|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИиППО)

Практические РАБОТы

по дисциплине «Программирование на языка Джава»

Выполнил студент группы ИКБО-30-22 *Сулейманов А.В.*

Принял старший преподаватель *Рачков А.В.*

Практические работы работа выполнены «\_\_\_»\_\_\_\_\_\_\_2023г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2023г.

Москва 2023

**Оглавление**

Практическая работа № 1 3

Практическая работа № 2 …

…

…

…

Практическая работа № 24 …

**Практическая работа № 1**

**Цель работы**

1. установить необходимое для разработки на Джава программное обеспечение.
2. научиться работать с программным обеспечение для разработки на Джава.
3. изучить синтаксис и основные управляющие конструкции языка Джава.

**Теоретическое введение**

Java - это высокоуровневый, многозадачный, объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (позднее приобретенной Oracle Corporation). Вот несколько ключевых аспектов Java для вашего доклада:

Одной из главных особенностей Java является её способность работать на разных операционных системах без изменения исходного кода. Это достигается благодаря виртуальной машине Java (JVM), которая переводит Java-код в байт-код, понимаемый JVM.

Java построена на основе концепции объектно-ориентированного программирования (ООП). Это позволяет разрабатывать модульные, легко поддерживаемые и масштабируемые приложения.

Java предоставляет обширную стандартную библиотеку классов, которая содержит множество готовых решений для разработки различных типов приложений, включая работу с сетью, базами данных, графическим интерфейсом и другими аспектами.

Java поддерживает многопоточность, что позволяет разрабатывать многозадачные приложения, способные эффективно использовать ресурсы многопроцессорных систем.

**Выполнение лабораторной работы**

*Задание:*

1. Создать проект в IntelliJ IDEA
2. Создать свой собственный Git репозиторий
3. Написать программу, в результате которой массив чисел создается с помощью инициализации (как в Си) вводится и считается в цикле сумма элементов целочисленного массива, а также среднее арифметическое его элементов результат выводится на экран. Использовать цикл for.
4. Написать программу, в результате которой массив чисел вводится пользователем с клавиатуры считается сумма элементов целочисленного массива с помощью циклов do while, while, также необходимо найти максимальный и минимальный элемент в массиве, результат выводится на экран.
5. Написать программу, в результате которой выводятся на экран аргументы командной строки в цикле for.
6. Написать программу, в результате работы которой выводятся на экран первые 10 чисел гармонического ряда (форматировать вывод).
7. Написать программу, которая с помощью метода класса, вычисляет факториал числа (использовать управляющую конструкцию цикла), проверить работу метода.
8. Результаты выполнения практической работы залить через IDE в свой репозиторий и продемонстрировать преподавателю.

*Решение:*

Для выполнения данной практической работы было установлено программное обеспечение «IntelliJ IDEA» с официального сайта и был создан собственный удалённый Git репозиторий на сайте github.com.

Для выполнения **задания №3** был написана программа, выполняющая следующие действия:

1. Создает объект Scanner для считывания ввода с консоли.
2. Просит пользователя ввести количество элементов в массиве.
3. Проверяет, что введенное значение является целым числом. Если нет, выводит ошибку и завершает программу.
4. Создает массив заданного размера.
5. Пользователь поочередно вводит значения для каждого элемента массива. Если введенное значение не является целым числом, программа завершает выполнение.
6. Считает сумму элементов в массиве и вычисляет среднее арифметическое.
7. Выводит сумму и среднее арифметическое на экран.
8. Закрывает Scanner для освобождения ресурсов.

Чтобы проверить, является ли введенное с консоли значение целым числом, используется метод класса Scanner.hasNextInt (рисунок 1).

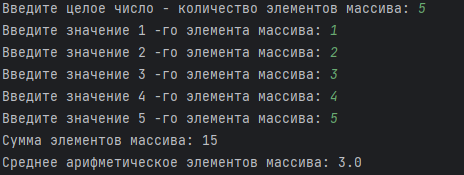
|  |
| --- |
| if (!scanner.hasNextInt()) {  System.*out*.println("Ошибка! Введено нецелое число");  System.*exit*(-1);  } |

Рисунок 1 – Проверка введенного значения

Для ввода элементов массива и определения их суммы и среднего значения использовался цикл for (рисунок 2).

|  |
| --- |
| for (int i = 0; i < sizeArray; i++) {  System.out.print("Введите значение " + (i + 1) + " -го элемента массива: ");  array[i] = scanner.nextInt();  sum += array[i];  }  double average = (double) sum / sizeArray; |

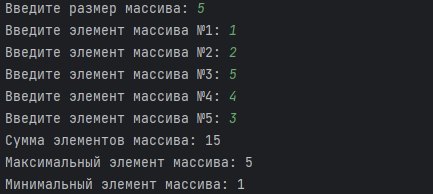
Рисунок 2 – Цикл for

Рисунок 3 – Результат работы задания №3

Для выполнения **задания №4** были написаны циклы do-while и while для ввода с консоли, проверки введенного значения, нахождения суммы элементов массива, а также поиска максимального и минимального значения элементов массива.

|  |
| --- |
| do {  System.out.print("Введите элемент массива №" + (i + 1) + ": ");  if (!scanner.hasNextInt()){  System.out.print("Ошибка! Введены неправильные параметры");  System.exit(-1);  }  arr[i] = scanner.nextInt();  i++;  } while (i < n);  while (i < n) {  sum += arr[i];  i++;  }  i = 1;  while (i < n) {  if (arr[i] > max) {  max = arr[i];  }  if (arr[i] < min) {  min = arr[i];  }  i++;  } |

Рисунок 4 – Циклы do-while и while

Рисунок 5 – Тестирование задания №4

Для выполнения **задания №5** был написана программа, выводящая все аргументы командной строки. Аргументы командной строки указаны в параметрах запуска программы в IntelliJ IDEA.

Полный код программы (рисунок 6):

|  |
| --- |
| public class Main {  public static void main(String[] args) {  if (args.length == 0) {  System.out.println("Аргументы отсутствуют.");  System.exit(0);  }  System.out.println("Аргументы командной строки:");  // Выводим аргументы в цикле for  for (int i = 0; i < args.length; i++) {  System.out.println("Аргумент " + (i + 1) + ": " + args[i]);  }  }  } |

Рисунок 6 – Полный код программы задания №5

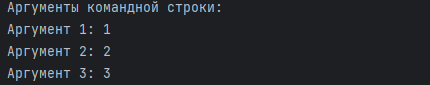


Рисунок 7 – Тестирование задания №5

Для выполнения **задания №6** был написана программа, выводящая первые десять чисел гармонического ряда при помощи форматированного вывода.

|  |
| --- |
| System.out.printf("Число %d: %.3f%n", i, val); |

Рисунок 8 – Форматированный вывод для задания №6

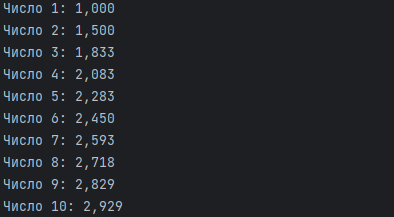


Рисунок 9 – Тестирование задания №6

Для выполнения **задания №7** был реализован метод calculateFactorial, определяющий факториал введенного числа при помощи цикла for.

|  |
| --- |
| public static long calculateFactorial(int n) {  if (n < 1 || n > 20) {  System.out.println("Введены неправильные параметры. Невозможно определить факториал");  System.exit(-1);  }  long factorial = 1;  for (int i = 1; i <= n; i++) {  factorial \*= i;  }  return factorial;  } |

Рисунок 10 – Метод calculateFactorial



Рисунок 11 – Тестирование задания №7

**Выводы по работе:**

В результате выполнения данной работы я скачал и научился работать с программным обеспечением для разработки Джава. Изучил синтаксис и основные управляющие конструкции языка Джава.

**Практическая работа № 2**

**Цель работы**

Работа с UML-диаграммами классов.

**Теоретическое введение**

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).

Методы (или поведение, функции, работа c данными): описывают динамическое поведение класса. Другими словами, класс инкапсулирует 33 статические свойства (данные) и динамические модели поведения (операции, которые работают с данными) в одном месте (“коробке” или прямоугольнике).

**Выполнение практической работы**

*Задание*

1. По диаграмме класса UML описывающей сущность Автор. Необходимо написать программу, которая состоит из двух классов **Author** и **TestAuthor**.
2. По UML диаграмме класса 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. Класс **Ball** моделирует движущийся мяч. В состав класса входят:

* Две переменные с модификатором private (поля данных класса): **х, у**, 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 х и у на данном участке на xDisp и yDisp, соответственно.
* Метод **toString**(), который возвращает "**Ball @ (х , у)** " .

*Решение*

Для решения первой задачи был описан класс **Author** с полями **name**, **email** и **gender**, параметризованным конструктором и методами **getName**, **getEmail**, **getGender**, **setEmail**, **toString**.

Этот класс позволяет представлять авторов книг и управлять информацией о них, включая изменение адреса электронной почты (с помощью метода **setEmail**) и получение информации об авторе (с помощью геттеров **getName**, **getEmail** и **getGender**). Метод **toString**() обеспечивает удобное форматирование информации для вывода.

|  |
| --- |
| @Override  public String toString() {  String genderStr;  if (gender == 'M') {  genderStr = "m";  } else if (gender == 'F') {  genderStr = "f";  } else {  genderStr = "u";  }  return name + " (" + genderStr + ") at " + email;  } |

Рисунок 12 – Метод форматированного вывода

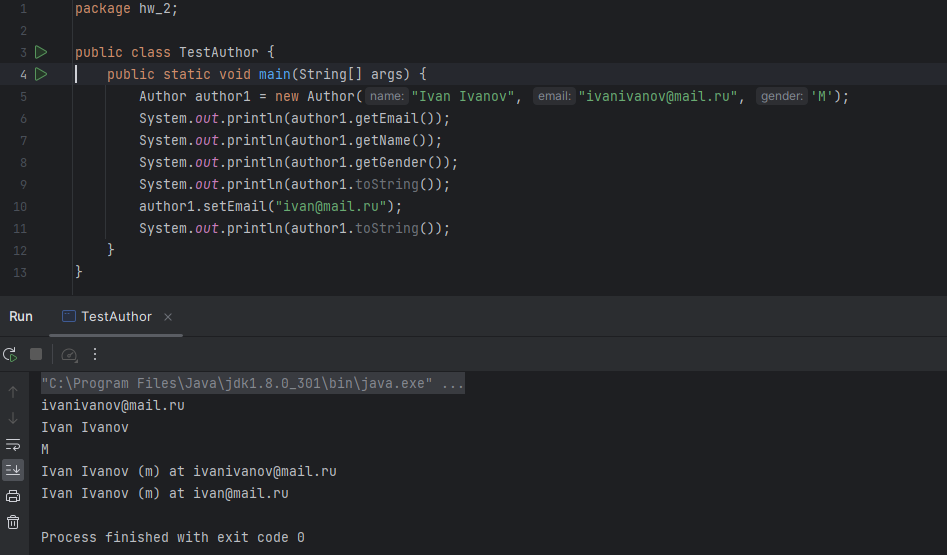


Рисунок 13 – Результат работы первого задания

Для выполнения второй задачи были реализованы классы **Ball** и **TestBall**. В классе **Ball** описаны:

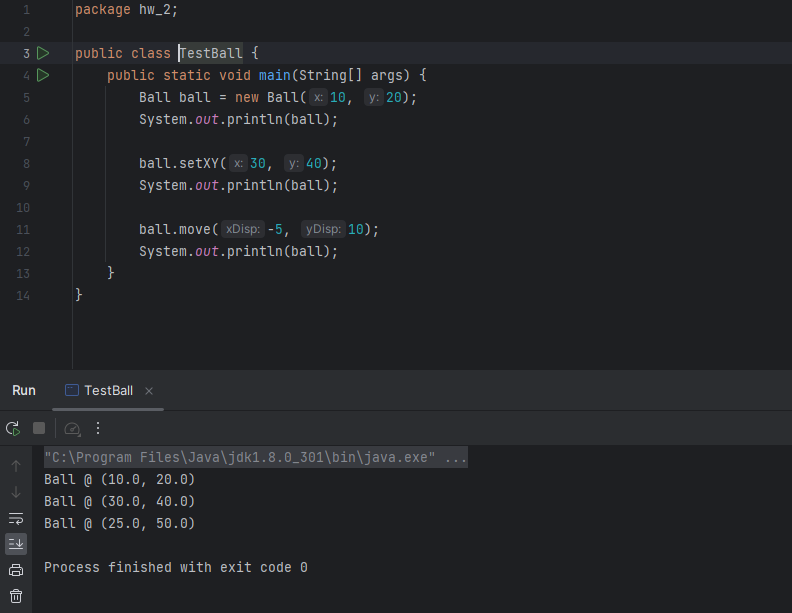
1. Поля x и y - это координаты мяча на плоскости, где x представляет горизонтальное положение мяча, а y - вертикальное положение.
2. Класс имеет два конструктора:

* Первый конструктор создает мяч с начальными координатами (0.0, 0.0).
* Второй конструктор позволяет создать мяч с заданными начальными координатами.

1. Геттеры и сеттеры (**getX(), getY(), setX(), setY()**) используются для получения и установки координат мяча.
2. Метод **setXY(float x, float y)** позволяет установить новые значения координат мяча одновременно.
3. Метод **move(float xDisp, float yDisp)** перемещает мяч на указанные смещения по горизонтали (**xDisp**) и вертикали (**yDisp**), изменяя его текущие координаты.
4. Метод **toString**() создает строку, представляющую мяч, в формате "Ball @ (X, Y)", где X и Y - текущие координаты мяча.

|  |
| --- |
| public void move(float xDisp, float yDisp) {  x += xDisp;  y += yDisp;  }  @Override  public String toString() {  return "Ball @ (" + x + ", " + y + ")";  } |

Рисунок 14 – Метод перемещения и форматированного вывода

Рисунок 15 – Тестирование класса Ball

**Выводы по работе**

В ходе данной практической работы я приобрел навыки работы с UML-диаграммами классов.

**Практическая работа № 3**

**Цель работы**

1. освоить на практике работу с абстрактными классами и наследованием на Java.

**Теоретическое введение**

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.

**Выполнение лабораторной работы**

*Задание*

1. Создайте абстрактный родительский суперкласс Shape и его дочерние классы (подклассы).
2. Перепишите суперкласс Shape и его подклассы, так как это представлено на диаграмме Circle, Rectangle and Square.
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.
4. Напишите два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable.

*Решение*

Для выполнения первого и второго упражнения были описан абстрактный класс Shape, который представляет абстрактную геометрическую фигуру. Этот класс служит в качестве базового класса для других конкретных геометрических фигур, таких как круг, прямоугольник, треугольник и другие.

Класс **Shape** является абстрактным, потому что нельзя создать экземпляр абстрактного класса напрямую. Он служит в качестве общего шаблона для конкретных геометрических фигур, которые будут созданы как его подклассы. Каждый подкласс (например, **Circle**, **Rectangle**, **Square**) должен реализовать методы **getArea()** и **getPerimeter()** в соответствии с собственной логикой для конкретной фигуры.

Класс **Circle** наследует от абстрактного класса **Shape** и представляет круг. В классе реализованы:

1. **Поля класса**:
   * **radius** (тип **double**): Хранит радиус круга.
   * **PI** (тип **static final float**): Константа, представляющая значение числа Пи (π), используется для вычисления площади и периметра круга.
2. **Конструкторы**:
   * **public Circle()**: Конструктор по умолчанию создает круг с радиусом 0 и базовыми значениями цвета и заполнения.
   * **public Circle(double radius)**: Пользовательский конструктор, позволяющий задать радиус круга при создании.
   * **public Circle(double radius, String color, boolean filled)**: Пользовательский конструктор, который также позволяет задать цвет и заполнение круга.
3. **Методы getArea() и getPerimeter()**: Переопределены из родительского класса **Shape** и реализованы для вычисления площади и периметра круга на основе его радиуса.
4. **Метод toString()**: Переопределен из родительского класса **Shape** и возвращает строковое представление объекта **Circle**, включая информацию о цвете, заполнении и радиусе.

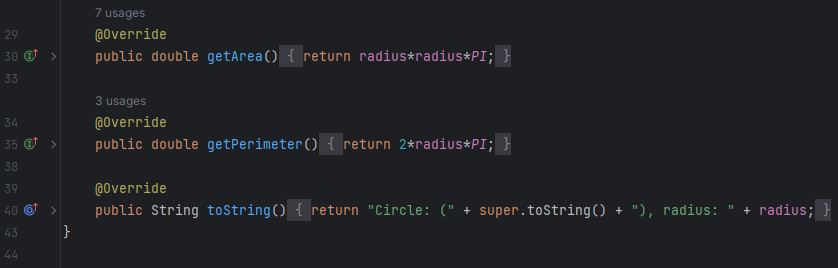
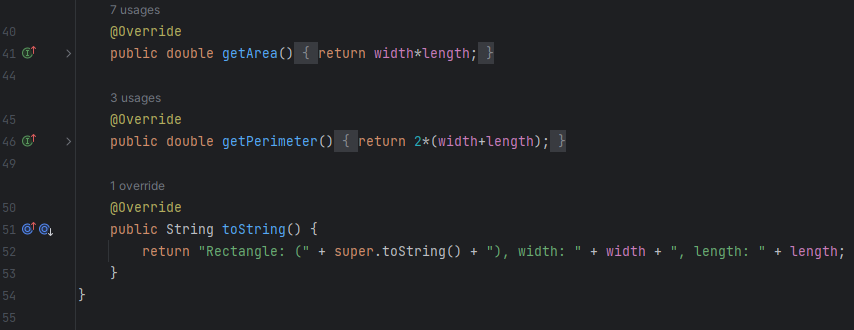


Рисунок 16 – Реализация методов класса Circle

Класс **Rectangle** наследует от абстрактного класса **Shape** и представляет прямоугольник. Вот краткое описание его работы:

1. **Поля класса**:
   * **width** (тип **double**): Хранит ширину прямоугольника.
   * **length** (тип **double**): Хранит длину прямоугольника.
2. **Методы getArea() и getPerimeter()**: Переопределены из родительского класса **Shape** и реализованы для вычисления площади и периметра прямоугольника на основе его ширины и длины.
3. **Метод toString()**: Переопределен из родительского класса **Shape** и возвращает строковое представление объекта **Rectangle**, включая информацию о цвете, заполнении, ширине и длине.

Рисунок 16 – Реализация методов класса Rectangle

Класс **Square** является подклассом класса **Rectangle. Square** предоставляет специфичную реализацию для квадрата, который является частным случаем прямоугольника, и позволяет создавать объекты квадратов с различными параметрами стороны, цвета и заполнения, при этом наследуя функциональность и характеристики от класса Rectangle.

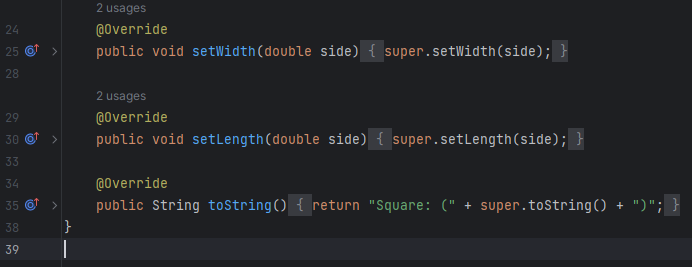


Рисунок 17 – Реализация методов класса Square

Для выполнения третьего задания был описан класс TestShape, предназначенный для тестирования первого и второго упражнения. Ошибки, возникшие в результате компиляции кода:

1. **Shape s1 = new Circle(5.5, "RED", false);** - Эта строка создает объект **Circle**, но он сохраняется в переменной типа **Shape**. В результате доступны только методы, объявленные в классе **Shape**, и не доступен метод **getRadius()**, так как он не объявлен в **Shape**.
2. **Shape s3 = new Rectangle(1.0, 2.0, "RED", false);** - Здесь создается объект **Rectangle**, который также сохраняется в переменной типа **Shape**. Снова доступны только методы **Shape**, и методы, специфичные для **Rectangle**, такие как **getLength()**, не доступны.
3. **Shape s4 = new Square(6.6);** - Здесь создается объект **Square**, сохраненный в переменной типа **Shape**. Снова доступны только методы **Shape**, и методы, специфичные для **Square**, такие как **getSide()**, не доступны.
4. **Rectangle r2 = (Rectangle)s4;** - Эта строка пытается выполнить downcast объекта **s4** обратно к типу **Rectangle**, что допустимо, так как **s4** был создан как **Square**. Однако это вызывает потенциальную проблему: **Square** наследует от **Rectangle**, но **Square** - это более специфичный тип, и некоторые методы **Rectangle** (например, **getSide()**) могут быть неадекватными для квадрата. Таким образом, вызов метода **getSide()** для **r2** могут вызвать ошибку времени выполнения, так как они не определены в **Rectangle**.
5. Попытка создать экземпляр абстрактного класса **Shape** также приводит к ошибке.

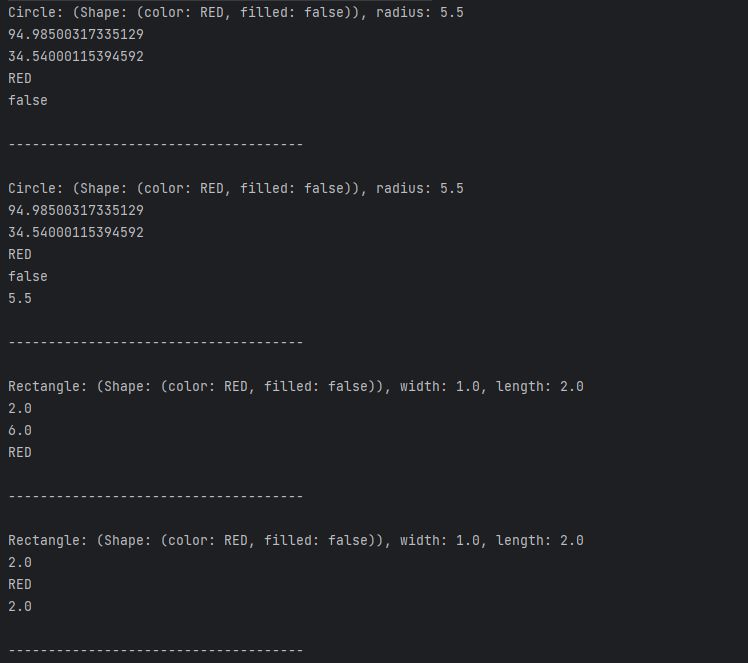


Рисунок 18 – Тестирование задания №3

Для выполнения четвертого задания был описан код, который определяет интерфейс **Movable**, который имеет четыре метода для перемещения объекта вверх, вниз, вправо и влево. Далее описаны два класса, **MovablePoint** и **MovableCircle**, которые реализуют этот интерфейс.

1. **MovablePoint** представляет точку с координатами **(x, y)** и скоростями перемещения **(xSpeed, ySpeed)**. Он реализует методы **moveUp**, **moveDown**, **moveRight** и **moveLeft**, которые изменяют координаты точки соответствующим образом.
2. **MovableCircle** представляет круг с центром в объекте **MovablePoint** и радиусом. Он также реализует методы интерфейса **Movable**, но вместо непосредственного изменения своих координат, он делегирует вызовы этих методов своему внутреннему объекту **center**, который является экземпляром **MovablePoint**.

В методе **main** происходит создание экземпляров **MovablePoint** и **MovableCircle**, их инициализация начальными значениями, а затем вызов методов перемещения и вывод текущих состояний объектов. Например, для **MovablePoint** выводится начальное положение точки, затем она перемещается влево и вверх, и выводится её новое положение. Аналогичные действия выполняются и для **MovableCircle**.

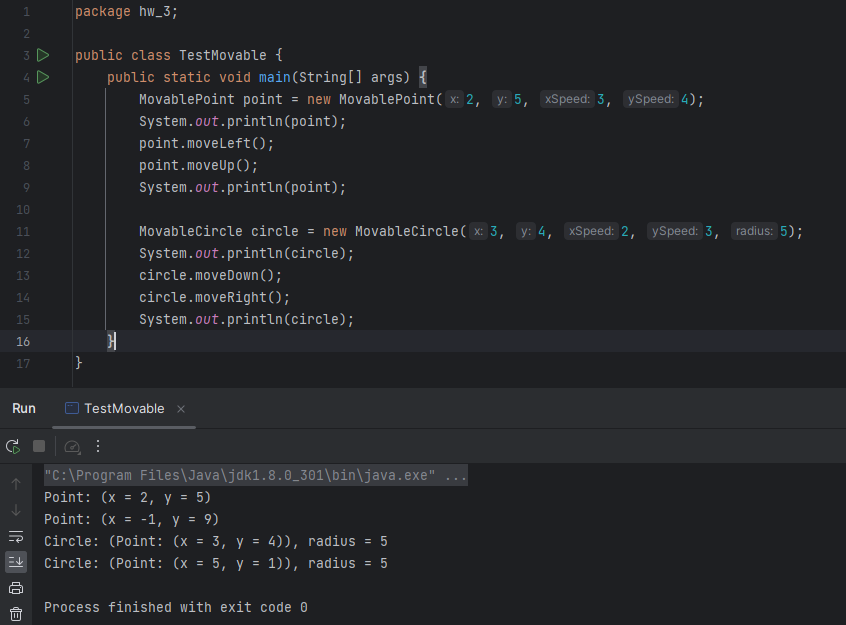


Рисунок 19 – Тестирование задания №4

**Выводы по работе**

В ходе данной практической работы мной был приобретен навык работы с абстрактными классами и наследованием на Java.

**Используемая литература *(в конце всего отчета)***

*Список литературы тоже должен быть оформлен по ГОСТ. Первым пунктом можно указать:*

1. *Конспект лекций по дисциплине «Программирование на языке Джава», РТУ МИРЭА, лектор – старший преподаватель Зорина Н.В.*
2. *Карпова, И.П. Базы данных: Учебное пособие / И.П. Карпова. –СПб.: Питер, 2013. – 240 с.*
3. *Фрэйн Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств – Питер: 2016г. Режим доступа свободный: https://www.htbook.ru/kompjutery\_i\_seti/setevye\_tekhnologii/html5-i-css3-razrabotka-sajtov-dlja-ljubykh-brauzerov-i-ustrojstv;*
4. *Справочник по языку PHP [Электронный ресурс]:php.su— Режим доступа свободный: http://www.php.su;*
5. *И т.д.*

*ОБРАТИТЕ ВНИМАНИЕ, ЧТО ТОЛЬКО ПЕРВЫЙ ИСТОЧНИЕ ОТНОСИТСЯ К КУРСУ!*