



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, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ 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

Название: Внутренние классы. Интерфейсы

Дисциплина: Языки программирования для работы с большими
данными

Студент

ИУ6-22М

(Группа)

(Подпись, дата)

А.М. Панфилкин

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

П.В. Степанов

(И.О. Фамилия)

Москва, 2022 г.

Весь приведенный ниже код также доступен в следующем репозитории:

<https://github.com/SilkSlime/iu6plfbd>

Задание 1: Создать класс Календарь с внутренним классом, с помощью объектов которого можно хранить информацию о выходных и праздничных днях.

Листинг 1 – Задание 1

```
package l4;

import java.time.LocalDate;
import java.time.DayOfWeek;
import java.util.HashSet;

public class e1 {

    /**
     * Вариант 1. Задача 5.
     * Создать класс Календарь с внутренним классом, с помощью объектов
     * которого можно хранить информацию о выходных и праздничных днях.
     */

    public static void main(String[] args) {
        Calendar calendar = new Calendar();
        calendar.addHoliday(LocalDate.of(2023, 1, 1)); // New Year's Day
        calendar.addHoliday(LocalDate.of(2023, 1, 16)); // Martin Luther King Jr. Day
        calendar.addHoliday(LocalDate.of(2023, 2, 20)); // Presidents Day
        calendar.addHoliday(LocalDate.of(2023, 5, 29)); // Memorial Day
        calendar.addHoliday(LocalDate.of(2023, 7, 4)); // Independence Day
        calendar.addHoliday(LocalDate.of(2023, 9, 4)); // Labor Day
        calendar.addHoliday(LocalDate.of(2023, 11, 23)); // Thanksgiving
        calendar.addHoliday(LocalDate.of(2023, 12, 25)); // Christmas

        LocalDate start = LocalDate.of(2023, 1, 1);
        LocalDate end = LocalDate.of(2023, 12, 31);

        Calendar.DayFilter dayFilter = calendar.new DayFilter();

        for (LocalDate date = start; !date.isAfter(end); date = date.plusDays(1)) {
            if (dayFilter.isAllowed(date)) {
                System.out.println(date);
            }
        }
    }

    class Calendar {

        private HashSet<LocalDate> holidays;

        public Calendar() {
```

```

        holidays = new HashSet<LocalDate>();
    }

    public void addHoliday(LocalDate date) {
        holidays.add(date);
    }

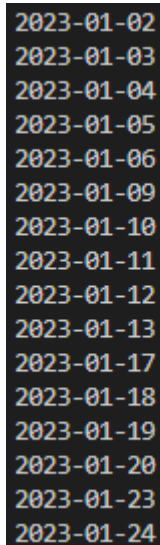
    public boolean isWeekend(LocalDate date) {
        DayOfWeek dayOfWeek = date.getDayOfWeek();
        return dayOfWeek == DayOfWeek.SATURDAY || dayOfWeek == DayOfWeek.SUNDAY;
    }

    public boolean isHoliday(LocalDate date) {
        return holidays.contains(date);
    }

    public class DayFilter {
        public boolean isAllowed(LocalDate date) {
            return !isWeekend(date) && !isHoliday(date);
        }
    }
}

```

Пример результата работы программы показан на рисунке 1.



```

2023-01-02
2023-01-03
2023-01-04
2023-01-05
2023-01-06
2023-01-09
2023-01-10
2023-01-11
2023-01-12
2023-01-13
2023-01-17
2023-01-18
2023-01-19
2023-01-20
2023-01-23
2023-01-24

```

Рисунок 1 – Пример результата работы программы

Задание 2: Создать класс Shop (магазин) с внутренним классом, с помощью объектов которого можно хранить информацию об отделах, товарах и услуг.

Листинг 2 – Задание 2

```

package l4;

import java.util.ArrayList;
import java.util.List;

```

```

public class e2 {

    /**
     * Вариант 1. Задача 6.
     * Создать класс Shop (магазин) с внутренним классом,
     * с помощью объектов которого можно хранить информацию
     * об отделах, товарах и услуг.
     */

    public static void main(String[] args) {
        Shop shop = new Shop("My Shop");
        shop.addDepartment("Electronics");
        shop.addDepartment("Clothing");

        Shop.Department electronics = shop.getDepartment("Electronics");
        electronics.addProduct("TV", 1000);
        electronics.addProduct("Laptop", 800);

        Shop.Department clothing = shop.getDepartment("Clothing");
        clothing.addProduct("Shirt", 50);
        clothing.addProduct("Jeans", 70);

        System.out.println("Welcome to "+shop.getName()+"!");
        System.out.println("Products in Electronics:");
        for (Shop.Department.Product product : electronics.getProducts()) {
            System.out.println(product.getName() + " - " + product.getPrice());
        }

        System.out.println("\nProducts in Clothing:");
        for (Shop.Department.Product product : clothing.getProducts()) {
            System.out.println(product.getName() + " - " + product.getPrice());
        }
    }
}

class Shop {

    private String name;
    private List<Department> departments;

    public Shop(String name) {
        this.name = name;
        this.departments = new ArrayList<>();
    }

    public String getName() {
        return name;
    }

    public void addDepartment(String name) {
        Department department = new Department(name);
        departments.add(department);
    }
}

```

```

    }

    public Department getDepartment(String name) {
        for (Department department : departments) {
            if (department.getName().equals(name)) {
                return department;
            }
        }
        return null;
    }

    public class Department {
        private String name;
        private List<Product> products;

        public Department(String name) {
            this.name = name;
            this.products = new ArrayList<>();
        }

        public void addProduct(String name, double price) {
            Product product = new Product(name, price);
            products.add(product);
        }

        public Product getProduct(String name) {
            for (Product product : products) {
                if (product.getName().equals(name)) {
                    return product;
                }
            }
            return null;
        }

        public String getName() {
            return name;
        }

        public List<Product> getProducts() {
            return products;
        }

        public class Product {
            private String name;
            private double price;

            public Product(String name, double price) {
                this.name = name;
                this.price = price;
            }

            public String getName() {
                return name;
            }
        }
    }

```

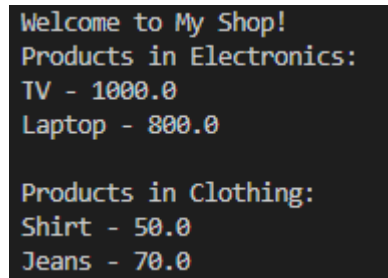
```

    }

    public double getPrice() {
        return price;
    }
}
}
}

```

Пример результата работы программы показан на рисунке 2.



```

Welcome to My Shop!
Products in Electronics:
TV - 1000.0
Laptop - 800.0

Products in Clothing:
Shirt - 50.0
Jeans - 70.0

```

Рисунок 2 – Пример результата работы программы

Задание 3: Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов: interface Mobile - abstract class Siemens Mobile - class Model

Листинг 3 – Задание 3

```

package l4;

public class e3 {

    /**
     * Вариант 2. Задача 5.
     * Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для
     * следующих классов:
     * interface Mobile - abstract class Siemens Mobile - class Model
     */

    public static void main(String[] args) {
        // Example of polymorphism
        Mobile mobile = new Model();
        mobile.call();
        mobile.text();

        // Example of inheritance
        SiemensMobile siemensMobile = new Model();
        siemensMobile.call();
        siemensMobile.sendSMS();
    }
}

```

```

}

interface Mobile {
    void call();
    void text();
}

// Siemens Mobile abstract class
abstract class SiemensMobile implements Mobile {
    public void call() {
        System.out.println("Calling from a Siemens mobile.");
    }

    // Abstract method for texting
    abstract void sendSMS();
}

// Model class that extends SiemensMobile
class Model extends SiemensMobile {
    public void text() {
        System.out.println("Sending text message from Model.");
    }

    // Implementation of sendSMS method
    public void sendSMS() {
        System.out.println("Sending SMS from Model.");
    }
}

```

Пример результата работы программы показан на рисунке 3.

```

Calling from a Siemens mobile.
Sending text message from Model.
Calling from a Siemens mobile.
Sending SMS from Model.

```

Рисунок 3 – Пример результата работы программы

Задание 4: Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов: interface Корабль - abstract class Военный Корабль - class Авианосец

Листинг 4 – Задание 4

```

package l4;

public class e4 {

    /**
     * Вариант 2. Задача 6.

```

```

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

public static void main(String[] args) {
    // Example of polymorphism
    Ship ship = new AircraftCarrier();
    ship.sail();

    // Example of inheritance
    Warship warship = new AircraftCarrier();
    warship.sail();
    warship.attack();
}

// Ship interface
interface Ship {
    void sail();
}

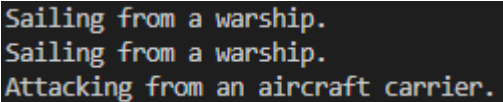
// Warship abstract class that implements Ship interface
abstract class Warship implements Ship {
    public void sail() {
        System.out.println("Sailing from a warship.");
    }

    // Abstract method for attacking
    abstract void attack();
}

// AircraftCarrier class that extends Warship
class AircraftCarrier extends Warship {
    public void attack() {
        System.out.println("Attacking from an aircraft carrier.");
    }
}

```

Пример результата работы программы показан на рисунке 4.



```

Sailing from a warship.
Sailing from a warship.
Attacking from an aircraft carrier.

```

Рисунок 4 – Пример результата работы программы

Вывод: В этой лабораторной работе мы изучили концепции внутренних классов, интерфейсов и абстрактных классов в Java. Внутренние классы — это классы, определенные в рамках другого класса, и они могут получить доступ к закрытым членам данных окружающего класса. Мы создали два класса, Календарь и Магазин, с внутренними классами, позволяющими хранить информацию о выходных и праздничных днях, а также об отделах, товарах и услугах. Интерфейсы и абстрактные классы используются для определения общего поведения классов и позволяют нам использовать полиморфизм для написания кода, который может работать с объектами разных классов. Мы реализовали интерфейс Mobile и абстрактный класс Siemens Mobile, а также интерфейс Ship и абстрактный класс Warship. Мы также создали конкретные классы, которые расширили абстрактные классы, такие как класс Model и класс Aircraft Carrier. В целом, эта лабораторная работа обеспечила четкое понимание концепций внутренних классов, интерфейсов и абстрактных классов в Java. Это важные концепции, которые должен освоить любой Java-программист, поскольку они позволяют нам писать модульный код многократного использования, который можно легко расширять и поддерживать с течением времени.