МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Программная инженерия»

Отчёт по Лабораторной работе №2

По дисциплине: «Информационные технологии и программирование»

«Объектно-ориентированное программирование в Java»

Выполнила: студентка группы БПИ2401

Алексеева Татьяна Игоревна

Проверил: Харрасов Камиль Раисович

Москва

2025

Цель работы

Изучить и применить на практике основные принципы объектно-ориентированного программирования в Java: инкапсуляцию, наследование, полиморфизм и абстракцию, реализовав иерархию классов.

Выполнение работы

Нам нужно создать иерархию классов в соответствии с вариантом. 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;
- счётчик созданных объектов с использованием статической переменной в одном из классов, работу которого затем нужно продемонстрировать.

1 Вариант: базовый класс – Животные, дочерние классы – кошка, попугай, рыбка.

Начнем работу с создания базового (абстрактного) класса. Он задает общие характеристики и поведение, присущие всем животным (имя, возраст, вес, способность есть, издавать звуки и двигаться).

```
package animals;

abstract class Animal {
    private String name;
    private int age;
    private double weight;

    public Animal () {
        this("Безымянный", 0, 0.0);
    }

    public Animal(String name, int age, double weight) {
        this.name = name;
        this.age = age;
        this.weight = weight;
    }

    public abstract void makeSound();
    public abstract void move();
```

```
    public void eat(String food) {
        System.out.println(getName() + " ест " + food);
    }

    public String Info() {
        return "Имя: " + name + ", возраст: " + age + ", вес: " + weight;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
        public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public double getWeight() {
        return weight;
    }
    public void setWeight(double weight) {
        this.weight = weight;
    }

    @Override
    public String toString() {
        return Info();
    }
}
```

package animals – помещает класс в пакет animals, что помогает структурировать код, abstract class Animal – объявление абстрактного класса, который не может быть создан напрямую, то есть нельзя написать new Animal().

Дале идёт объявление полей класса: имени, возраста и веса животных. Все животные имеют такие параметры. Поля закрыты (private), чтобы доступ к ним был только через методы – это и есть инкапсуляция.

Далее прописываются конструктор без параметров (по умолчанию), он вызывает другой конструктов через this(…), устанавливая значения по умолчанию, а также конструктов с параметрами, который позволяет задать характеристики конкретного животного при создании объекта.

Абстрактные методы makeSound() и move() не имеют реализации, каждое животное будет реализовывать их по-своему. Это пример полиморфизма.

Метод eat() одинаков для всех животных – реализует поведение. Не абстрактный, потому что одинаков для всех потомков.

Метод Info() выводит информацию об объекте, его можно будет использовать для демонстрации переопределения.

Геттеры и сеттеры – это пример для инкапсуляции, то есть доступ к приватным полям осуществляется через методы.

Метод toString() вызывается, когда объект нужно вывести как текст, без него запись выглядела бы как-то так animals.Cat@7a81197d.

Продолжим работу и создадим класс Mammal (млекопитающие) – это наследник абстрактного класса Animal, который добавляется специфические свойства и методы, характерные для млекопитающих.

```
package animals;

class Mammal extends Animal {
    private boolean hasFur;
    private String furColor;

    public Mammal() {
        super();
        this.hasFur = true;
        this.furColor = "неизвестный";
    }

    public Mammal (String name, int age, double weight, boolean hasFur, String furColor)
{
        super(name, age, weight);
        this.hasFur = hasFur;
        this.furColor = furColor;
    }

    public void feedMilk() {
        System.out.println(getName() + " кормит детёнышей молоком");
    }

    public void sleep() {
        System.out.println(getName() + " спит");
    }

    @Override
    public void makeSound() {
        System.out.println(getName() + " издает звук (млекопитающее)");
    }

    @Override
    public void move() {
        System.out.println(getName() + " ходит или бегает");
```

```
    }

    public boolean isHasFur() {
        return hasFur;
    }
    public void setHasFur(boolean hasFur) {
        this.hasFur = hasFur;
    }
    public String getFurColor() {
        return furColor;
    }
    public void setFurColor(String furColor) {
        this.furColor = furColor;
    }
}
```

С помощью ключевого слова extends класс Mammal наследует все поля и методы родителя.

Поля hasFur и furCol закрыты, то есть приватные (private), поэтому к ним нельзя обратиться напрямую из другого класса. Доступ к ним идет через геттеры и сеттеры.

Создаем конструктор по умолчанию, super() вызывает конструктор родителя, чтобы инициализировать name, age и weight значениями по умолчанию, затем устанавливаем значения по умолчанию для оставшихся полей. Затем идёт конструктор с параметрами, тут super() вызывает конструктор родителя с параметрами, потом устанавливаем поля для конкретного объекта.

Методы feedMilk() и sleep() уникальны для млекопитающих, это пример специфического поведения подкласса.

Далее идет переопределение абстрактных методов родителя makeSound() и move(). Ключевое слово @Override говорит компилятору, что метод замещает метод родителя. Можно определить специфическое поведение для млекопитающих.

В конце идут геттеры и сеттеры для работы с приватными полями hasFur() и furCol().

После создаём ещё два класса Bird и Aquatic, которые также являются наследниками абстрактного класса Animal.

```
package animals;

class Bird extends Animal {
    private boolean canFly;
    private double wingSpan;

    public Bird() {
        super();
        this.canFly = true;
        this.wingSpan = 0.0;
```

```
    }

    public Bird(String name, int age, double weight, boolean canFly, double wingSpan) {
        super(name, age, weight);
        this.canFly = canFly;
        this.wingSpan = wingSpan;
    }

    public void fly() {
        if (canFly) {
            System.out.println(getName() + " летит (размах крыла: " + wingSpan + " м)");
        } else {
            System.out.println(getName() + " не умеет летать");
        }
    }

    public void buildNest() {
        System.out.println(getName() + " строит гнездо");
    }

    @Override
    public void makeSound() {
        System.out.println(getName() + " издает звук (птица)");
    }
    @Override
    public void move() {
        System.out.println(getName() + " ходит или прыгает");
    }

    public boolean isCanFly() {
        return canFly;
    }
    public void setCanFly(boolean canFly) {
        this.canFly = canFly;
    }

    public double getWingSpan() {
        return wingSpan;
    }
    public void setWingSpan(double wingSpan) {
        this.wingSpan = wingSpan;
    }
}
```

```
package animals;

class Aquatic extends Animal {
    private String waterType;
    private int fins;
```

```
    public Aquatic() {
        super();
        this.waterType = "неизвестный";
        this.fins = 0;
    }

    public Aquatic (String name, int age, double weight, String waterType, int fins) {
        super(name, age, weight);
        this.waterType = waterType;
        this.fins = fins;
    }

     public void swim() {
        System.out.println(getName() + " плавает (" + waterType + ")");
    }

    public void blowBubbles() {
        System.out.println(getName() + " выпускает пузырьки");
    }

    @Override
    public void makeSound() {
        System.out.println(getName() + " издаёт тихие водные звуки");
    }

    @Override
    public void move() {
        swim();
    }

    public String getWaterType() {
        return waterType;
    }
    public void setWaterType(String waterType) {
        this.waterType = waterType;
    }

    public int getFins() {
        return fins;
    }
    public void setFins(int fins) {
        this.fins = fins;
    }
}
```

Далее создаём класс Cat, которые наследует от Mammal поля и методы и представляет собой конкретный вид млекопитающих – кошку. Он добавляет новые поля, методы и использует механизмы наследования, инкапсуляции и статических членов класса.

```
package animals;
```

```
class Cat extends Mammal {
    private String breed;
    private boolean isIndoor;
    private int clawSharpness;

    private static int catCount = 0;

    public Cat() {
        super();
        this.breed = "неизвестный";
        this.isIndoor = true;
        this.clawSharpness = 5;
        catCount++;
    }

    public Cat (String name, int age, double weight, boolean hasFur, String furColor,
String breed, boolean isIndoor, int clawSharpness) {
        super(name, age, weight, hasFur, furColor);
        this.breed = breed;
        this.isIndoor = isIndoor;
        this.clawSharpness = clawSharpness;
        catCount++;
    }

    public void purr() {
        System.out.println(getName() + " мурчит");
    }

    public void scratch() {
        System.out.println(getName() + " точит когти (острота:" + clawSharpness + ")");
    }

    @Override
    public void makeSound() {
        System.out.println(getName() + " говорит: Мяу!");
    }

    @Override
    public void move() {
        System.out.println(getName() + " крадётся и прыгает");
    }

    public static int getCatCount() {
        return catCount;
    }

    public String getBreed() {
        return breed;
    }
    public void setBreed(String breed) {
```

```
        this.breed = breed;
    }

    public boolean isIndoor() {
        return isIndoor;
    }
    public void setIndoor(boolean indoor) {
        isIndoor = indoor;
    }

    public int getClawSharpness() {
        return clawSharpness;
    }
    public void setClawSharpness(int clawSharpness) {
        this.clawSharpness = clawSharpness;
    }
}
```

private static int catCount – создаем статическое поле – общее количество объектов класса Cat. Оно принадлежит всему классу, а не отдельному объекту, и увеличивается при каждом создании кошки.

По аналогии создаем еще два конкретных класса Parrot и Fish для родительских классов Bird и Aquatic соответственно.

```
package animals;

class Parrot extends Bird {
    private int vocabularySize;
    private String color;
    private boolean canTalk;

    public Parrot() {
        super();
        this.vocabularySize = 0;
        this.color = "неизвестный";
        this.canTalk = false;
    }

    public Parrot(String name, int age, double weight, boolean canFly, double wingSpan,
int vocabularySize, String color, boolean canTalk) {
        super(name, age, weight, canFly, wingSpan);
        this.vocabularySize = vocabularySize;
        this.color = color;
        this.canTalk = canTalk;
    }

    public void mimic(String phrase) {
        System.out.println(getName() + " повторяет: \"" + phrase + "\"");
    }
```

```
    public void talk() {
        if (canTalk && vocabularySize > 0) {
            System.out.println(getName() + " говорит несколько слов (словарь: " +
vocabularySize + ")");
        } else {
            System.out.println(getName() + " молчит или издаёт крики");
        }
    }

    @Override
    public void makeSound() {
        System.out.println(getName() + " кричит или щебечет");
    }

    @Override
    public void move() {
        if (isCanFly()) {
            fly();
        } else {
            System.out.println(getName() + " прыгает по жердочке");
        }
    }

    public int getVocabularySize() {
        return vocabularySize;
    }
    public void setVocabularySize(int vocabularySize) {
        this.vocabularySize = vocabularySize;
    }

    public String getColor() {
        return color;
    }
    public void setColor(String color) {
        this.color = color;
    }

    public boolean isCanTalk() {
        return canTalk;
    }
    public void setCanTalk(boolean canTalk) {
        this.canTalk = canTalk;
    }
}
```

```
package animals;

class Fish extends Aquatic {
    private String species;
```

```
    private String scaleColor;
    private double tankVolume;

    public Fish() {
        super();
        this.species = "неизвестный";
        this.scaleColor = "неизвестный";
        this.tankVolume = 0.0;
    }

    public Fish(String name, int age, double weight, String waterType, int fins, String
species, String scaleColor, double tankVolume) {
        super(name, age, weight, waterType, fins);
        this.species = species;
        this.scaleColor = scaleColor;
        this.tankVolume = tankVolume;
    }

    public void blowBubbles() {
        System.out.println(getName() + " выпускает пузырьки");
    }

    public void swimFaster() {
        System.out.println(getName() + " ускоряется и быстро плывёт");
    }

    @Override
    public void makeSound() {
        System.out.println(getName() + " тихо \"булькает\"");
    }

    @Override
    public void move() {
        swim();
    }

    public String getSpecies() {
        return species;
    }
    public void setSpecies(String species) {
        this.species = species;
    }

    public String getScaleColor() {
        return scaleColor;
    }
    public void setScaleColor(String scaleColor) {
        this.scaleColor = scaleColor;
    }

    public double getTankVolume() {
```

```
        return tankVolume;
    }
    public void setTankVolume(double tankVolume) {
        this.tankVolume = tankVolume;
    }
}
```

Продемонстрируем работу всех созданных классов (Animal, Mammal, Bird, Aquatic, Cat, Parrot, Fish) и применение принципов ООП на практике.

```
package animals;
import java.util.Scanner;

public class LabAnimals {
    public static void main(String[] args) {
        Cat catOne = new Cat("Кузьма", 3, 4.5, true, "серый", "Британец", true, 7);
        Cat catTwo = new Cat();
        Parrot parrot = new Parrot("Кеша", 2, 0.4, true, 0.25, 100, "зелёный", true);
        Fish fish = new Fish("Немо", 1, 0.1, "пресная", 2, "рыба-клоун", "оранжевый",
10.0);

        System.out.println("Счётчик кошек (после создания cat1, cat2): " +
Cat.getCatCount());
```

Класс LabAnimals содержит метод main() – точку входа программы, где создаются объекты различных животных, демонстрируется полиморфизм, работа со статическими полями и ввод данных от пользователя.

Создется четыре объекта разных подклассов. Демонстрируется наследование и разнообразие поведения объектов разных классов. Также при каждом создании нового объекта класса Cat значение catCount увеличивается, затем это значение выводится.

```
Animal[] animals = new Animal[]{catOne, parrot, fish, catTwo};
        System.out.println("\nДемонстрация полиморфизма и общих действий");
        for (Animal a : animals) {
            System.out.println("\n" + a);
            a.makeSound();
            a.move();
            a.eat("корм");
        }
```

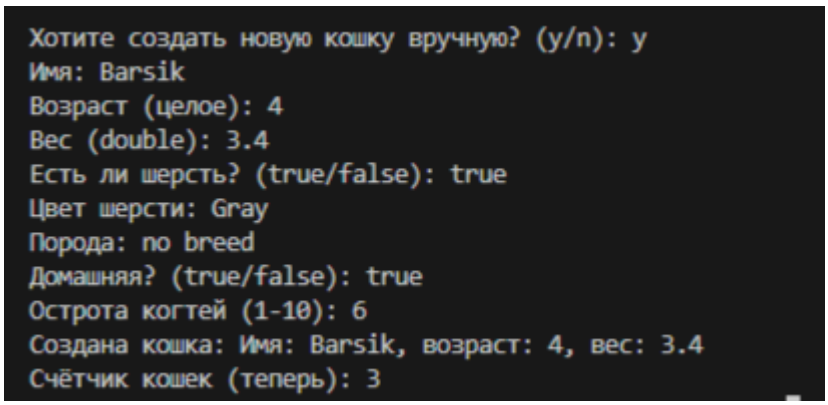
Создаем массив ссылок на базовый класс Animal, куда помещаются разнообразные объекты. Это позволит вызвать одни и те же методы (makeSound(), move(), eat()) для всех объектов, но при этом выполняются разные реализации, соответствующие каждому подклассу. Так демонстрируется полиморфизм – способность объектов разных типов по-разному реагировать на одинаковые вызовы методов.

Ниже показан пример вывода:

```
Демонстрация полиморфизма и общих действий

Имя: Кузьма, возраст: 3, вес: 4.5
Кузьма говорит: Мяу!
Кузьма крадётся и прыгает
Кузьма ест корм

Имя: Кеша, возраст: 2, вес: 0.4
Кеша кричит или щебечет
Кеша летит (размах крыла: 0.25 м)
Кеша ест корм

Имя: Немо, возраст: 1, вес: 0.1
Немо тихо "булькает"
Немо плавает (пресная)
Немо ест корм

Имя: Безымянный, возраст: 0, вес: 0.0
Безымянный говорит: Мяу!
Безымянный крадётся и прыгает
Безымянный ест корм
```

```
Scanner scanner = new Scanner(System.in);
        System.out.print("\nХотите создать новую кошку вручную? (y/n): ");
        String answer = scanner.nextLine().trim();
        if (answer.equalsIgnoreCase("y")) {
            System.out.print("Имя: ");
            String name = scanner.nextLine();
            System.out.print("Возраст (целое): ");
            int age = Integer.parseInt(scanner.nextLine());
            System.out.print("Вес (double): ");
            double weight = Double.parseDouble(scanner.nextLine());
            System.out.print("Есть ли шерсть? (true/false): ");
            boolean hasFur = Boolean.parseBoolean(scanner.nextLine());
            System.out.print("Цвет шерсти: ");
            String furColor = scanner.nextLine();
            System.out.print("Порода: ");
            String breed = scanner.nextLine();
            System.out.print("Домашняя? (true/false): ");
            boolean isIndoor = Boolean.parseBoolean(scanner.nextLine());
            System.out.print("Острота когтей (1-10): ");
            int sharpness = Integer.parseInt(scanner.nextLine());

            Cat userCat = new Cat(name, age, weight, hasFur, furColor, breed, isIndoor,
sharpness);
            System.out.println("Создана кошка: " + userCat);
            System.out.println("Счётчик кошек (теперь): " + Cat.getCatCount());
        }

        scanner.close();
    }
}
```

С помощью класса Scanner программа спрашивает пользователя, хочет ли он создать нового питомца. Если ответ y, то далее по шагам запрашиваются все характеристики кошки. После ввода создается новый объект с этими параметрами.



Вывод

В ходе лабораторной работы была реализована иерархия классов животных с использованием принципов объектно-ориентированного программирования на языке Java.

Был создан абстрактный класс Animal, от которого наследовали поля и методы классы Mammal, Bird, Fish. От этих классов уже были написаны конкретные классы Cat, Parrot, Fish, демонстрирующие специфические свойства и методы животных. Все это пример **наследования**.

Методы makeSound() и move() переопределены в подклассах, что позволило вызвать одни и те же методы для разных объектов и получить разное поведение. Пример **полиморфизма**.

Все поля классов были сделаны приватными, доступ к ним осуществляется через геттеры и сеттеры, что обеспечивает контроль над изменениями состояния объектов. Это пример **инкапсуляции**.

Класс Animal описывает общие характеристики и поведение всех животных, но сам по себе не создаёт конкретных объектов, потому что это абстрактное понятие животного. Это явный пример **абстракции**.

Контрольные вопросы

1. Абстракция – это выделение наиболее значимых характеристик объекта, скрывая детали его реализации. В Java реализуется с помощью абстрактных классов или интерфейсов.
2. Инкапсуляция – это сокрытие внутренней реализации объекта от внешнего мира и предоставление доступа к данным только через определенные методы класса. Реализация в Java основана на создании privet (приватных) полей класса и геттерах/сеттерах для доступа к ним.
3. Наследование – это создание новых классов на основе существующих, новые классы перенимают свойства и методы родительских классов. В Java реализуется с помощью ключевого слова extends.
4. Полиморфизм – это принцип ООП, который позволяет объектам разных классов реагировать по-разному на один и тот же вызов метода. В Java

реализуется через переопределение методов (@Override) и через работу с объектами через ссылки базового класса.

5. Множественное наследование – это возможность класса наследоваться от более чем одного класса одновременно. В Java множественного наследования нет, но его можно реализовать через интерфейсы.
6. Ключевое слово final используется для ограничения изменений, оно защищает данные, методы или классы от нежелательных изменений.
7. Модификаторы доступа определяют, кто может видеть и использовать поля, методы и классы.
   public – везде, без ограничений
   protected – в пределах пакетов + в подклассах
   default – в пределах пакета (package-private)
   private – только внутри текущего класса
8. Конструктор – это специальный метод класса, который вызывается при создании объекта для инициализации его полей. Есть конструктор по умолчанию и параметрических конструктор.
9. this – это ссылка на текущий объект класса, в котором выполняется код. Используется для различия полей класса и параметров метода/конструктора, когда они имеют одинаковые имена. Также позволяет вызывать другие методы текущего объекта.
10. super – это ссылка на родительский класс текущего объекта. Позволяет вызвать конструктор родителя и вызвать метод родителя, если он переопределен в подклассе.
11. Геттеры и сеттеры – это методы для доступа и изменения закрытых полей класса. Геттер возвращает значение поля, сеттер – изменяет значение поля с контролем и проверками.
12. Переопределение – это создание в подклассе новой реализации метода родителя с тем же именем, типом возвращаемого значения и параметрами. Позволяет объектам разных классов по-разному реагировать на один и тот же вызов метода.
13. Перегрузка – это создание нескольких методов с одинаковым именем, но с разными параметрами внутри одного класса. Позволяет использовать один метод для разных типов и количества данных.