**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

# Лабораторная работа №2. Вариант 11

Выполнил: студент группы БПИ2401

Исламов Эмин Маратович

Проверил: Харрасов Камиль Раисович

Москва, 2025

## Цель работы

Закрепить основные принципы объектно-ориентированного программирования на языке Java (инкапсуляция, наследование, полиморфизм, абстракция) путём создания иерархии классов на примере предметной области «Бытовая техника».

**Тема иерархии:** Бытовая техника — Холодильник, Посудомоечная машина, Пылесос

## Краткая теория

Объектно-ориентированное программирование (ООП) — это подход, при котором программа строится как совокупность объектов, взаимодействующих между собой.

Основные принципы:

- Инкапсуляция — защита данных от прямого доступа.
- Наследование — создание новых классов на основе существующих.
- Полиморфизм — способность методов вести себя по-разному в зависимости от объекта.
- Абстракция — выделение главных свойств объекта и скрытие лишних деталей.

## Ход работы

1. **Абстрактный базовый класс**
   Данный класс является основой всей иерархии бытовой техники.
   Он описывает общие характеристики для всех видов приборов: марку, мощность и цену.
   В нём реализованы принципы инкапсуляции (поля закрыты, используется доступ через геттеры и сеттеры) и абстракции — методы turnOn() и turnOff() объявлены абстрактными, так как их реализация зависит от конкретного устройства.

Также в классе присутствует статическая переменная count, которая выступает в роли счётчика созданных объектов, демонстрируя применение статических членов класса.

```
public abstract class Appliance {
    private String brand;
    private double power;
    private double price;
    private static int count = 0;

    public Appliance(String brand, double power, double price) {
        this.brand = brand;
        this.power = power;
        this.price = price;
        count++;
    }

    public Appliance() {
        this("Неизвестно", 0, 0);
    }

    public abstract void turnOn();
    public abstract void turnOff();

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public double getPower() {
        return power;
    }

    public void setPower(double power) {
        this.power = power;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public static int getCount() {
        return count;
    }
}
```

2. **Класс “Холодильник”**

   Класс Refrigerator наследуется от базового Appliance и реализует конкретное поведение метода включения и выключения холодильника.

   Добавлено дополнительное поле temperature, отражающее температуру внутри камеры.

   В классе реализованы **конструкторы по умолчанию и с параметрами**, а также методы getTemperature() и setTemperature() для работы с температурой.

   Здесь проявляется принцип **наследования** (унаследованы общие свойства) и **полиморфизма** — переопределены абстрактные методы базового класса.

```
public class Refrigerator extends Appliance {
    private double temperature;

    public Refrigerator(String brand, double power, double price,
double temperature) {
        super(brand, power, price);
        this.temperature = temperature;
    }

    public Refrigerator() {
        super();
        this.temperature = 4;
    }

    @Override
    public void turnOn() {
        System.out.println("Холодильник включен. Температура: " +
temperature + "°C");
    }

    @Override
    public void turnOff() {
        System.out.println("Холодильник выключен.");
    }

    public double getTemperature() {
        return temperature;
    }

    public void setTemperature(double temperature) {
        this.temperature = temperature;
    }
}
```

3. **Класс “Посудомоечная машина”**

Класс Dishwasher расширяет Appliance и моделирует поведение посудомоечной машины.

Он содержит собственное поле capacity, указывающее количество комплектов посуды, которые можно загрузить за один цикл.

Методы turnOn() и turnOff() переопределены, чтобы отразить индивидуальное поведение данного типа техники.

Таким образом, реализован динамический полиморфизм, где одна и та же операция (turnOn) выполняется по-разному для разных объектов.

```
public class Dishwasher extends Appliance {
    private int capacity;

    public Dishwasher(String brand, double power, double price, int
capacity) {
        super(brand, power, price);
        this.capacity = capacity;
    }

    public Dishwasher() {
        super();
        this.capacity = 10;
    }

    @Override
    public void turnOn() {
        System.out.println("Посудомоечная машина запущена.
Вместимость: " + capacity + " комплектов.");
    }

    @Override
    public void turnOff() {
        System.out.println("Посудомоечная машина завершила
работу.");
    }

    public int getCapacity() {
        return capacity;
    }

    public void setCapacity(int capacity) {
        this.capacity = capacity;
    }
}
```

4. **Класс “Пылесос”**

   **Класс VacuumCleaner также наследует от Appliance и описывает функциональность пылесоса.**

**Добавлено поле type, которое хранит тип устройства (например, «беспроводной», «вертикальный» и т. д.).**
**Реализованы конструкторы, а также методы доступа к полю.**
**Класс демонстрирует переопределение методов и расширение базового функционала.**
**В нём также применяется инкапсуляция и переопределение методов (@Override).**

```
public class VacuumCleaner extends Appliance {
    private String type;

    public VacuumCleaner(String brand, double power, double price,
String type) {
        super(brand, power, price);
        this.type = type;
    }

    public VacuumCleaner() {
        super();
        this.type = "Вертикальный";
    }

    @Override
    public void turnOn() {
        System.out.println("Пылесос включен. Тип: " + type);
    }

    @Override
    public void turnOff() {
        System.out.println("Пылесос выключен.");
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }
}
```

5. **Класс с демонстрацией**

Класс Main служит для демонстрации работы созданной иерархии.
В методе main() создаются объекты различных подклассов (Refrigerator, Dishwasher, VacuumCleaner), вызываются их методы и выводится количество созданных объектов.

Этот класс иллюстрирует работу полиморфизма, поскольку методы turnOn() вызываются для разных типов бытовой техники, но выполняются по-разному.
Таким образом, Main объединяет все принципы ООП, реализованные в предыдущих классах, и демонстрирует их взаимодействие на практике.

```
public class Main {
    public static void main(String[] args) {
        Refrigerator fridge = new Refrigerator("LG", 250, 45000,
3.5);
        Dishwasher washer = new Dishwasher("Bosch", 1800, 60000,
12);
        VacuumCleaner vacuum = new VacuumCleaner("Dyson", 900,
70000, "Беспроводной");

        fridge.turnOn();
        washer.turnOn();
        vacuum.turnOn();

        System.out.println("Всего создано объектов: " +
Appliance.getCount());
    }
}
```

```
/Users/emin/Library/Java/JavaVirtualMachines/openjdk-22.0.1/Con
Холодильник включен. Температура: 3.5°C
Посудомоечная машина запущена. Вместимость: 12 комплектов.
Пылесос включен. Тип: Беспроводной
Всего создано объектов: 3

Process finished with exit code 0
```

**Вывод**

В ходе лабораторной работы была создана иерархия классов с базовым абстрактным классом Appliance и тремя дочерними классами: Refrigerator, Dishwasher, VacuumCleaner.

Реализованы все основные принципы ООП:

- Инкапсуляция — поля скрыты, доступ через геттеры/сеттеры;
- Наследование — дочерние классы наследуют поведение базового;
- Полиморфизм — методы turnOn() и turnOff() переопределены;
- Абстракция — общий интерфейс устройства задан через абстрактный класс.

Программа корректно демонстрирует работу с объектами разных типов и подсчёт количества созданных экземпляров.

**Контрольные вопросы**

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

Абстракция — это выделение существенных свойств объекта и скрытие несущественных деталей реализации.

В Java абстракция реализуется через абстрактные классы (abstract class) и интерфейсы (interface), которые задают общий шаблон поведения, а конкретные **классы реализуют их методы.**

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

Инкапсуляция — это сокрытие внутреннего состояния объекта и предоставление доступа к нему только через методы.

В Java она реализуется с помощью модификаторов доступа (private, public, protected) и геттеров/сеттеров для работы с приватными полями.

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

Наследование — это механизм, который позволяет создавать новые классы на основе уже существующих, наследуя их поля и методы.

В Java наследование реализуется с помощью ключевого слова extends для классов и implements для интерфейсов.

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

Полиморфизм — это способность объектов разных классов реагировать по-разному на одинаковые вызовы методов.

Реализуется двумя способами:

- Переопределение методов (динамический полиморфизм);
- Перегрузка методов (статический полиморфизм).

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

Множественное наследование — это возможность наследовать функциональность сразу от нескольких классов.

В Java множественное наследование классов запрещено, чтобы избежать конфликтов (“ромбовидной проблемы”).

Вместо этого используется множественная реализация интерфейсов.

**6. Для чего нужно ключевое слово final?**

Ключевое слово final используется для:

- переменных — запрет изменения значения;
- методов — запрет переопределения в подклассах;
- классов — запрет наследования.

**7. Какие в Java есть модификаторы доступа?**

1. public — доступен везде;

2. protected — доступен в пределах пакета и наследникам;
3. *без модификатора* (package-private) — доступен только в пакете;
4. private — доступен только внутри класса.

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

Конструктор — это специальный метод, который вызывается при создании объекта для инициализации его полей.

Бывают:

- по умолчанию (без параметров);
- с параметрами (для задания значений полей при создании).

**9. Для чего нужно ключевое слово this в Java?**

this указывает на текущий объект класса.

Оно используется для обращения к полям и методам объекта, а также для вызова другого конструктора внутри класса.

**10. Для чего нужно ключевое слово super в Java?**

super используется для обращения к элементам суперкласса:

- для вызова конструктора родителя;
- для обращения к переопределённым методам или полям.

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

Геттеры — методы для получения значений приватных полей.

Сеттеры — методы для их изменения.

Они обеспечивают контролируемый доступ к данным и реализуют инкапсуляцию.

**12. Что такое переопределение?**

Переопределение (override) — это изменение реализации метода родительского класса в дочернем классе при сохранении сигнатуры.

В Java используется аннотация @Override.

**13. Что такое перегрузка?**

Перегрузка (overload) — это создание нескольких методов с одинаковым именем, но разными параметрами (типом, количеством или порядком).

Применяется для удобства вызова методов с разными типами данных.