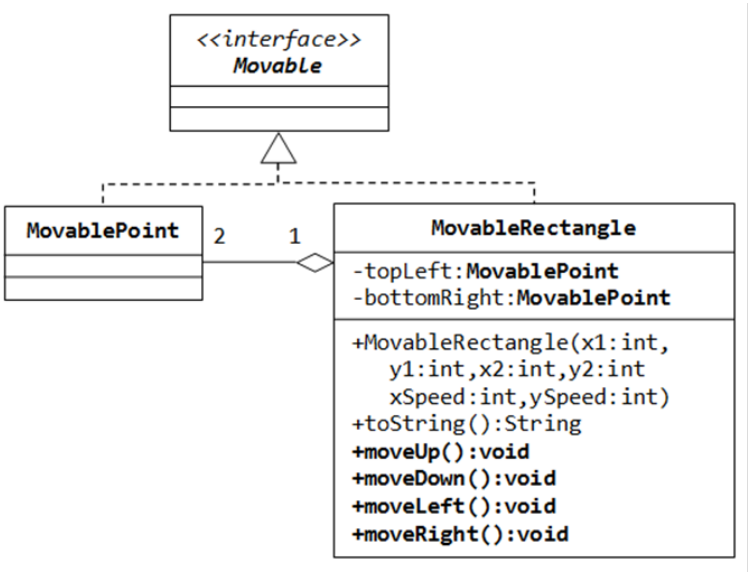

Рисунок 3.3 – Диаграмма класса MovableRectangle.

ПРИМЕР РЕШЕНИЯ ЗАДАНИЯ 1.

Реализации класса Circle:

Circle.java

```
package shape;
import java.math.*;
public class Circle extends Shape{
     protected double radius;
     public Circle(){
          this.filled = false;
          this.color = "blue";
          radius = 1;
     }
     public Circle(double radius){
          this.filled = false;
          this.color = "blue";
          this.radius = radius;
     }
     public Circle(double radius, String color, boolean filled){
          this.radius = radius;
          this.color = color;
          this.filled = filled;
     }
     public double getRadius() {
          return radius; }
     public void setRadius(double radius) {
          this.radius = radius; }
     @Override
     public double getArea() {
          return Math.PI*radius*radius; }
     @Override
```

```
     public double getPerimeter() {
          return 2*Math.PI*radius; }
     @Override
     public String toString() {
          return "Shape: circle, radius: "+this.radius+", color:
"+this.color;
           }
     }
```

ПРАКТИЧЕСКАЯ РАБОТА №4.СОЗДАНИЕ GUI. СОБЫТИЙНОЕ ПРОГРАММИРОВАНИЕ В JAVA.

Цель работы: введение в событийное программирование на языке Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Данная практическая работа посвящена закреплению практических навыков по созданию приложений на Java с использованием следующих элементов GUI:

- Текстовые поля и области ввода текста;
- Менеджеры компоновки компонентов;
- Слушатель мыши;
- Создание меню.

Text Fields - текстовое поле или поля для ввода текста (можно ввести только одну строку). Примерами текстовых полей являются поля для ввода логина и пароля, например, используемые, при входе в электронную почту.

Пример создания объекта класса JTextField:

```
JTextField jta = new JTextField (10);
```

В параметрах конструктора задано число 10, это количество символов, которые могут быть видны в текстовом поле. Текст веденный в поле JText может быть возвращен с помощью метода getText(). Также в поле можно записать новое значение с помощью метода setText(String s).

Как и у других компонентов, мы можем изменять цвет и шрифт текста в текстовом поле.

Пример 1.

```
class LabExample extends JFrame{
    JTextField jta = new JTextField(10);
    Font fnt = new Font("Times new roman",Font.BOLD,20);
    LabExample(){
        super("Example");
        setLayout(new FlowLayout());
        setSize(250,100);
        add(jta);
        jta.setForeground(Color.PINK);
        jta.setFont(fnt);
        setVisible(true);
    }
    public static void main(String[]args) {
        new LabExample();
      }
}
```

Рисунок 4.1

Важное замечание

Ответственность за выполнение проверки на наличие ошибок в коде лежит полностью на программисте, например, чтобы проверить, произойдет ли ошибка, когда в качестве входных данных в JTextField ожидается ввод числа. Компилятор не будет ловить такого рода ошибку, поэтому ее необходимо обрабатывать пользовательским кодом.

Выполните следующий пример и наблюдайте за результатом, когда число вводится в неправильном формате:

Пример 2.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class LabExample extends JFrame {
    JTextField jta1 = new JTextField(10);
    JTextField jta2 = new JTextField(10);
    JButton button = new JButton(" Add them up");
    Font fnt = new Font("Times new roman",Font.BOLD,20);
    LabExample()
    {
        super("Example");
        setLayout(new FlowLayout());
        setSize(250,150);
        add(new JLabel("1st Number"));
        add(jta1);
        add(new JLabel("2nd Number"));
        add(jta2);
        add(button);
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                try {
                    double x1 =
Double.parseDouble(jta1.getText().trim());
                    double x2 =
Double.parseDouble(jta2.getText().trim());
                    JOptionPane.showMessageDialog(null, "Result
= "+(x1+x2),"Alert",JOptionPane.INFORMATION_MESSAGE);
                }
                catch(Exception e)
                {
                    JOptionPane.showMessageDialog(null, "Error
in Numbers !","alert" , JOptionPane.ERROR_MESSAGE);
                }
            }
});
        setVisible(true); }

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

JTextArea

Компонент TextArea похож на TextField, но в него можно вводить более одной строки. В качестве примера TextArea можно рассмотреть текст, который мы набираем в теле сообщения электронной почты.

Пример 3.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class TextAreaExample extends JFrame {
    JTextArea jta1 = new JTextArea(10,25);
    JButton button = new JButton("Add some Text");
    public TextAreaExample()
    {
        super("Example");
        setSize(300,300);
        setLayout(new FlowLayout());
        add(jta1);
        add(button);
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                String txt =
JOptionPane.showInputDialog(null,"Insert some text");
                jta1.append(txt);

            }
        });
    }
    public static void main(String[]args){
        new TextAreaExample().setVisible(true);
    }
}
```

Замечание

Мы можем легко добавить возможность прокрутки к текстовому полю, добавив его в контейнер с именем `JScrollPane` следующим образом:

```
JTextArea txtArea = new JTextArea(20,20)
JScrollPane jScroll = new JScrollPane(txtArea);
// ...
add(Scroll); // we add the scrollPane and not the text area.
```

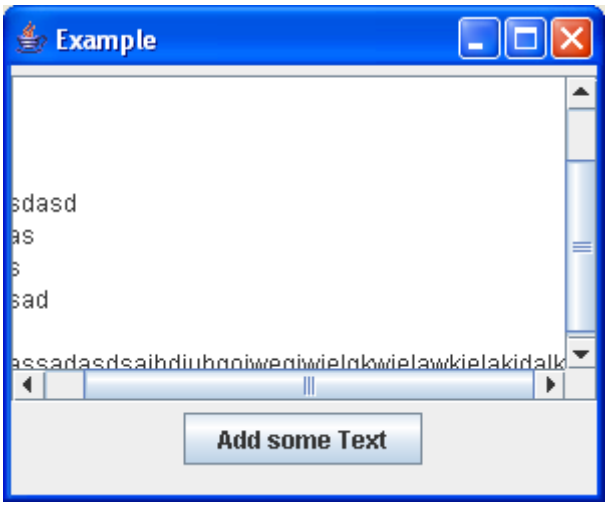

Рисунок 4.2

Менеджеры компоновки компонентов или Layout Менеджеры.

Менеджер BorderLayout:

Разделяет компонент на пять областей (WEST, EAST, NOTH, SOUTH and Center). Другие компоненты могут быть добавлены в любой из этих компонентов пятерками.

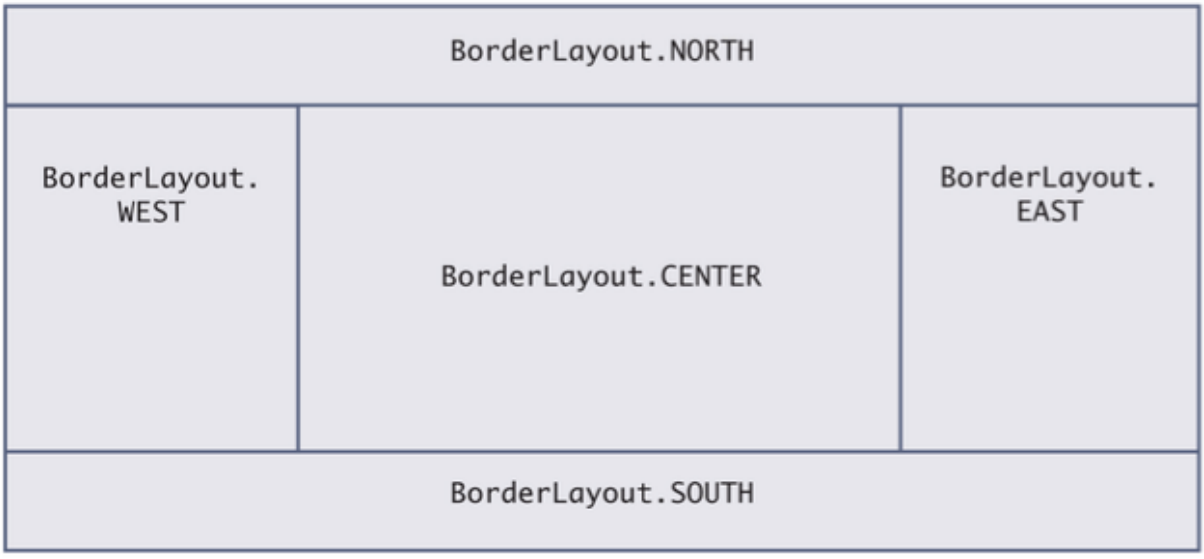

Рисунок 4.3

Метод для добавления в контейнер, который есть у менеджера BorderLayout отличается и выглядит следующим образом:

```
add(comp, BorderLayout.EAST);
```

Обратите внимание, что мы можем, например, добавить панели JPanel в эти области и затем добавлять компоненты этих панелей. Мы можем установить расположение этих JPanel используя другие менеджеры.

Менеджер GridLayout.

С помощью менеджера GridLayout компонент может принимать форму таблицы, где можно задать число строк и столбцов.

Таблица 1.

1	2	3	4
5	6	7	8
9	10	11	12

Если компоненту GridLayout задать 3 строки и 4 столбца, то компоненты будут принимать форму таблицы, показанной выше, и будут всегда будут добавляться в порядке их появления.

Следующий пример иллюстрирует смесь компоновки различных компонентов.

Пример 4.

```
Import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class BorderExample extends Jframe {
    Jpanel[] pnl = new Jpanel[12];
    public BorderExample() {
        setLayout(new GridLayout(3,4));
        for(int I = 0 ; I < pnl.length ; i++) {
            int r = (int) (Math.random() * 255);
            int b = (int) (Math.random() * 255);
            int g = (int) (Math.random() * 255);
            pnl[i] = new Jpanel();
            pnl[i].setBackground(new Color(r,g,b));
            add(pnl[i]);
        }
        pnl[4].setLayout(new BorderLayout());
```

```
        pnl[4].add(new Jbutton("one"),BorderLayout.WEST);
        pnl[4].add(new Jbutton(«two»),BorderLayout.EAST);
        pnl[4].add(new Jbutton(«three»),BorderLayout.SOUTH);
        pnl[4].add(new Jbutton(«four»),BorderLayout.NORTH);
        pnl[4].add(new Jbutton(«five»),BorderLayout.CENTER);

        pnl[10].setLayout(new FlowLayout()); pnl[10].add(new
Jbutton(«one»));
        pnl[10].add(new Jbutton(«two»));
        pnl[10].add(new Jbutton(«three»));
        pnl[10].add(new Jbutton(«four»));
        pnl[10].add(new Jbutton(«fve»));
        setSize(800,500);
    }
    public static void main(String[]args) {
        new BorderExample().setVisible(true);
    }
}
```

Код, представленный выше, будет иметь вид, который указан на рисунке 4.4.

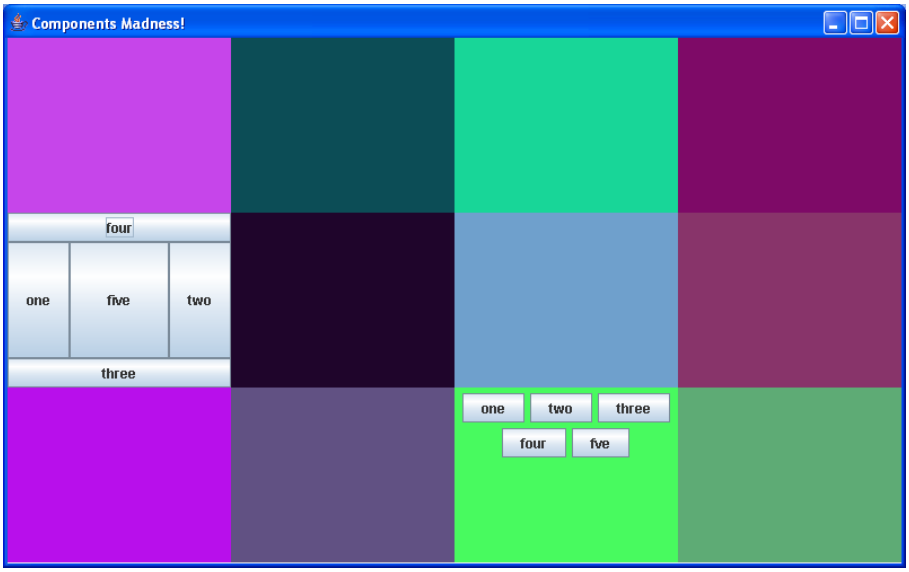

Рисунок 4.4 – Пример выполнения кода

Заметьте, что JFrame имеет GridLayout размера 3 на 4 (таблица), в то время как JPanel размером (2, 1) имеет менеджер BorderLayout. A JPanel (3, 3) имеет FLowLayout.

Менеджер Null Layout Manager.

Иногда бывает нужно изменить размер и расположение компонента в контейнере. Таким образом, мы должны указать программе не использовать никакой менеджер компоновки, то есть (setLayout (нуль)). Так что мы получим результат, отраженный на рисунке 4.5.

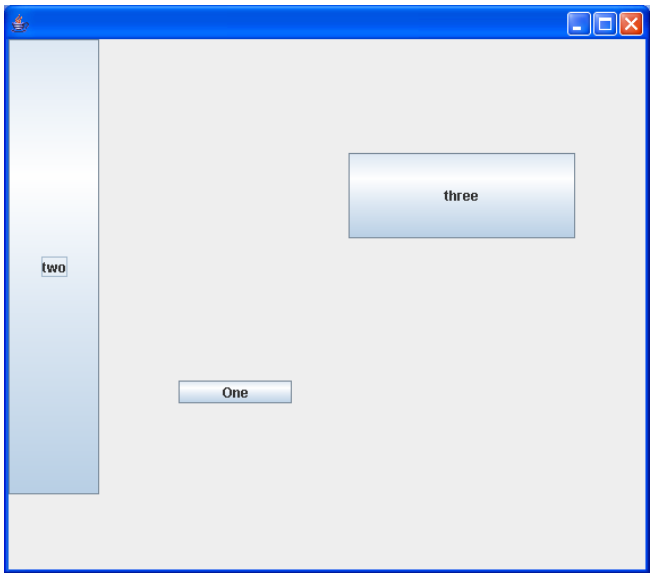

Рисунок 4.5 – Пример работы программы при setLayout равным нулю

Пример 5.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class NullLayout extends JFrame{
      JButton but1 = new JButton("One");;
      JButton but2 = new JButton("two");;
      JButton but3 = new JButton("three");;
      public NullLayout() {
        setLayout(null);
        but1.setBounds(150,300,100,20); // added at 150,300
width = 100,height=20
        but2.setSize(80,400); // added at 0,0 width = 80,
height=400
        but3.setLocation(300,100);
        but3.setSize(200,75);
        // those two steps can be combined in one setBounds
method call
        add(but1);
        add(but2);
        add(but3);
```

```
        setSize(500,500);
    }
    public static void main(String[]args){
        new NullLayout().setVisible(true);
    }
}
```

Слушатель событий мыши MouseListener.

Мы можем реализовывать слушателей мыши и также слушателей клавиатуры на компонентах GUI. Интерфейс MouseListener имеет следующие методы:

Таблица 2 - Методы класса MouseListener.

Методы класса		
Возвращаемое значение	Прототип метода	Описание
void	mouseClicked(MouseEvent e)	Вызывается, когда кнопка мыши была нажата (нажата и отпущена) на области компонента.
void	mouseEntered(MouseEvent e)	Вызывается, когда мышь входит в область компонент.
void	mouseExites(MouseEvent e)	Вызывается, когда мышь выходит из области компонента.
void	mousePressed(MouseEvent e)	Вызывается при нажатии кнопки мыши на область компонента.

Продолжение таблицы 2

void	mouseReleased(MouseEvent e)	Вызывается, когда над областью компонента отпущена кнопка мыши.

Слушатель MouseListener можно добавить к компоненту следующим образом:

Component.addMouseListener(listener);

Здесь слушатель является экземпляром класса, который реализует интерфейс MouseListener. Обратите внимание, что он должен обеспечивать выполнение всех методов, перечисленных в таблице 2 в данной практической работе. Пример 6.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class MyMouse extends JFrame {
    JLabel lbl = new JLabel("");
    public MyMouse()
{
        super("Dude! Where's my mouse ?");
        setSize(400,400);
        setLayout(new BorderLayout());
        add(lbl,BorderLayout.SOUTH);
        addMouseListener(new MouseListener() {
            public void mouseExited(MouseEvent a){}
            public void mouseClicked(MouseEvent a)
{lbl.setText("X="+a.getX()+" Y="+a.getY());}
            public void mouseEntered(MouseEvent a) {}
            public void mouseReleased(MouseEvent a) {}
            public void mousePressed(MouseEvent a) {}
        });
    }
    public static void main(String[]args){
        new MyMouse().setVisible(true);
    }
}
```

Рисунок 4.6 – Пример выполнения программы 6

Создание меню

Добавление меню в программе Java происходит несложно. Java определяет три компонента для обработки:

- JMenuBar: который представляет собой компонент, который содержит меню.
- JMenu: который представляет меню элементов для выбора.
- JMenuItem: представляет собой элемент, который можно кликнуть из меню.

Рисунок 4.7 – Меню в языке Java

Подобно компоненту Button (на самом деле MenuItems являются подклассами класса AbstractButton). Мы можем добавить ActionListener к ним так же, как мы делали с кнопками

ЗАДАНИЯ

Упражнение 1.

Напишите интерактивную программу с использованием GUI имитирует таблицу результатов матчей между командами Милан и Мадрид. Создайте JFrame приложение у которого есть следующие компоненты GUI:

- одна кнопка JButton labeled “AC Milan”
- другая JButton подписана “Real Madrid”
- надпись JLabel содержит текст “Result: 0 X 0”
- надпись JLabel содержит текст “Last Scorer: N/A”
- надпись Label содержит текст “Winner: DRAW”;

Всякий раз, когда пользователь нажимает на кнопку AC Milan, результат будет увеличиваться для Милана, сначала 1 X 0, затем 2 X 0 и так далее. Last Scorer означает последнюю забившую команду. В этом случае: AC Milan. Если пользователь нажимает кнопку для команды Мадрид, то счет приписывается ей. Победителем становится команда, которая имеет больше кликов кнопку на соответствующую, чем другая.