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»

Выполнил: студент группы

БПИ2401

Старков Дмитрий Константинович

Проверил:

Харрасов Камиль Раисович

Москва, 2025

Содержание

Цель работы

Освоить принципы объектно-ориентированного программирования (ООП).

Ход работы

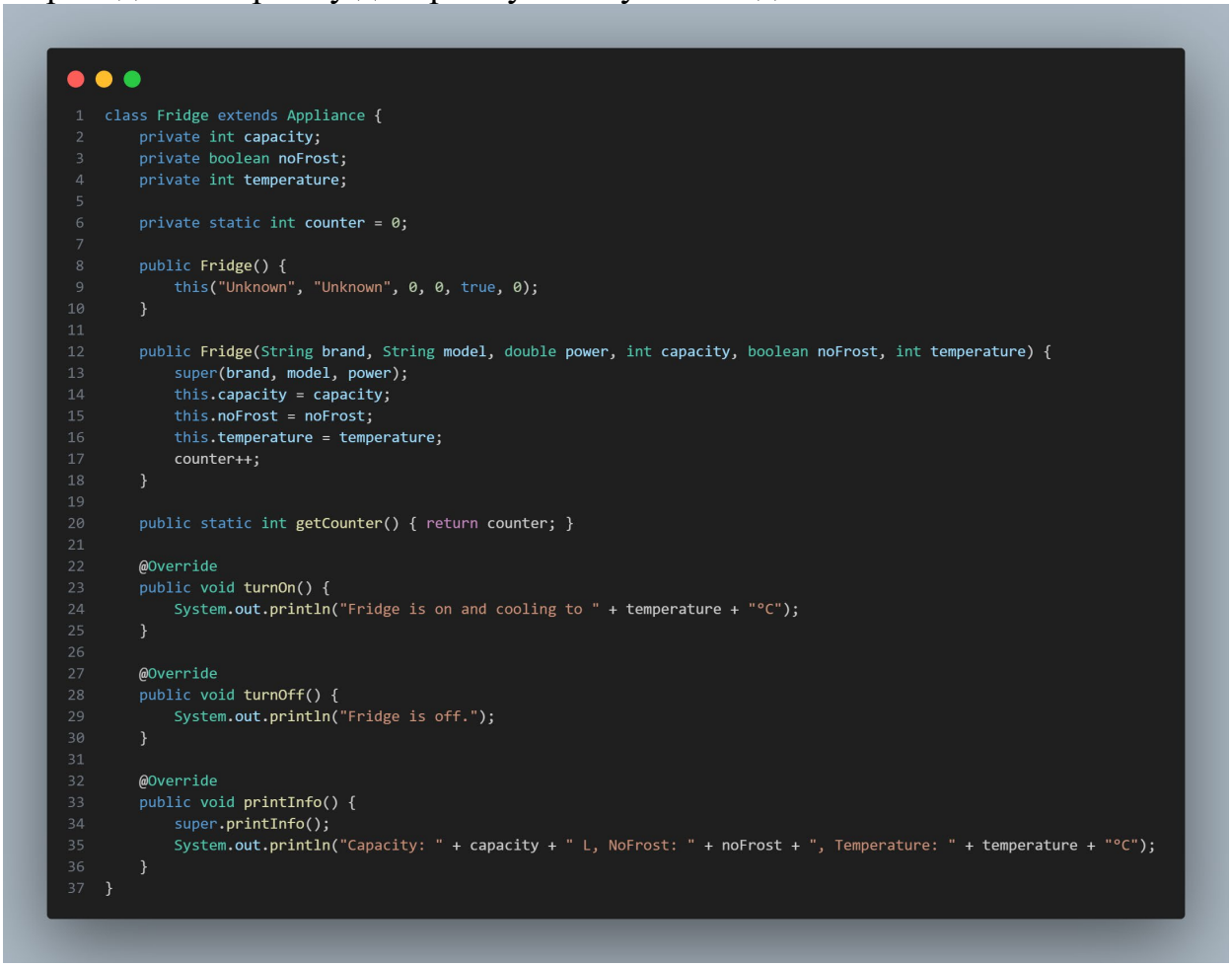
Начнем выполнение данной лабораторной с создания общего родительского абстрактного класса Appliance, вот как будет выглядеть код:

```java
abstract class Appliance {
    protected String brand;
    protected String model;
    protected double power;

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

    public Appliance(String brand, String model, double power) {
        this.brand = brand;
        this.model = model;
        this.power = power;
    }

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

    public String getModel() { return model; }
    public void setModel(String model) { this.model = model; }

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

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

    public void printInfo() {
        System.out.println("Appliance: " + brand + " " + model + ", power: " + power + " W");
    }
}

class Fridge extends Appliance {
    private int capacity;
    private boolean noFrost;
    private int temperature;

    private static int counter = 0;

```

В начале инициализирую 3 поля, которые будут у всех подклассов – бренд, модель и мощность, эти 3 поля я делаю типа Protected, чтобы дочерние классы имели к ним прямой доступ, это конечно нарушает строгую

концепцию инкапсуляции, но зато мне будет удобнее работать с ними в дочерних классах.
Далее идет конструктор, который вызывается при отсутствии аргументов в объявлении нового объекта данного класса. В этом конструкторе через оператор this я устанавливаю занчения Unknown и 0.
Джава понимает, какое значение относится к каждому атрибуту, потому что ниже у меня еще один конструктор, в котором переменные совпадают с типом значений.
Далее у меня идёт объявление сеттеров и геттеров, а также абстрактных классов turnOn и turnOff, которые в последствии будут перезаписаны для каждого дочернего класса. Ну и в конце общий метод printInfo, который будет выводить общую информацию для бытовой техники.

Переходим к первому дочернему классу – Холодильник:



```java
class Fridge extends Appliance {
    private int capacity;
    private boolean noFrost;
    private int temperature;

    private static int counter = 0;

    public Fridge() {
        this("Unknown", "Unknown", 0, 0, true, 0);
    }

    public Fridge(String brand, String model, double power, int capacity, boolean noFrost, int temperature) {
        super(brand, model, power);
        this.capacity = capacity;
        this.noFrost = noFrost;
        this.temperature = temperature;
        counter++;
    }

    public static int getCounter() { return counter; }

    @Override
    public void turnOn() {
        System.out.println("Fridge is on and cooling to " + temperature + "°C");
    }

    @Override
    public void turnOff() {
        System.out.println("Fridge is off.");
    }

    @Override
    public void printInfo() {
        super.printInfo();
        System.out.println("Capacity: " + capacity + " L, NoFrost: " + noFrost + ", Temperature: " + temperature + "°C");
    }
}
```

Здесь я инициализирую уже 3 приватных поля с объемом холодильника, наличием/отсутствием морозильной камеры и его температурой. Далее идёт объявление счетчика, который будет отображать количество объектов данного класса.
После чего идут конструкторы. Первый из них создан для пустого вызова, и несет в себе значения по умолчанию. Второй конструктор уже для общего случая. Первой строкой идёт ссылка на конструктор для родительского

класса с помощью оператора super, а далее уже приствaиваются значения для уникальных атрибутов данного класса.
Отмечу, что оператор this используется из-за одинакового названия атрибутов класса и переменных, переданных в конструктор. Это нужно для того, чтобы не происходило двусмысленности, из-за которой операция x=x расценивалась бы как присвание переменной x собственного значения. Оператор this позволяет точно отделить поле класса и переменную, которую нужно туда записать.
В конце конструктора идет инкрементация счетчика, означающая создание очередного объекта данного класса.

Далее идет определение метода getCounter(), чтобы получать значение счетчика. Ну и последнее – перезапись родительских методов с предварительной аннотацие @Override, которая однозначно определяет перезапись родительского метода, и в случае, если именя не совпадают – выдастся ошибка, которая предотвратит создание нежелательного метода у класса. Отмечу также использование super.printInfo() в последнем методе. Данную конструкцию я буду использовать во всех оставшихся классах, это позволит сначала вывести общую информацию о приборе, а потом уже более специфическую.

Следующуй класс:

```java
class Dishwasher extends Appliance {
    private int programs;
    private boolean drying;
    private int waterConsumption;

    public Dishwasher() {
        this("Unknown", "Unknown", 0, 0, true, 0);
    }

    public Dishwasher(String brand, String model, double power, int programs, boolean drying, int waterConsumption) {
        super(brand, model, power);
        this.programs = programs;
        this.drying = drying;
        this.waterConsumption = waterConsumption;
    }

    @Override
    public void turnOn() {
        System.out.println("Dishwasher started. Programs: " + programs);
    }

    @Override
    public void turnOff() {
        System.out.println("Dishwasher is off.");
    }

    @Override
    public void printInfo() {
        super.printInfo();
        System.out.println("Programs: " + programs + ", Drying: " + drying + ", Water consumption: " + waterConsumption + " L");
    }
}
```

Здесь особо объяснять нечего, ситуация аналогична с предыдущим классом. Идёи далее:

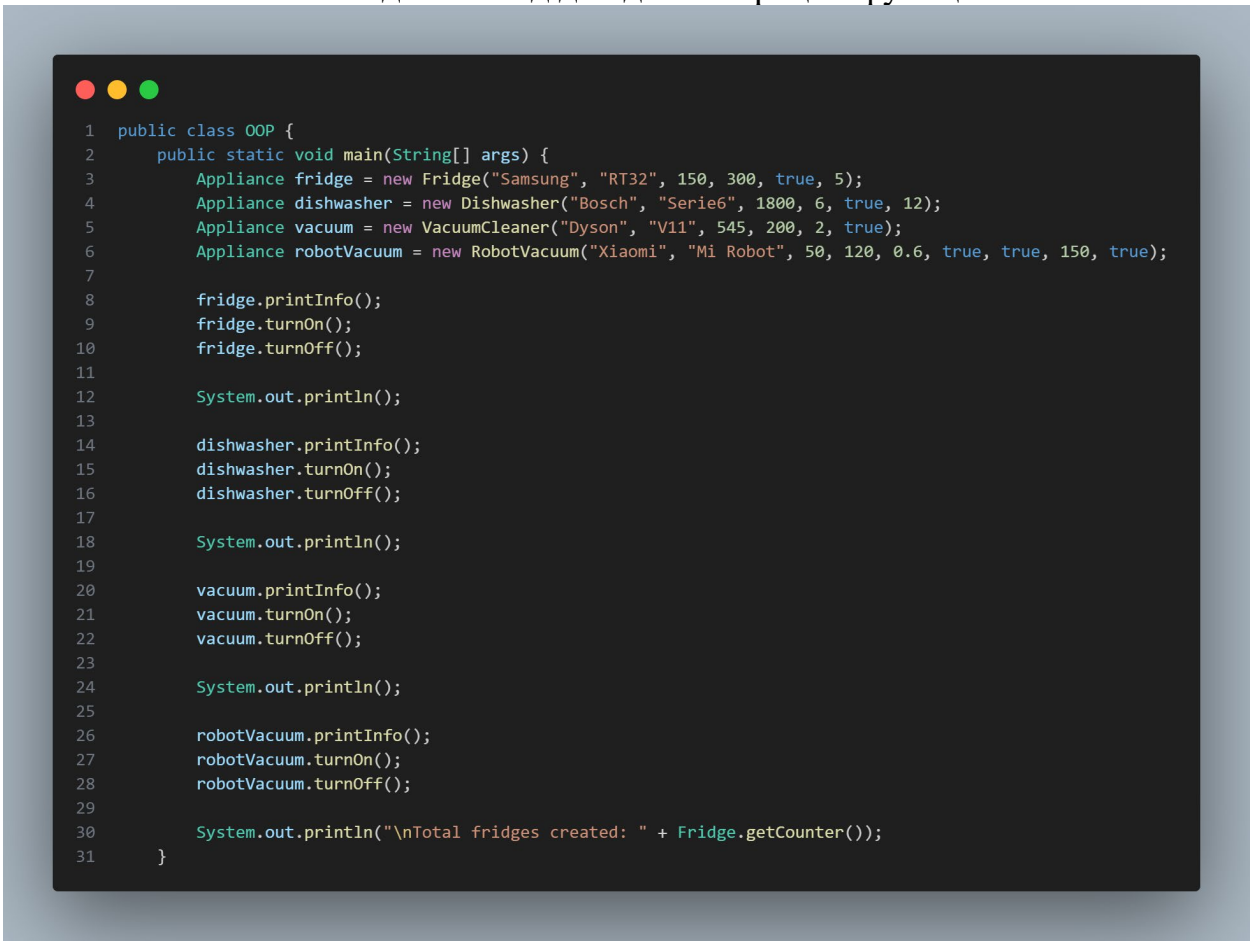
```java
class VacuumCleaner extends Appliance {
    protected int suctionPower;
    protected double dustContainerVolume;
    protected boolean isWireless;

    public VacuumCleaner() {
        this("Unknown", "Unknown", 0, 0, 0, false);
    }

    public VacuumCleaner(String brand, String model, double power, int suctionPower, double dustContainerVolume, boolean isWireless) {
        super(brand, model, power);
        this.suctionPower = suctionPower;
        this.dustContainerVolume = dustContainerVolume;
        this.isWireless = isWireless;
    }

    @Override
    public void turnOn() {
        System.out.println("Vacuum cleaner is on. Suction power: " + suctionPower + " W");
    }

    @Override
    public void turnOff() {
        System.out.println("Vacuum cleaner is off.");
    }

    @Override
    public void printInfo() {
        super.printInfo();
        System.out.println("Suction power: " + suctionPower + " W, Container: " + dustContainerVolume + " L, Wireless: " + isWireless);
    }
}
```

Вот в этом примере все как обычно, за исключением того, что я решил взять этот класс за основу для создания нового дочернего класса, именно поэтому, по аналогии с абстрактным общим классом Appliance, я сделал его поля protected, а не private. Теперь к самому дочернему классу:

```java
class RobotVacuum extends VacuumCleaner {
    private boolean autoNavigation;
    private int batteryLife;
    private boolean mopFunction;

    public RobotVacuum() {
        this("Unknown", "Unknown", 0, 0, 0, false, false, 0, false);
    }

    public RobotVacuum(String brand, String model, double power, int suctionPower, double dustContainerVolume, boolean isWireless,
                       boolean autoNavigation, int batteryLife, boolean mopFunction) {
        super(brand, model, power, suctionPower, dustContainerVolume, isWireless);
        this.autoNavigation = autoNavigation;
        this.batteryLife = batteryLife;
        this.mopFunction = mopFunction;
    }

    @Override
    public void turnOn() {
        System.out.println("Robot vacuum started cleaning. Auto navigation: " + autoNavigation);
    }

    @Override
    public void turnOff() {
        System.out.println("Robot vacuum finished cleaning and returned to base.");
    }

    @Override
    public void printInfo() {
        super.printInfo();
        System.out.println("Auto navigation: " + autoNavigation + ", Battery life: " + batteryLife + " min, Mop function: " + mopFunction);
    }
}
```

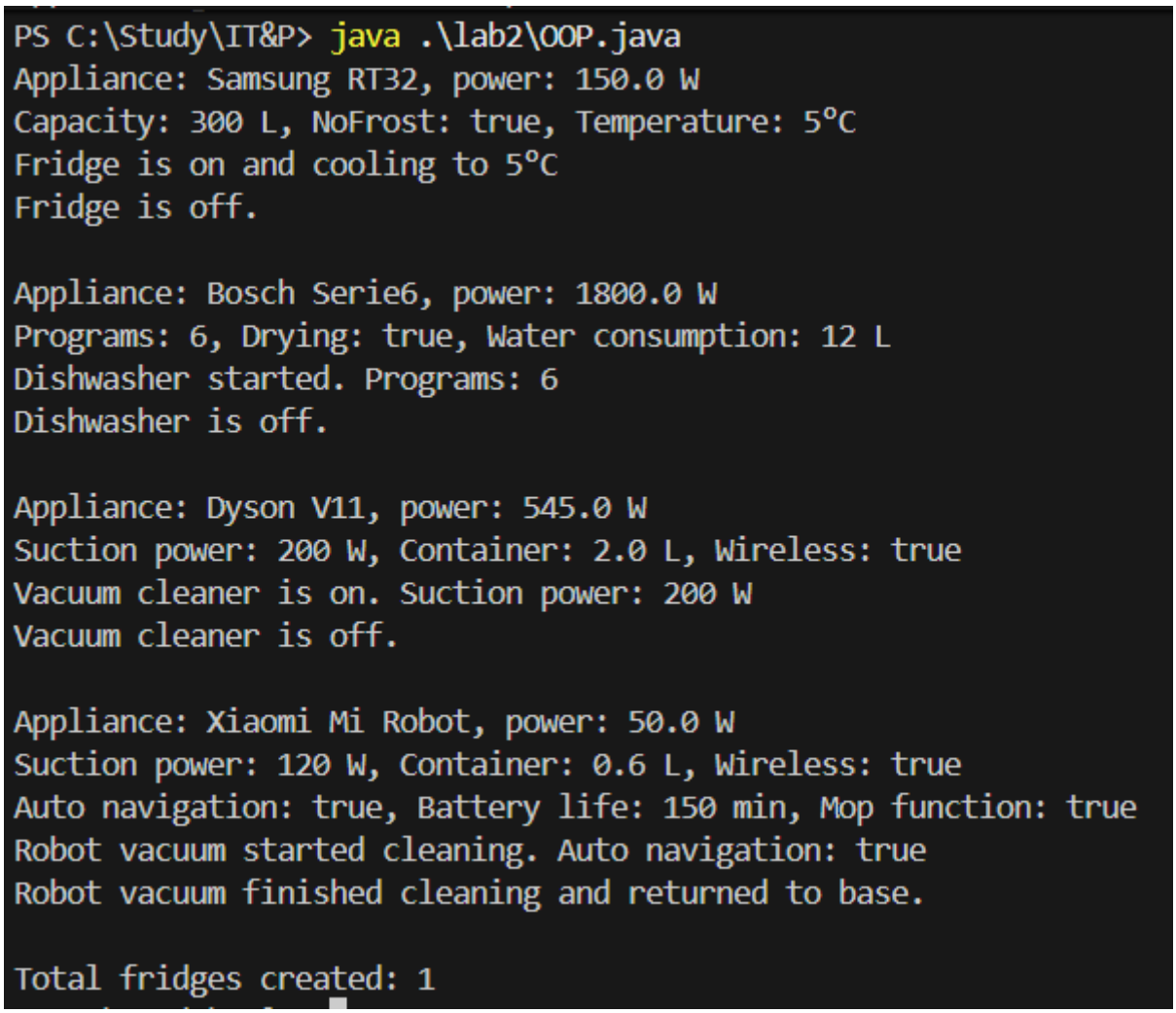
Тут тоже все как обычно, единственное отличие – большее количество полей ввиду двойного наследования, вышло целых 9 показателей, информация о

которых будет выведена в 3 строки, так как super.printInfo() будет вызван как в этом, так и в родительском классе.
Итак, все классы готовы, осталось написать основной класс, создать пару объектов и вызвать каждый метод для демонстрации функционала:



```java
public class OOP {
    public static void main(String[] args) {
        Appliance fridge = new Fridge("Samsung", "RT32", 150, 300, true, 5);
        Appliance dishwasher = new Dishwasher("Bosch", "Serie6", 1800, 6, true, 12);
        Appliance vacuum = new VacuumCleaner("Dyson", "V11", 545, 200, 2, true);
        Appliance robotVacuum = new RobotVacuum("Xiaomi", "Mi Robot", 50, 120, 0.6, true, true, 150, true);

        fridge.printInfo();
        fridge.turnOn();
        fridge.turnOff();

        System.out.println();

        dishwasher.printInfo();
        dishwasher.turnOn();
        dishwasher.turnOff();

        System.out.println();

        vacuum.printInfo();
        vacuum.turnOn();
        vacuum.turnOff();

        System.out.println();

        robotVacuum.printInfo();
        robotVacuum.turnOn();
        robotVacuum.turnOff();

        System.out.println("\nTotal fridges created: " + Fridge.getCounter());
    }
```

Вот код, а вот итоговый вывод:



Вывод

В ходе лабораторной работы мной были освоены все 4 принципа ООП.

Ссылка на гит - https://github.com/BestStarProggramer/IT-P/tree/main/lab2