



**Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)**

---

**ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

**КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

**НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.04.01 Информатика и вычислительная техника**

**МАГИСТЕРСКАЯ ПРОГРАММА 09.04.01/07 Интеллектуальные системы анализа,  
обработки и интерпретации больших данных**

**О Т Ч Е Т**

**по лабораторной работе № 4**

**Вариант № 5**

**Название лабораторной работы:** Внутренние классы. Интерфейсы

**Дисциплина:** Языки программирования для работы с большими данными

Студент гр. ИУ6-22М

\_\_\_\_\_  
(Подпись, дата)

**Р.Г. Гаделия**

(И.О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

**П.В. Степанов**

(И.О. Фамилия)

Москва, 2023

## **Введение**

Целью лабораторной работы является приобретение навыков работы с внутренними классами и интерфейсами на языке программирования Java.

## Практическая часть

### Задание 1

Создать класс Shop (магазин) с внутренним классом, с помощью объектов которого можно хранить информацию об отделах, товарах и услуг.

Код написанной программы представлен в листинге 1.

Листинг 1 – Программа для первого задания

```
package com.java.lab;

import java.util.ArrayList;

// 6 Создать класс Shop (магазин) с внутренним классом, с помощью
// объектов которого можно хранить информацию об отделах, товарах и
// услуг.
public class Main {

    public static void main(String[] args) {
        var shop = new Shop("Federation");
        shop.getStorage().addDepartment(new
StorageDepartment("Sellers"));
        shop.getStorage().addProduct(new StorageProduct("Nike"));
        shop.getStorage().addService(new
StorageServivice("Cleaning"));

        System.out.println(shop.toString());
    }
}

class Shop {
    private String name;

    private final Storage storage;

    Shop(String name) {
        this.name = name;
        this.storage = new Storage();
    }

    String getName() {
        return name;
    }

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

    public Storage getStorage() {
        return storage;
    }
}
```

```

@Override
public String toString() {
    return "Shop{" +
        "name='" + name + '\'' +
        ", storage=" + storage +
        '}';
}

static class Storage {
    private ArrayList<StorageItemI> storage;

    public Storage() {
        this.storage = new ArrayList<>();
    }

    public void addDepartment(StorageDepartment department){
        this.storage.add(department);
    }

    public void addProduct(StorageProduct product){
        this.storage.add(product);
    }

    public void addService(StorageService servivice){
        this.storage.add(servivice);
    }

    @Override
    public String toString() {
        return "Storage{" +
            "storage=" + storage +
            '}';
    }
}

interface StorageItemI {
    String getName();
    void setName(String name);
}

class StorageDepartment implements StorageItemI {
    private String name;

    StorageDepartment(String name) {
        this.name = name;
    }

    @Override
    public String getName() {
        return "Department: " + name;
    }

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

```

```

        @Override
        public String toString() {
            return "StorageDepartment {" +
                "name='" + name + '\'' +
                '}';
        }
    }

class StorageProduct implements StorageItemI {
    private String name;

    StorageProduct(String name) {
        this.name = name;
    }

    @Override
    public String getName() {
        return "Product: " + name;
    }

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

    @Override
    public String toString() {
        return "StorageProduct {" +
            "name='" + name + '\'' +
            '}';
    }
}

class StorageService implements StorageItemI {
    private String name;

    StorageService(String name) {
        this.name = name;
    }

    @Override
    public String getName() {
        return "Service: " + name;
    }

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

    @Override
    public String toString() {
        return "StorageService {" +
            "name='" + name + '\'' +
            '}';
    }
}

```

Результат выполнения программы показан на рисунке 1.

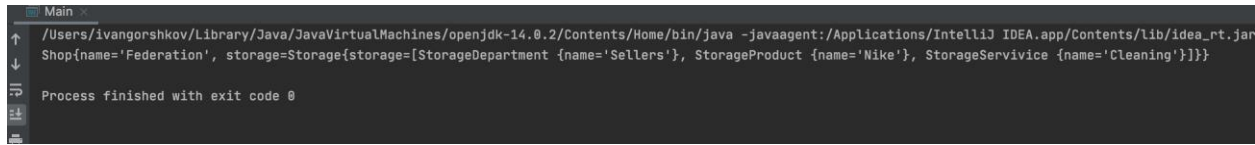


Рисунок 1 – Результат выполнения программы

## Задание 2

Создать класс Справочная Служба Общественного Транспорта с внутренним классом, с помощью объектов которого можно хранить информацию о времени, линиях маршрутов и стоимости проезда.

Код написанной программы представлен в листинге 2.

### Листинг 2 – Программа для второго задания

```
package com.java.lab;

import java.util.ArrayList;

// 7 Создать класс Справочная Служба Общественного Транспорта с
// внутренним классом, с помощью объектов которого можно хранить
// информацию о времени, линиях маршрутов и стоимости проезда.
public class Main {

    public static void main(String[] args) {
        var transportHelper = new TransportHelper();
        transportHelper.addRoute(new TransportHelper.Route("12:00",
"redLine", 123));
        transportHelper.addRoute(new TransportHelper.Route("12:40",
"greenLine", 12));
        transportHelper.addRoute(new TransportHelper.Route("13:10",
"blueLine", 223));
        System.out.println(transportHelper.toString());
    }
}

class TransportHelper {
    ArrayList<Route> routes = new ArrayList<>();

    public void addRoute(Route route) {
        routes.add(route);
    }

    static class Route {
        String time;
        String lineDescription;
        int price;

        public Route(String time, String lineDescription, int price) {
            this.time = time;
            this.lineDescription = lineDescription;
        }
    }
}
```

```

        this.price = price;
    }

    public String getTime() {
        return time;
    }

    public void setTime(String time) {
        this.time = time;
    }

    public String getLineDescription() {
        return lineDescription;
    }

    public void setLineDescription(String lineDescription) {
        this.lineDescription = lineDescription;
    }

    public int getPrice() {
        return price;
    }

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

    @Override
    public String toString() {
        return "Route{" +
            "time='" + time + '\'' +
            ", lineDescription='" + lineDescription + '\'' +
            ", price=" + price +
            '}';
    }
}

@Override
public String toString() {
    return "TransportHelper{" +
        "routes=" + routes +
        '}';
}
}

```

Результат выполнения программы показан на рисунке 2.



```

/Users/ivangorshkov/Library/Java/JavaVirtualMachines/openjdk-14.0.2/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=52908:/Applications/IntelliJ IDEA.app
TransportHelper{routes=[Route{time='12:00', lineDescription='redLine', price=123}, Route{time='12:40', lineDescription='greenLine', price=12}, Route{time='13:10', lineDescription='blueLine', price=10}]}
Process finished with exit code 0

```

Рисунок 2 – Результат выполнения программы

### Задание 3

Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов. interface Корабль <- abstract class Военный Корабль <- class Авианосец.

Код написанной программы представлен в листинге 3.

#### Листинг 3 – Программа для третьего задания

```
package com.java.lab;

// 6.interface Корабль <- abstract class Военный Корабль <- class
Авианосец.

public class Main {

    public static void main(String[] args) {
        AircraftCarrier aircraft = new AircraftCarrier( 190, 35);
        aircraft.sail();
        aircraft.attack();
    }
}

interface Ship {
    void sail();
}

abstract class WarShip implements Ship {
    private int speed;
    public WarShip(int speed) {
        this.speed = speed;
    }

    public int getSpeed() {
        return speed;
    }

    public void setSpeed(int speed) {
        this.speed = speed;
    }

    abstract void attack();

    @Override
    public void sail() {
        System.out.println("Военный корабль плывет со скоростью " +
getSpeed());
    }
}

class AircraftCarrier extends WarShip {
    private int numberOfAircraft;
    public AircraftCarrier(int speed, int numberOfAircraft) {
        super(speed);
    }
}
```



```

        this.numberOfAircraft = numberOfAircraft;
    }

    @Override
    public void sail() {
        System.out.println("Авианосец плывет со скоростью " +
getSpeed());
    }

    public int getNumberOfAircraft() {
        return numberOfAircraft;
    }

    public void setNumberOfAircraft(int numberOfAircraft) {
        this.numberOfAircraft = numberOfAircraft;
    }

    @Override
    public void attack() {
        System.out.println(numberOfAircraft + " самолета вылетели на
атаку");
    }
}

```

Результат выполнения программы показан на рисунке 3.

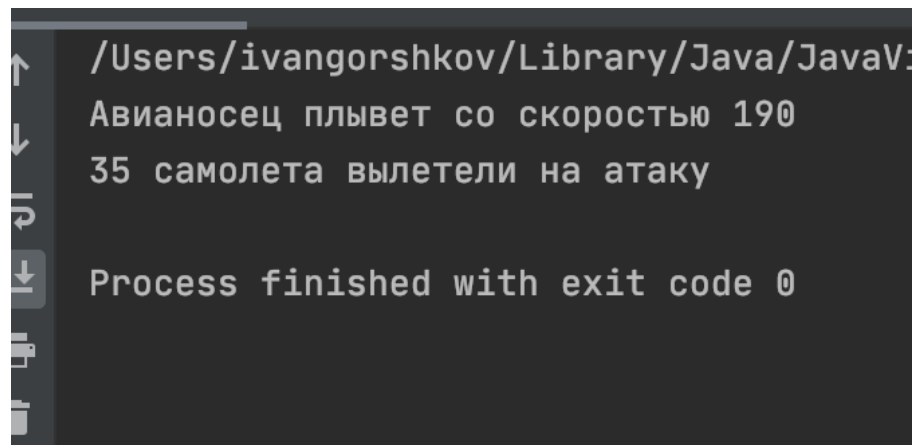


Рисунок 3 – Результат выполнения программы

#### Задание 4

Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов. interface Врач <- class Хирург <- class Нейрохирург.

Код написанной программы представлен в листинге 4.

Листинг 4 – Программа для четвертого задания

```

package com.java.lab;

// 7.interface Врач <- class Хирург <- class Нейрохирург.

public class Main {

```

```

public static void main(String[] args) {
    var surgeon = new Surgeon("Иван");
    var neurosurgeon = new Neurosurgeon("Стёпа");
    doOperation(surgeon);
    doOperation(neurosurgeon);
    checkNeurons(neurosurgeon);
}

static void doOperation(Doctor doctor) {
    doctor.doOperation();
}

static void checkNeurons(Neurosurgeon doctor) {
    doctor.checkNeurons();
}
}

interface Doctor {
    void doOperation();
}

class Surgeon implements Doctor {

    String name;

    Surgeon(String name) {
        this.name = name;
    }

    @Override
    public void doOperation() {
        System.out.println("Врач хирург " + name + " оперирует");
    }
}

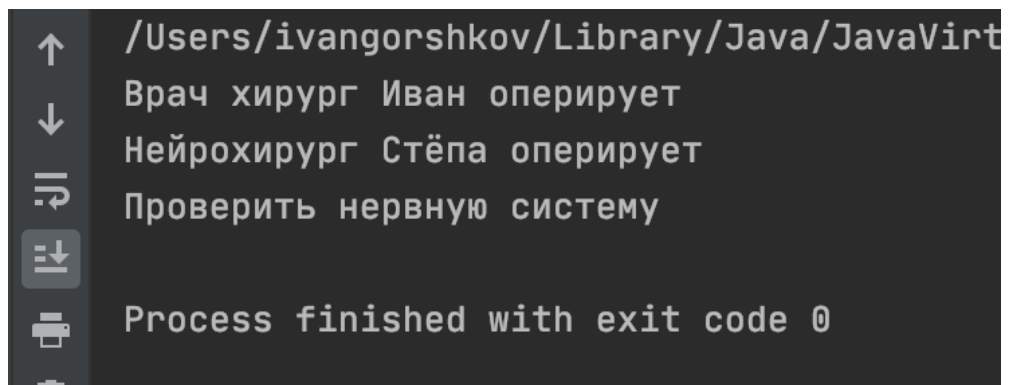
class Neurosurgeon extends Surgeon {
    Neurosurgeon(String name) {
        super(name);
    }

    @Override
    public void doOperation() {
        System.out.println("Нейрохирург " + name + " оперирует");
    }

    public void checkNeurons() {
        System.out.println("Проверить нервную систему");
    }
}
}

```

Результат выполнения программы показан на рисунке 4.

A screenshot of a terminal window with a dark background. On the left side, there is a vertical toolbar with icons for navigation (up, down), undo, redo, and a list view. The terminal text is as follows:

```
/Users/ivangorshkov/Library/Java/JavaVirt  
Врач хирург Иван оперирует  
Нейрохирург Стёпа оперирует  
Проверить нервную систему  
  
Process finished with exit code 0
```

Рисунок 4 – Результат выполнения программы

**Вывод:** В результате выполнения лабораторной работы были приобретены навыки работы с внутренними классами и интерфейсами на языке программирования Java.