

**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 Математическая кибернетика и информационные технологии

Отчет по лабораторной работе №2
по дисциплине «Информационные технологии и программирование на языке
Java»

Выполнил: студент группы БВТ2403

Подлуцкий Никита Сергеевич

Руководитель: _____

Москва, 2025

2. Цель работы

Цель работы — освоение и практическое применение основных принципов объектно-ориентированного программирования (ООП) на языке Java. В ходе работы необходимо было спроектировать и реализовать иерархию классов, используя такие концепции, как абстракция, инкапсуляция, наследование и полиморфизм, а также научиться применять конструкторы, геттеры и сеттеры для управления состоянием объектов.

3. Индивидуальное задание

(Здесь просто перечисляем требования из учебника)

Создать иерархию классов в соответствии с вариантом №1 (Базовый класс: Животные. Дочерние классы: Кошка, Попугай, Рыбка). Иерархия должна содержать:

- абстрактный класс;
- два уровня наследуемых классов (минимум 3 поля и 2 метода);
- демонстрацию реализации всех принципов ООП;
- наличие конструкторов (в том числе по умолчанию);
- наличие геттеров и сеттеров;
- ввод/вывод информации о создаваемых объектах;
- реализацию счетчика созданных объектов с использованием статической переменной.

4. Основная часть

Animal — абстрактный базовый класс, который описывает общие для всех животных поля (name, age, weight) и методы (eat, showInfo). Для реализации инкапсуляции поля объявлены как `private`, а для доступа к ним созданы публичные геттеры и сеттеры. Метод `makeSound()` объявлен как абстрактный, что обязывает все дочерние классы предоставить его реализацию. `@Override` я использовал для переопределения функций для того чтобы компилятор нормально реагировал на ошибки. В классе также реализован статический счетчик `animalCount` для подсчета созданных объектов.

Cat, Parrot и Fish — это конкретные классы, которые наследуются от **Animal** с помощью ключевого слова `extends`. Каждый из этих классов предоставляет собственную реализацию (переопределение) метода `makeSound()`, что демонстрирует принцип полиморфизма. В каждом классе реализовано два конструктора: один с параметрами для создания полноценного объекта и один по умолчанию.

```

package com.example;

// Класс для кошки, наследуется от Animal
public class Cat extends Animal {

    // Конструктор кошки
    public Cat(String name, int age, double weight) {
        // super() - это вызов конструктора родителя (Animal)
        // чтобы передать ему имя, возраст и вес
        super(name, age, weight);
    }

    public Cat() {
        super(name: "Безымянный кот", age: 1, weight: 4.0);
    }

    // @Override - значит мы переписываем метод из Animal
    // мы обязаны это сделать, потому что makeSound абстрактный. Но без него тоже можно хотя плохо для дебагинга
    @Override
    public void makeSound() {
        System.out.println(x: "Мяу!");
    }
}

```

```

package com.example;

// Класс для рыбы
public class Fish extends Animal {

    public Fish(String name, int age, double weight) {
        super(name, age, weight);
    }

    public Fish() {
        super(name: "Безымянный рыба", age: 1, weight: 4.0);
    }

    // рыбы молчат, но метод все равно должен быть
    @Override
    public void makeSound() {
        System.out.println(x: "... (бульк)");
    }
}

```

```

5 public abstract class Animal {
6     // Конструктор - вызывается когда пишем new Animal()
7     public Animal(String name, int age, double weight) {
8         this.name = name;
9         this.age = age;
10        this.weight = weight;
11        animalCount++;
12    }
13
14
15
16    // этот метод одинаковый для всех, поэтому пишем его здесь
17    public void eat() {
18        System.out.println(this.name + " кушает.");
19    }
20
21    // тоже общий метод чтобы показать инфу
22    public void showInfo() {
23        System.out.println("Имя: " + this.name + ", Возраст: " + this.age + " лет, Вес: " + this.weight + " кг");
24    }
25
26
27    // этот метод у всех разный, поэтому тут его не пишем
28    // но заставляем всех наследников его написать у себя
29    public abstract void makeSound();
30
31
32
33    public String getName() { return this.name; }
34    public int getAge() { return this.age; }
35    public double getWeight() { return this.weight; }
36
37
38    public void setName(String name) { this.name = name; }
39    public void setAge(int age) { this.age = age; }
40    public void setWeight(double weight) { this.weight = weight; }
41
42
43    public static int getAnimalCount() {
44        return animalCount;
45    }
46 }

```

```

1 package com.example;
2
3 // Класс для попугая, тоже наследуется от Animal
4 public class Parrot extends Animal {
5
6     // Конструктор попугая
7     public Parrot(String name, int age, double weight) {
8         // опять вызываем конструктор Animal
9         super(name, age, weight);
10    }
11    public Parrot(){
12        super(name:"Безымянный попуг", age:1, weight:4.0);
13    }
14    // даем попугаю его собственный звук
15    @Override
16    public void makeSound() {
17        System.out.println(x:"Чирик!");
18    }
19 }

```

Main — основной класс, содержащий точку входа в программу. В методе `main` создаются экземпляры дочерних классов, демонстрируется работа их методов, а также выводится итоговое значение статического счетчика, чтобы показать его работоспособность.

```
package com.example;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        System.out.println(x:"--- Создаем наших животных ---");

        // Cat - это тун, cat - это имя переменной, а new Cat(...) - создает сам объект
        Cat cat = new Cat(name:"Барсик", age:3, weight:5.5);

        Parrot parrot = new Parrot(name:"Кеша", age:2, weight:0.8);

        Fish fish = new Fish(name:"Немо", age:100, weight:0.1);

        System.out.println(x:"\n--- Проверяем что они умеют делать ---");

        // заставляем кота делать что-то
        System.out.println(x:"Действия кота:");
        cat.eat();
        cat.makeSound();

        // инфо про попугая
        System.out.println(x:"\nИнформация о попугае:");
        parrot.showInfo();

        parrot.makeSound();

        System.out.println(x:"\nИнформация о рыбе:");
        fish.showInfo();

        System.out.println(x:"\n--- Проверяем общий счетчик ---");

        System.out.println("Всего животных в программе: " + Animal.getAnimalCount());
    }
}
```

5. Заключение

В ходе выполнения данной лабораторной работы я успешно применил на практике ключевые принципы объектно-ориентированного программирования. Я научился создавать иерархии классов с использованием абстракции и наследования, реализовывать инкапсуляцию с помощью частных полей, геттеров и сеттеров, а также понял, как работает полиморфизм на примере переопределения методов. Полученные навыки являются важной основой для дальнейшей разработки на языке Java.

GitHub: <https://github.com/Nikita-Podlutsky/ITIP/tree/main>

Контрольные вопросы

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

Абстракция — это выделение самых главных характеристик объекта и игнорирование второстепенных. В Java она реализуется с помощью абстрактных классов и абстрактных методов. В нашем коде `abstract class Animal` — это абстракция, потому что она описывает общие свойства всех животных, а `abstract void makeSound()` — это абстрактный метод, который говорит, что все животные издают звук, но не уточняет, какой именно.

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

Инкапсуляция — это сокрытие данных объекта от прямого доступа и предоставление специальных методов для работы с ними. Она реализуется с помощью модификатора доступа `private` для полей. В нашем коде поля `name`, `age` и `weight` в классе `Animal` являются приватными. Доступ к ним осуществляется только через публичные методы — геттеры и сеттеры.

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

Наследование — это механизм, который позволяет одному классу перенимать свойства и методы другого класса. Оно реализуется с помощью ключевого слова `extends`. В нашем коде `public class Cat extends Animal` означает, что класс `Cat` наследует все публичные поля и методы от класса `Animal`, например, метод `eat()`.

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

Полиморфизм — это способность объектов с одним и тем же интерфейсом (или родительским классом) иметь разную реализацию методов. В `main` мы можем написать `Animal animal = new Cat();`, и вызов `animal.makeSound()` напечатает "Мяу!", а если написать `Animal animal = new Fish();`, то тот же самый вызов `animal.makeSound()` напечатает "... (бульк)". Одна и та же команда приводит к разному поведению.

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

Множественное наследование — это возможность класса наследовать свойства сразу от нескольких родительских классов. В Java нет множественного наследования классов.

6. Для чего нужно ключевое слово `final`?

Ключевое слово `final` делает что-либо неизменяемым. `final`-переменную нельзя переопределить, `final`-метод нельзя переопределить в классе-наследнике, а от `final`-класса нельзя наследоваться.

7. Какие в Java есть модификаторы доступа?

В Java есть четыре модификатора доступа: `public` (виден всем), `protected` (виден внутри пакета и наследникам), `default` (без ключевого слова, виден только внутри пакета) и `private` (виден только внутри своего класса).

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

Конструктор — это специальный метод, который вызывается при создании нового объекта (`new`). Его задача — инициализировать поля объекта. Бывают конструкторы с

параметрами (как наш `public Cat(String name, ...)`), и конструкторы по умолчанию (без параметров, как наш `public Cat()`).

9. Для чего нужно ключевое слово `this`?

Ключевое слово `this` — это ссылка на текущий экземпляр объекта. Оно используется, чтобы отличить поле класса от параметра метода, если у них одинаковые имена. В нашем коде в конструкторе `Animal` мы пишем `this.name = name`;, чтобы присвоить значение параметра `name` полю `name` нашего объекта. Аналогична `self` в `python`.

10. Для чего нужно ключевое слово `super`?

Ключевое слово `super` используется для обращения к родительскому классу. В нашем коде мы используем `super(name, age, weight)`; в конструкторе `Cat`, чтобы вызвать конструктор родительского класса `Animal` и передать ему нужные параметры.

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

Геттеры — это методы для получения (чтения) значения приватного поля (например, `getName()`). Сеттеры — это методы для установки (изменения) значения приватного поля (например, `setName()`). Они нужны для реализации принципа инкапсуляции, чтобы контролировать доступ к данным.

12. Что такое переопределение?

Переопределение (`Override`) — это когда класс-наследник предоставляет свою собственную реализацию метода, который уже есть у родительского класса. В нашем коде класс `Cat` переопределяет абстрактный метод `makeSound()` из класса `Animal`.

13. Что такое перегрузка?

Перегрузка (`Overload`) — это создание в одном классе нескольких методов с одинаковым именем, но с разными параметрами (разное количество или типы). Например, в классе `Cat` мы могли бы иметь два конструктора: `Cat(String name)` и `Cat(String name, int age)`.