

## **Практическая работа №2. Объектно-ориентированное программирование в Джава. Классы в Джава.**

**Цель** данной практической работы - изучить основные концепции объектно-ориентированного программирования, изучить понятие класса и научиться создавать классы.

### **Теоретические сведения**

Для начала разберем, что такое модификаторы доступа в Java. В Java существуют следующие модификаторы доступа:

- `private`: данные класса доступны только внутри класса;
- `protected`: данные класса доступны внутри пакета и в наследниках;
- `public`: данные класса доступны всем.

В качестве примера рассмотрим программу, описывающую окружность. Создадим класс `Circle`. Каждая окружность имеет несколько параметров:

- Координаты по оси `x`;
- Координаты по оси `y`;
- Радиус `r`;
- Цвет `colour`.

Объявим переменные класса:

```
private double x;  
private double y;  
private double r;  
private String colour;
```

Как мы уже знаем, тип данных `double` приемняется для числовых значений, а `String` для текстовых.

Одним из стандартных подходов при проектировании классов на языке Джава является управление доступом к атрибутам класса через пару методов `get` и `set`. Метод `get` позволяет получить значение поля, `set` — установить новое значение. Общий принцип именования этих методов (называемых, также, геттером и сеттером). Для того, чтобы автоматически добавить эти методы в класс, щелкаем правой кнопкой мыши (или путем сочетания клавиш `Alt+Insert`) и из выпадающего меню

выбираем «Generate»<sup>1</sup>, а затем щелкаем по «Getter» или «Setter», в зависимости от того, что нам необходимо.

Листинг 2.1 – Пример сеттеров и геттеров для класса

```
public double getX()
{return x;}
public void setX(double x)
{this.x = x;}
public double getY()
{return y;}
public void setY(double y)
{this.y = y;}
public double getR()
{return r;}
public void setR(double r)
{this.r = r;}
public String getColour()
{return colour;}
public void setColour(String colour)
{this.colour = colour;}
```

Обратите внимание, что в теле метода автоматически прописались два вида строчек: return (для getter) и this. (для setter). Return пишется для возврата значений переменной (чтения), this. пишется в качестве ссылки на данные переменной.

Затем также, как мы вызывали методы getter и setter, вызываем метод конструктор класса (Constructor). Конмструктор — это специальный метод, который вызывается при создании нового объекта. Не всегда удобно инициализировать все переменные класса при создании его экземпляра. Иногда проще, чтобы какие-то значения были бы созданы по умолчанию при создании объекта. По сути, конструктор класса нужен для автоматической инициализации переменных.

Листинг 2.2 – Пример конструктора для класса Circle

```
public Circle(double x, double y, double r, String
colour) {
this.x = x;
```

---

<sup>1</sup> Корректно для IntelliJ IDEA

```

this.y = y;
this.r = r;
this.colour = colour;
}

```

Далее мы также вызываем еще один метод – toString(). Метод toString в Java используется для предоставления ясной и достаточной информации об объекте (Object) в удобном для человека виде.

Листинг 2.3 – Пример переопределения метода toString()

```

@Override
    public String toString() {
        return "Circle{" +
            "x=" + x + ", y=" + y +
            ", r=" + r +
            ", colour='" + colour + '\'' +
            '}';
    }

```

Последним этапом формирования класса Circle будет создание метода getLength(). В теле данного метода мы объявляем переменную *c*, которая обозначает длину нашей окружности. Длина окружности описывается формулой:  $c = 2 * \pi * r$ . В Java нельзя просто так написать  $\pi$ , поэтому следует обратиться к библиотеке Math. В конце тела метода необходимо вернуть значение *c*. Получается такая запись:

Листинг 2.3 – Пример функции класса getLength()

```

public double getLength() {
    double c;
    c = 2*Math.PI*r;
    return c;
}
}

```

Последняя фигурная скобка не является лишней – она закрывает тело нашего класса Circle.

Следующим этапом написания нашей программы будет создание в этом же пакете, в котором мы работали, класса Tester (часто называют Main). Затем добавляем необходимые библиотеки: lang и Scanner (в дальнейшем мы будем использовать ввод с клавиатуры):

```

import java.lang.*;

```

```
import java.util.Scanner;
```

Далее прописываем аббревиатуру “psvm” для создания статического метода `main(String[])`. Следующим шагом будет объявление переменной *r*. Второй раз мы объявляем переменную с таким же именем, но в другом классе, чтобы ввести значение *r* с клавиатуры и посмотреть, как меняется значение *c*:

```
public class Tester {  
    public static void main(String[] args) {  
        double r;
```

Затем мы создаем экземпляр (*k1*) класса `Circle`:

```
Circle k1 = new Circle(x:3.1, y:4.1, r:5.1,  
colour:"red");
```

Далее прописываем стандартную процедуру вывода значений:

```
System.out.println("Длина окружности = " +  
k1.getLength() + "см\n");
```

После ввода этой строчки на консоли выведется длина окружности *c*, при радиусе *k*, который мы ввели для экземпляра класса (*r*:5.1).

Включаем ввод с клавиатуры и пишем процедуру «Вывод», в которой будет содержаться информация о том, что нам необходимо ввести, и пишем переменную, которую нам необходимо будет ввести.

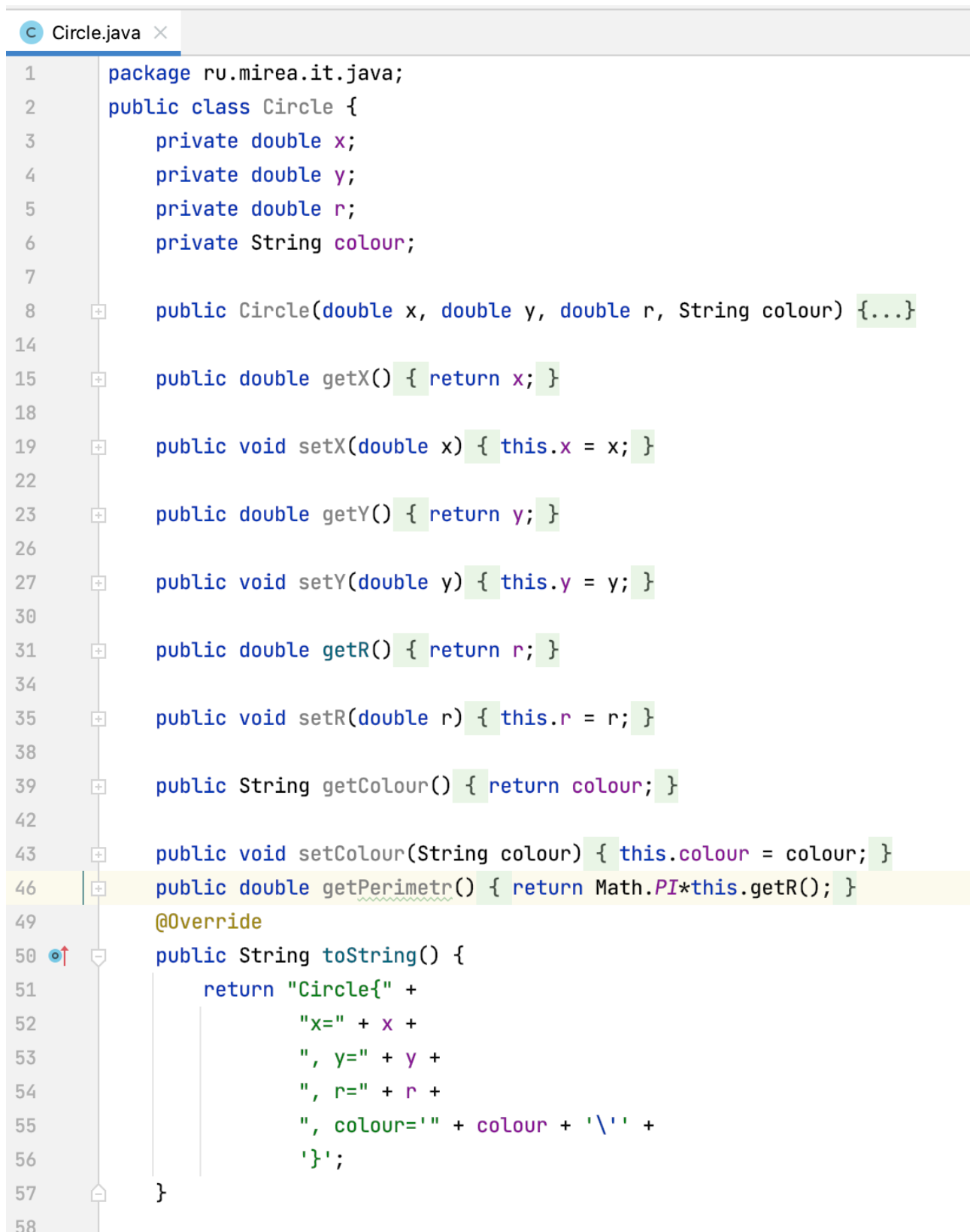
Делается это так: `r = source.nextDouble();`. В итоге получается следующее:

```
Scanner source = new Scanner(System.in);  
System.out.println("Введите радиус ");  
r = source.nextDouble();
```

Пишем `k1.setR(r);`. Это необходимо для того, чтобы при работе программа обращалась уже к новому значению *r*, а не к тому, которое мы ввели для экземпляра класса. Завершением класса `Tester` будет написание процедуры “Вывод”.

```
k1.setR(r);  
System.out.println("\nДлина окружности = " +  
k1.getLength() + "см");  
}  
}
```

В итоге получаем следующий код программы, включающий в себя 2 класса. На рис. 2.1 представлен класс `Circle`



```

1  package ru.mirea.it.java;
2  public class Circle {
3      private double x;
4      private double y;
5      private double r;
6      private String colour;
7
8      public Circle(double x, double y, double r, String colour) {...}
9
10
11
12
13
14
15      public double getX() { return x; }
16
17
18
19      public void setX(double x) { this.x = x; }
20
21
22
23      public double getY() { return y; }
24
25
26
27      public void setY(double y) { this.y = y; }
28
29
30
31      public double getR() { return r; }
32
33
34
35      public void setR(double r) { this.r = r; }
36
37
38
39      public String getColour() { return colour; }
40
41
42
43      public void setColour(String colour) { this.colour = colour; }
44
45
46      public double getPerimetr() { return Math.PI*this.getR(); }
47
48
49      @Override
50      public String toString() {
51          return "Circle{" +
52              "x=" + x +
53              ", y=" + y +
54              ", r=" + r +
55              ", colour='" + colour + '\'' +
56              '}';
57      }
58

```

Рисунок 2.1 Класс Circle.java

На рис. 2.2 представлен класс Tester в котором создается объект класса Circle и вызываются его методы.

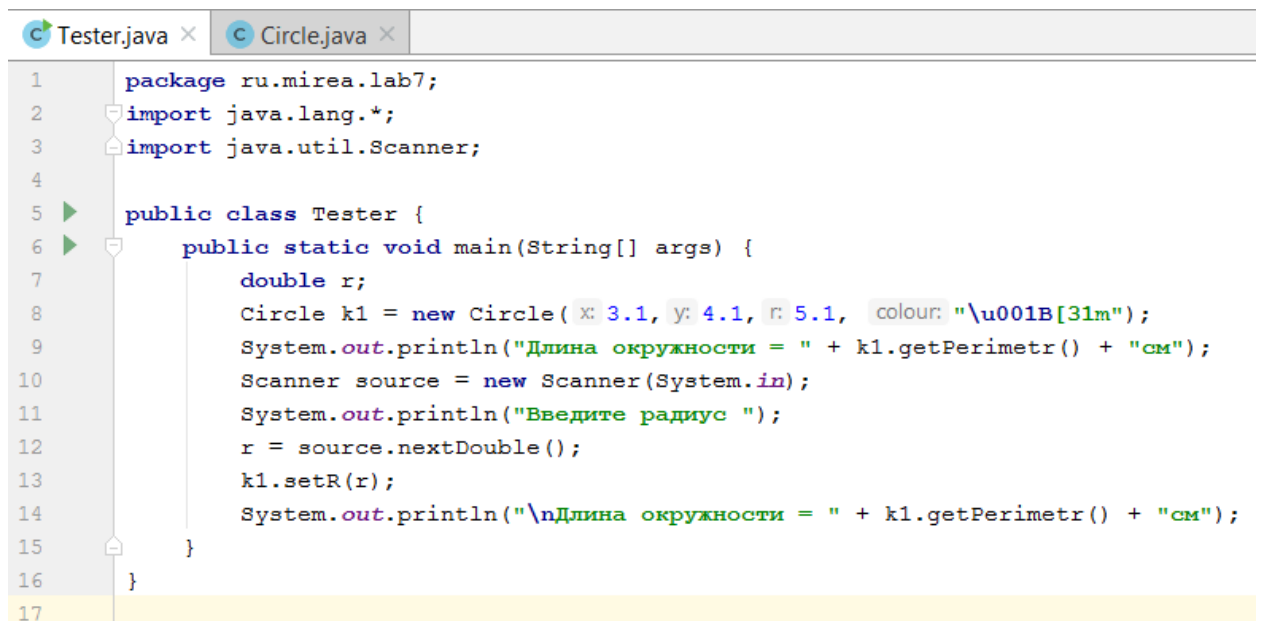


Рисунок 2.2 Класс Tester.java

На рисунке 2.3 приведен пример результат работы программы.

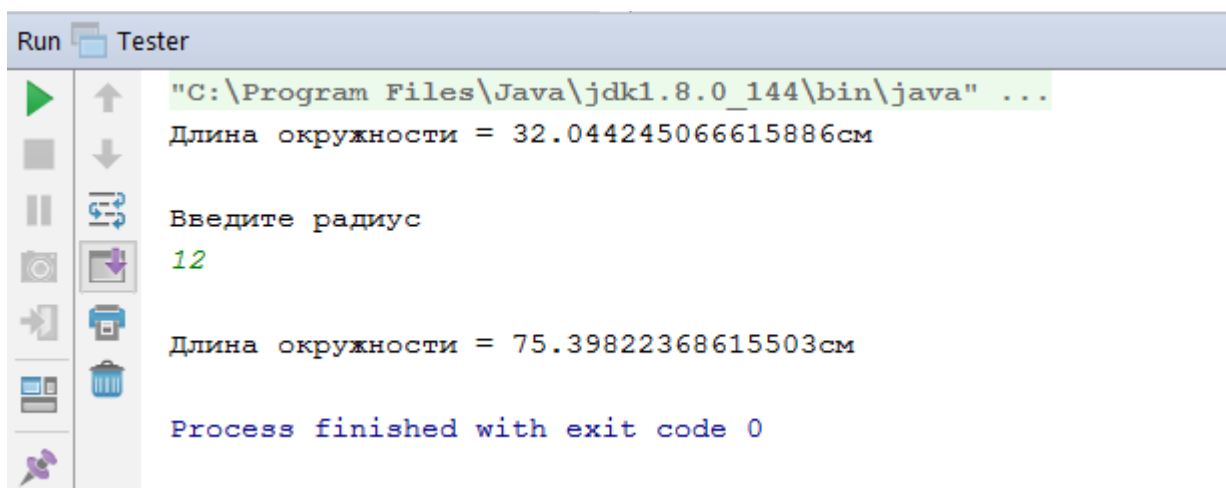
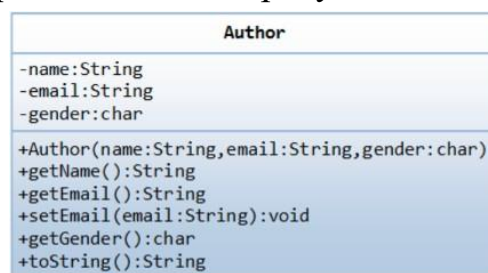


Рисунок 2.3 Результат работы Tester.java

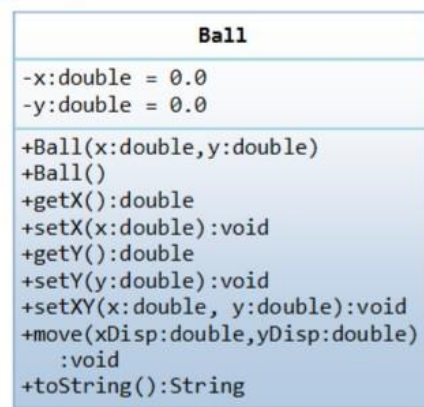
## Задания на практическую работу №2

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 3.4.



*Рисунок 3.4 Диаграмма класса Author.*

2. По UML диаграмме класса, представленной на рисунке 3.5. 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. Класс Ball моделирует движущийся мяч.



*Рисунок 3.5 Диаграмма класса Ball.*

3. Создать класс точка Point, описывающий точку на плоскости. Создать Circle класс, в котором одно поле представляет точку – центр окружности, и добавить другие свойства, позволяющие задать точку на плоскости. Создать третий класс Tester который использует для хранения объектов массив объектов Circle и второе поле количество элементов в массиве.

4. Разработайте класс Shop для, реализуйте методы добавления и удаления компьютеров в магазине, добавьте метод поиска в магазине компьютера, нужного пользователю. Протестируйте работу созданных классов. Данные для заполнения массива компьютеров вводятся с клавиатуры пользователем. Для этого реализуйте интерфейс.

5. Разработайте и реализуйте класс Dog (Собака), поля класса описывают кличку и возраст собаки. Необходимо выполнить следующие действия: определить конструктор собаки, чтобы принять и инициализировать данные экземпляра., включить стандартные методы (аксессоры) для получения и установки для имени и возраста, включить метод для перевода возраста собаки в “человеческий” возраст (возраст семь раз собаки), включите метод ToString, который возвращает описание

экземпляра собаки в виде строки. Создание класса тестера под названием ПитомникСобак, реализует массив собак и основной метод этого класса позволяет добавить в него несколько объектов собаки.

6. Создать класс, описывающий модель окружности (Circle). В классе должны быть описаны нужные свойства окружности и методы для получения и изменения этих свойств. Добавить методы для расчета площади круга и длины окружности, а также метод позволяющий сравнивать две окружности. При помощи класса CircleTest, содержащего статический метод main(String[] args), протестировать работу класса Circle.

7. Создать класс, описывающий книгу (Book). В классе должны быть описаны нужные свойства книги (автор, название, год написания и т. д.) и методы для получения, изменения этих свойств. Протестировать работу класса в классе BookTest, содержащим метод статический main(String[] args). Создать класс книжная полка, в котором поля данных класса это массив объектов типа книги (Book, и количество книг на книжной полке. Написать методы класса, которые возвращают книги с самым поздним и самым ранним сроком издания. Написать метод класса, позволяющий расставить книги на книжной полке в порядке возрастания года выпуска. Используйте реализацию отношений композиция классов<sup>2</sup>

8. Напишите программу, которая меняет местами элементы одномерного массива из String в обратном порядке. Не используйте дополнительный массив для хранения результатов.

9. Напишите программу Poker.java, которая должна имитировать раздачу карт для игры в покер. Программа получает число n, задаваемое с консоли пользователем, и раздает карты на n игроков (по 5 карт каждому) из перетасованной колоды. Разделяйте пять карт, выданных каждому игроку, пустой строкой.

10. Напишите программу HowMany.java, которая определит, сколько слов Вы ввели с консоли

---

<sup>2</sup> Композиция [https://ru.wikipedia.org/wiki/Диаграмма\\_классов](https://ru.wikipedia.org/wiki/Диаграмма_классов)



