МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена трудового Красного Знамени федеральное государственное бюджетное

образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Программная инженерия»

Отчет по лабораторной работе 2

Объектно-ориентированное программирование

Выполнила: студентка группы БПИ2401

Садовникова Евгения

Проверил: Харрасов Камиль Раисович

Москва, 2025

Содержание

Цель работы: создать иерархию классов на языке Java с использованием абстрактного базового класса и наследуемых классов, реализовать основные принципы объектно-ориентированного программирования (ООП):

- абстракцию;
- инкапсуляцию;
- наследование;
- полиморфизм.

Задания

Создайте иерархию классов в соответствии с вариантом. Ваша иерархия должна содержать:

• абстрактный класс;

• два уровня наследуемых классов (классы должны содержать в себе минимум 3 поля и 2 метода, описывающих поведение объекта);

• демонстрацию реализации всех принципов ООП;

• наличие конструкторов (в том числе по умолчанию);

• наличие геттеров и сеттеров;

• ввод/вывод информации о создаваемых объектах;

• предусмотрите в одном из классов создание счетчика созданных объектов с использованием статической переменной, продемонстрируйте работу.

Вариант 9.

Базовый класс: Монстр. Дочерние классы: Гоблин, Русалка, Дракон.

Ход работы:

Создадим абстрактный класс Monster с общими полями: name, habitat, notableFeatures и абстрактным методом getMovement(). (листинг 1)

Листинг 1. class Monster

```
public abstract class Monster {
    private String name;
```

```
        protected String habitat; // естественная среда обитания
        protected String notableFeatures; // отличительные черты
        private static int monsterCount = 0; // счетчик монсторов

            public Monster(String name, String habitat, String
notableFeatures) {
            this.name = name;
            this.habitat = habitat;
            this.notableFeatures = notableFeatures;
            monsterCount++;
        }

        public Monster() {
            this("Monster", "unknown", "unknown");
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        public String getHabitat() {
            return habitat;
        }

        public void setHabitat(String habitat) {
            this.habitat = habitat;
        }

        public String getNotableFeatures() {
            return notableFeatures;
        }

        public void setNotableFeatures(String notableFeatures) {
            this.notableFeatures = notableFeatures;
        }

        public static int getMonsterCount() {
            return monsterCount;
        };

        public void printDescription() {
            System.out.println("name: " + getName()
                    + "\nhabitat: " + habitat +
                    "\nnotable Features: " + notableFeatures);
        }

        abstract String getMovement();

    }
    public static void main(String[] args) {
            System.out.println("is Prime:");
            for (int i = 1; i <= 100; i++) {
```

```
                if (isPrime(i)) {
                    System.out.printf(i + " ");
                }
            }
        }
```

Создадим абстрактный класс Goblin с полями: noseLength, height, footSize. Переопределяет методы getMovement() и printDescription(). (листинг 2)

Листинг 2. class Goblin

```
    public class Goblin extends Monster {
        private double noseLength; // длина носа
        private double height; // рост
        private double footSize; // размер ноги

            public Goblin(String name, String habitat, String
notableFeatures,
                double noseLength, double height, double footSize) {
            super(name, habitat, notableFeatures);
            this.noseLength = noseLength;
            this.height = height;
            this.footSize = footSize;
        }

        public Goblin() {
            this("unknown", "unknown", "unknown",
                    0.0, 0.0, 0.0);
        }

        public double getNoseLength() {
            return noseLength;
        }

        public void setNoseLength(double noseLength) {
            this.noseLength = noseLength;
        }

        public double getHeight() {
            return height;
        }

        public void setHeight(double height) {
            this.height = height;
        }

        public double getFootSize() {
            return footSize;
        }

        public void setFootSize(double footSize) {
            this.footSize = footSize;
        }
```

```
        @Override
        public String getMovement() {
            return getName() + " goes"; // ходит
        }

        @Override
        public void printDescription() {
            super.printDescription();
            System.out.println("nose Length: " + noseLength
                    + "\nheight: " + height
                    + "\nfoot Size: " + footSize);
        }
    }
```

Создадим абстрактный класс Mermaid с полями: tailLength, hairColor, vocalEvaluation. Переопределяет методы getMovement() и printDescription(). (листинг 3)

Листинг 3. class Mermaid

```
    public class Mermaid extends Monster {
        private double tailLength; // длина хвоста
        private String hairColor; // цвет волос
        private int vocalEvaluation; // оценка вокала

            public Mermaid(String name, String habitat, String
notableFeatures,
                double tailLength, String hairColor, int vocalEvaluation)
{
            super(name, habitat, notableFeatures);
            this.tailLength = tailLength;
            this.hairColor = hairColor;
            this.vocalEvaluation = vocalEvaluation;
        }

        public Mermaid() {
            this("Mermaid", "unknown", "unknown",
                    0.0, "unknown", 0);
        }

        public double getTailLength() {
            return tailLength;
        }

        public void setTailLength(double tailLength) {
            this.tailLength = tailLength;
        }

        public String getHairColor() {
            return hairColor;
        }

        public void setHairColor(String hairColor) {
            this.hairColor = hairColor;
```

```
    }

    public int getVocalEvaluation() {
        return vocalEvaluation;
    }

    public void setVocalEvaluation(int vocalEvaluation) {
        this.vocalEvaluation = vocalEvaluation;
    }

    @Override
    public String getMovement() {
        return getName() + " floating"; // плывет
    }

    @Override
    public void printDescription() {
        super.printDescription();
        System.out.println("tail Length: " + tailLength
                + "\nhair Color: " + hairColor
                + "\nvocalEvaluation: " + vocalEvaluation);
    }
}
```

Создадим абстрактный класс Dragon с полями: wingLength, wingWidth, numberHeads. Переопределяет методы getMovement() и printDescription(). Имеет перегрузку метода getMovement(double distance). (листинг 4)

Листинг 4. class Dragon

```
public class Dragon extends Monster {
    protected double wingLength; // длина крыльев
    protected double wingWidth; // ширина крыльев
    protected int numberHeads; // количество голов

        public Dragon(String name, String habitat, String
notableFeatures,
            double wingLength, double wingWidth, int numberHeads) {
        super(name, habitat, notableFeatures);
        this.wingLength = wingLength;
        this.wingWidth = wingWidth;
        this.numberHeads = numberHeads;
    }

    public Dragon() {
        this("unknown", "unknown", "unknown",
                0.0, 0.0, 0);
    }

    public double getWingLength() {
        return wingLength;
    }

    public void setWingLength(double wingLength) {
        this.wingLength = wingLength;
```

```
        }

        public double getWingWidth() {
            return wingWidth;
        }

        public void setWingWidth(double wingWidth) {
            this.wingWidth = wingWidth;
        }

        public int getNumberHeads() {
            return numberHeads;
        }

        public void setNumberHeads(int numberHeads) {
            this.numberHeads = numberHeads;
        }

        @Override
        public String getMovement() {
            return getName() + " flies"; // летает
        }

        public String getMovement(double distance) {// растояние
             return getName() + " flies at a distance " + distance; //
летает
        }

        @Override
        public void printDescription() {
            super.printDescription();
            System.out.println("wingLength: " + wingLength
                    + "\nwingWidth: " + wingWidth
                    + "\nnumberHeads: " + numberHeads);
        }
    }
```

Создадим абстрактный класс Wyvern потомок дракона. Переопределяет методы getMovement() и printDescription(). (листинг 5)

Листинг 5. class Wyvern

```
    public class Wyvern extends Dragon {
            public   Wyvern(String   name,   String   habitat,   String
notableFeatures,
                double wingLength, double wingWidth) {
            super(name, habitat, notableFeatures,
                    wingLength, wingWidth, 2);
        }

        public Wyvern() {
            this("unknown", "unknown", "unknown",
                    0.0, 0.0);
        }

        @Override
```

```
        public String getMovement() {
            return getName() + " flies with two heads"; // летает с двумя
головами
        }

        @Override
        public void printDescription() {
            super.printDescription();
            System.out.println("Type: Wyvern");
        }
    }
```

Создадим класс Main

```
    public class Main {
        public static void main(String[] args) {
             Goblin goblin = new Goblin("Goblin", "Cave", "Green and
sneaky", 0.4, 1.1, 0.3);
             Mermaid mermaid = new Mermaid("Ariel", "Sea", "Beautiful
voice", 2.0, "red", 10);
              Dragon dragon = new Dragon("Smaug", "Mountain", "Fire-
breathing", 6.0, 3.5, 3);
            Wyvern wyvern = new Wyvern("Twinfang", "Cliffs", "Swift and
deadly", 4.5, 2.8);
            Wyvern wyvern2 = new Wyvern();
             Monster monster = new Goblin(); // upcasting (повышающее
преобразование)

            goblin.printDescription();
            System.out.println(goblin.getMovement());
            System.out.println();

            mermaid.printDescription();
            System.out.println(mermaid.getMovement());
            System.out.println();

            dragon.printDescription();
            System.out.println(dragon.getMovement());
            System.out.println(dragon.getMovement(100));
            System.out.println();

            wyvern.printDescription();
            System.out.println(wyvern.getMovement());
            System.out.println();

            wyvern2.printDescription();
            System.out.println(wyvern2.getMovement());
            System.out.println();

             if (monster instanceof Goblin) { // инструкцию instanceof
для проверки наследования между объектами
                 Goblin goblin2 = (Goblin) monster; // downcasting
(понижающее преобразование)
                goblin2.printDescription();
                System.out.println(goblin2.getMovement());
                System.out.println();
```

```
            }
                    System.out.println("Total   monsters   created:   "   +
Monster.getMonsterCount());
        }
    }
```

Выходные данные:

Изображены на рисунке 1–2

```
PS C:\Users\Evgen\Desktop\ИТИП> javac Mermaid.java
PS C:\Users\Evgen\Desktop\ИТИП> javac Goblin.java
PS C:\Users\Evgen\Desktop\ИТИП> javac Dragon.java
PS C:\Users\Evgen\Desktop\ИТИП> javac Monster.java
PS C:\Users\Evgen\Desktop\ИТИП> javac Wyvern.java
PS C:\Users\Evgen\Desktop\ИТИП> java Main.java
name: Goblin
habitat: Cave
notable Features: Green and sneaky
nose Length: 0.4
height: 1.1
foot Size: 0.3
Goblin goes

name: Ariel
habitat: Sea
notable Features: Beautiful voice
tail Length: 2.0
hair Color: red
vocalEvaluation: 10
Ariel floating

name: Smaug
habitat: Mountain
notable Features: Fire-breathing
wingLength: 6.0
wingWidth: 3.5
numberHeads: 3
Smaug flies
Smaug flies at a distance 100.0

name: Twinfang
habitat: Cliffs
notable Features: Swift and deadly
wingLength: 4.5
wingWidth: 2.8
numberHeads: 2
Type: Wyvern
Twinfang flies with two heads

name: unknown
habitat: unknown
notable Features: unknown
wingLength: 0.0
wingWidth: 0.0
numberHeads: 2
Type: Wyvern
unknown flies with two heads
```

Рисунок 3. Результат программы Main.java

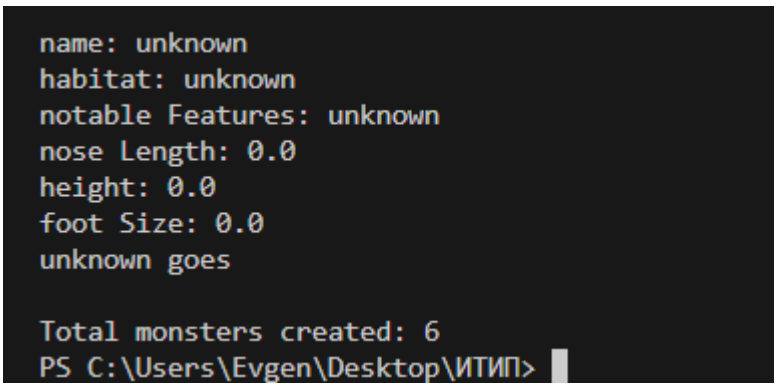

Рисунок 4. Результат программы Main.java

Контрольные вопросы

1. Что такое абстракция и как она реализуется в языке Java?

Абстракция подразумевает выделение наиболее значимых характеристик объекта, скрывая детали его реализации. Абстрактный класс или интерфейс предоставляет общий шаблон поведения, оставляя конкретные реализации подклассам.

```
abstract class Shape {
    abstract double area();
}

class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    double area() {
        return Math.PI * radius * radius;
    }
}

class Rectangle extends Shape {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    @Override
    double area() {
        return width * height;
    }
}

public class Main {
    public static void main(String[] args) {
        Shape circle = new Circle(5);
```



```
        Shape rectangle = new Rectangle(10, 20);

        System.out.println(circle.area()); // 78.53981633974483
        System.out.println(rectangle.area()); // 200.0
    }
```

Абстрактный класс Shape определяет абстрактный метод area(), который должен быть реализован в каждом классе-наследнике. Классы Circle и Rectangle предоставляют свои реализации метода area(), соответствующие их форме.

2. Что такое инкапсуляция и как она реализуется в языке Java?

Инкапсуляция заключается в сокрытии внутренней реализации объекта от внешнего мира и предоставлении доступа к данным только через определенные методы класса. Это позволяет защитить данные от несанкционированного изменения и повысить надежность кода.

```
public class Account {
    private double balance;

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }

    public void withdraw(double amount) {
        if (balance >= amount && amount > 0) {
            balance -= amount;
        } else {
            System.out.println("Недостаточно средств для снятия.");
        }
    }

    public double getBalance() {
        return balance;
    }
}
```

В этом примере переменная balance объявлена как приватная (private), что означает, что доступ к ней возможен только внутри класса. Внешний мир может взаимодействовать с объектом только через публичные методы (deposit, withdraw, getBalance), которые контролируют доступ к данным

3. Что такое наследование и как они реализуется в языке Java?

Наследование позволяет создавать новые классы на основе существующих, наследуя их свойства и поведение. Это помогает избежать дублирования кода и упрощает его поддержку.

```
class Animal {
    protected String name;

    public void eat() {
      System.out.println(name + " ест.");
    }
}

class Dog extends Animal {
    public void bark() {
      System.out.println(name + " лает.");
    }
}

public class Main {
    public static void main(String[] args) {
      Dog dog = new Dog();

      dog.name = "Шарик";
      dog.eat(); // Шарик ест.
      dog.bark(); // Шарик лает.
    }
```

Здесь класс Dog наследует все поля и методы класса Animal. Класс Dog добавляет новый метод bark() и использует унаследованный метод eat().

4. Что такое полимофизм и как он реализуется в языке Java?

Полиморфизм позволяет объектам разных типов реагировать по-разному на одни и те же сообщения. Полиморфизм бывает двух видов: статический (перегрузка методов) и динамический (переопределение методов).

```
1.  class Calculator {
2.        public int add(int a, int b) {
3.           return a + b;
4.        }
5.
```



```
    public double add(double a, double b) {
       return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
       Calculator calc = new Calculator();
       System.out.println(calc.add(5, 7)); // 12
       System.out.println(calc.add(5.5, 7.5)); // 13.0
    }
}
```

Метод add ерегружен дважды: один принимает два целых числа, другой — два числа с плавающей точкой.

5. Что такое множественное наследование и есть ли оно в Java?

Java не поддерживает множественное наследование через классы. Это означает, что класс может расширять (наследовать) только один другой класс.

Однако Java предоставляет альтернативу множественному наследованию через интерфейсы. Множественное наследование может привести к проблемам, известным как «проблема ромба» (или проблема бриллианта).

Интерфейсы позволяют классу реализовать несколько контрактов (интерфейсов), тем самым предоставляя функциональность, схожую с множественным наследованием. Интерфейсы содержат только объявления методов без их реализации, а также могут содержать статические константы и методы по умолчанию.

6. Для чего нужно ключевое слово final?

final в Java используется для **ограничения изменений**:

Переменная: её значение нельзя изменить после инициализации.

Метод: его нельзя переопределять в подклассах.

Класс: его нельзя наследовать.

в Java мы не можем расширять final-классы;

7. Какие в Java есть модификаторы доступа?

1) public — элементы с этим модификатором доступны везде, то есть они могут использоваться любым классом, вне зависимости от того, где он находится;

2) protected — доступ к элементам с таким модификатором возможен только для классов в том же пакете или для подклассов этого класса, даже если они находятся в другом пакете;

3) без модификатора (по умолчанию) — если ни один модификатор не указан, то элементы будут видны только в пределах пакета, но недоступны за его пределами;

4) private — элементы с таким модификатором доступны только внутри класса, в котором они объявлены. Они невидимы снаружи, даже для наследников.

8. Что такое конструктор? Какие типы конструкторов бывают в Java?

Конструктор – это специальный метод класса, который вызывается при создании объекта, чтобы **инициализировать его поля**.

Типы конструкторов:

Конструктор по умолчанию – без параметров.

Параметризованный конструктор – с параметрами.

9. Для чего нужно ключевое слово this в Java?

this – ссылка на текущий объект, в котором выполняется метод или конструктор.

Используется для:

- Различия между полями и параметрами метода/конструктора:
- Вызова другого конструктора текущего класса
- Передачи текущего объекта в метод или конструктор

10. Для чего нужно ключевое слово super в Java?

super – ссылка на объект суперкласса.

Используется для:

- Вызова конструктора суперкласса:
- Доступа к полям суперкласса, если они скрыты полями подкласса:
- Вызова метода суперкласса, если он переопределён в подклассе:

11. Что такое геттеры и сеттеры? Зачем они нужны?

Геттер (getter) – метод для получения значения поля.

Сеттер (setter) – метод для установки значения поля.

Используются для инкапсуляции: поля обычно private, доступ к ним только через методы.

Зачем:

- Контроль доступа к полям
- Валидация значений
- Сохранение инвариантов объекта

12. Что такое переопределение?

Переопределение (overriding) – это когда подкласс реализует метод суперкласса с той же сигнатурой (имя + параметры).

13. Что такое перегрузка?

Перегрузка – это создание **нескольких методов с одинаковым именем, но разными параметрами** в одном классе.

Теоретические сведения

• мы можем создать экземпляр подкласса и затем присвоить его переменной суперкласса, это называется upcasting (повышающее преобразование);

• когда экземпляр суперкласса присваивается переменной подкласса, это называется downcasting (понижающее преобразование). Нам необходимо явно привести этот экземпляр к подклассу;

• мы можем использовать инструкцию instanceof для проверки наследования между объектами;

Вывод

В лабораторной работе создана иерархия абстрактного класса и подклассов, продемонстрированы принципы ООП, переопределение методов и использование статической переменной для подсчёта объектов.

Ссылка на GitHub: https://github.com/Evgeshhha/Information-technology-and-programming