|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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.**

**Отчет**

**по лабораторной работе № 4**

**Вариант № 12**

**Название:** внутренние классы и интерфейсы

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-22М |  |  | Н.М. Кулинич |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | П.В. Степанов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2024

**Цель:** освоить принципы ООП на языке программирования Java.

**Задание 1:** Создать класс Mobile с внутренним классом, с помощью объектов которого можно хранить информацию о моделях телефонов и их свойствах.

Код класса Mobile:

import java.util.ArrayList;

import java.util.List;

public class Mobile {

// Внутренний класс Model для хранения информации о моделях телефонов

public class Model {

private String modelName;

private String brand;

private double price;

private String[] features;

// Конструктор внутреннего класса

public Model(String modelName, String brand, double price, String[] features) {

this.modelName = modelName;

this.brand = brand;

this.price = price;

this.features = features;

}

// Методы set и get

public void setModelName(String modelName) { this.modelName = modelName; }

public String getModelName() { return modelName; }

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

public String getBrand() { return brand; }

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

public double getPrice() { return price; }

public void setFeatures(String[] features) { this.features = features; }

public String[] getFeatures() { return features; }

// Переопределение метода toString для представления объекта в виде строки

@Override

public String toString() {

return "Model{" +

"modelName='" + modelName + '\'' +

", brand='" + brand + '\'' +

", price=" + price +

", features=" + String.join(", ", features) +

'}';

}

}

// Список моделей телефонов

private List<Model> models;

// Конструктор класса Mobile

public Mobile() {

models = new ArrayList<>();

}

// Метод для добавления новой модели телефона

public void addModel(String modelName, String brand, double price, String[] features) {

Model model = new Model(modelName, brand, price, features);

models.add(model);

}

// Метод для получения списка всех моделей

public List<Model> getModels() {

return models;

}

// Метод для поиска модели по названию

public Model findModelByName(String modelName) {

for (Model model : models) {

if (model.getModelName().equalsIgnoreCase(modelName)) {

return model;

}

}

return null; // Если модель не найдена

}

public static void main(String[] args) {

Mobile mobileStore = new Mobile();

// Добавление моделей телефонов

mobileStore.addModel("iPhone 14", "Apple", 999.99, new String[]{"5G", "128GB Storage", "6GB RAM"});

mobileStore.addModel("Galaxy S22", "Samsung", 899.99, new String[]{"5G", "256GB Storage", "8GB RAM"});

mobileStore.addModel("Pixel 7", "Google", 799.99, new String[]{"5G", "128GB Storage", "8GB RAM"});

// Получение списка всех моделей

System.out.println("All Models:");

for (Model model : mobileStore.getModels()) {

System.out.println(model);

}

// Поиск модели по названию

System.out.println("\nSearch Model by Name:");

Model foundModel = mobileStore.findModelByName("iPhone 14");

if (foundModel != null) {

System.out.println(foundModel);

} else {

System.out.println("Model not found");

}

}

}}}Работа программы показана на рисунке 1.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1 – Работа программы

**Задание 2:** создать класс Художественная Выставка с внутренним классом, с помощью объектов которого можно хранить информацию о картинах, авторах и времени проведения выставок.

Код класса Art Exhibition:

//4. Создать класс Художественная Выставка с внутренним классом, с помощью объектов которого можно хранить

// информацию о картинах, авторах и времени проведения выставок.

import java.util.ArrayList;

import java.util.List;

public class ArtExhibition {

// Внутренний класс Painting для хранения информации о картинах

public class Painting {

private String title;

private String artist;

private String exhibitionDate;

// Конструктор внутреннего класса

public Painting(String title, String artist, String exhibitionDate) {

this.title = title;

this.artist = artist;

this.exhibitionDate = exhibitionDate;

}

// Методы set и get

public void setTitle(String title) { this.title = title; }

public String getTitle() { return title; }

public void setArtist(String artist) { this.artist = artist; }

public String getArtist() { return artist; }

public void setExhibitionDate(String exhibitionDate) { this.exhibitionDate = exhibitionDate; }

public String getExhibitionDate() { return exhibitionDate; }

// Переопределение метода toString для представления объекта в виде строки

@Override

public String toString() {

return "Painting{" +

"title='" + title + '\'' +

", artist='" + artist + '\'' +

", exhibitionDate='" + exhibitionDate + '\'' +

'}';

}

}

// Список картин на выставке

private List<Painting> paintings;

// Конструктор класса ArtExhibition

public ArtExhibition() {

paintings = new ArrayList<>();

}

// Метод для добавления новой картины

public void addPainting(String title, String artist, String exhibitionDate) {

Painting painting = new Painting(title, artist, exhibitionDate);

paintings.add(painting);

}

// Метод для получения списка всех картин

public List<Painting> getPaintings() {

return paintings;

}

// Метод для поиска картины по названию

public Painting findPaintingByTitle(String title) {

for (Painting painting : paintings) {

if (painting.getTitle().equalsIgnoreCase(title)) {

return painting;

}

}

return null; // Если картина не найдена

}

public static void main(String[] args) {

ArtExhibition exhibition = new ArtExhibition();

// Добавление картин

exhibition.addPainting("Starry Night", "Vincent van Gogh", "2024-06-15");

exhibition.addPainting("Mona Lisa", "Leonardo da Vinci", "2024-07-01");

exhibition.addPainting("The Persistence of Memory", "Salvador Dali", "2024-08-20");

// Получение списка всех картин

System.out.println("All Paintings:");

for (Painting painting : exhibition.getPaintings()) {

System.out.println(painting);

}

// Поиск картины по названию

System.out.println("\nSearch Painting by Title:");

Painting foundPainting = exhibition.findPaintingByTitle("Mona Lisa");

if (foundPainting != null) {

System.out.println(foundPainting);

} else {

System.out.println("Painting not found");

}

}

}} Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2 – Работа программы

Работа программы показана на рисунке 2.

**Задание 3:** реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов. Interface Сотрудник <- class Инженер <- class Руководитель.

Код класса Employee:

public interface Employee {

void getDetails();

void performDuties();

}

Код класса Engineer:

public abstract class Engineer implements Employee {

protected String name;

protected String department;

protected double salary;

public Engineer(String name, String department, double salary) {

this.name = name;

this.department = department;

this.salary = salary;

}

@Override

public void getDetails() {

System.out.println("Name: " + name + ", Department: " + department + ", Salary: $" + salary);

}

@Override

public abstract void performDuties(); // Абстрактный метод, должен быть реализован в подклассах

}

Код класса Manger:

public class Manager extends Engineer {

private int numberOfReports;

public Manager(String name, String department, double salary, int numberOfReports) {

super(name, department, salary);

this.numberOfReports = numberOfReports;

}

@Override

public void performDuties() {

System.out.println(name + " is managing " + numberOfReports + " reports.");

}

@Override

public void getDetails() {

super.getDetails();

System.out.println("Number of Reports: " + numberOfReports);

}

}

Работа программы показана на рисунке 3.

Изображение выглядит как текст, снимок экрана, Шрифт, информация

Автоматически созданное описание

Рисунок 3 – Работа программы

**Задание 4:** реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов. Interface Здание <- abstract class Общественное Здание <- class Театр.

Код класса Building:

public interface Building {

void getDetails();

void getCapacity();

}

Код класса PunlicBuilding:

public abstract class PublicBuilding implements Building {

protected String name;

protected String address;

protected int capacity;

public PublicBuilding(String name, String address, int capacity) {

this.name = name;

this.address = address;

this.capacity = capacity;

}

@Override

public void getDetails() {

System.out.println("Name: " + name + ", Address: " + address + ", Capacity: " + capacity);

}

@Override

public abstract void getCapacity(); // Абстрактный метод, должен быть реализован в подклассах

}

Код класса Theter:

public class Theater extends PublicBuilding {

private String[] shows;

public Theater(String name, String address, int capacity, String[] shows) {

super(name, address, capacity);

this.shows = shows;

}

@Override

public void getCapacity() {

System.out.println("Theater Capacity: " + capacity + " seats");

}

@Override

public void getDetails() {

super.getDetails();

System.out.println("Shows: " + String.join(", ", shows));

}

}

**Вывод:** была изучена работа внутренних классов и интерфейсов в java.