|  |  |
| --- | --- |
| 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, 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**

**Отчет**

**по лабораторной работе №4**

**Название:** Внутренние классы и интерфейсы

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

Вариант: 2

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

**Вариант 1**

2. Создать класс CD (mp3-диск) с внутренним классом, с помощью объектов которого можно хранить информацию о каталогах, подкаталогах и записях. Листинг 1 – Код программы

import java.util.ArrayList;

import java.util.List;

public class Main {

/\*\*

\* Вариант 1. Задача 2.

\* Создать класс CD (mp3-диск) с внутренним классом, с помощью объектов

\* которого можно хранить информацию о каталогах, подкаталогах и записях.

\*/

public static void main(String[] args) {

CD cd = new CD("MP3-disk");

cd.addCatalog("Catalog1");

cd.addCatalog("Catalog2");

cd.addCatalog("Catalog2");

CD.Department catalog1 = cd.getDepartment("Catalog1");

catalog1.addSubDir("Subdirectory11", "property 11");

catalog1.addSubDir("Subdirectory12", "property 12");

CD.Department catalog2 = cd.getDepartment("Catalog2");

catalog2.addSubDir("Subdirectory21", "property 21");

catalog2.addSubDir("Subdirectory22", "property 22");

System.out.println("Welcome to "+cd.getName()+"!");

System.out.println("Subdirectories in Catalog1:");

for (CD.Department.Prop prop : catalog1.getProps()) {

System.out.println(prop.getName() + " - " + prop.getProperty());

}

System.out.println("\nSubdirectories in Catalog2:");

for (CD.Department.Prop prop : catalog2.getProps()) {

System.out.println(prop.getName() + " - " + prop.getProperty());

}

}

}

class CD {

private String name;

private List<Department> departments;

public CD(String name) {

this.name = name;

this.departments = new ArrayList<>();

}

public String getName() {

return name;

}

public void addCatalog(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<Prop> props;

public Department(String name) {

this.name = name;

this.props = new ArrayList<>();

}

public void addSubDir(String name, String property) {

Prop prop = new Prop(name, property);

props.add(prop);

}

public Prop getProp(String name) {

for (Prop prop : props) {

if (prop.getName().equals(name)) {

return prop;

}

}

return null;

}

public String getName() {

return name;

}

public List<Prop> getProps() {

return props;

}

public class Prop {

private String name;

private String property;

public Prop(String name, String property) {

this.name = name;

this.property = property;

}

public String getName() {

return name;

}

public String getProperty() {

return property;

}

}

}

}

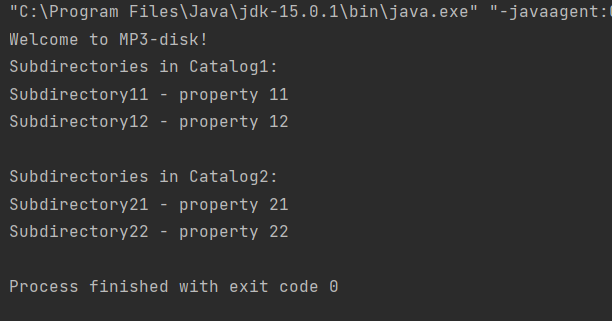


Рисунок 1 – Результат работы программы

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

Листинг 2 – Код программы

import java.util.ArrayList;

import java.util.List;

public class Main {

/\*\*

\* Вариант 1. Задача 3.

\* Создать класс Mobile с внутренним классом,

\* с помощью объектов которого можно хранить информацию о моделях телефонов и их свойствах.

\*/

public static void main(String[] args) {

Mobile mobile = new Mobile("Mobile");

mobile.addCatalog("Model 1");

mobile.addCatalog("Model 2");

Mobile.Department model1 = mobile.getDepartment("Model 1");

model1.addSubDir("Telephon11", "property 11");

model1.addSubDir("Telephon12", "property 12");

Mobile.Department model2 = mobile.getDepartment("Model 2");

model2.addSubDir("Telephon21", "property 21");

model2.addSubDir("Telephon22", "property 22");

System.out.println("Welcome to "+mobile.getName()+"!");

System.out.println("Telephons in model1:");

for (Mobile.Department.Prop prop : model1.getProps()) {

System.out.println(prop.getName() + " - " + prop.getProperty());

}

System.out.println("\nTelephons in model2:");

for (Mobile.Department.Prop prop : model2.getProps()) {

System.out.println(prop.getName() + " - " + prop.getProperty());

}

}

}

class Mobile {

private String name;

private List<Department> departments;

public Mobile(String name) {

this.name = name;

this.departments = new ArrayList<>();

}

public String getName() {

return name;

}

public void addCatalog(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<Prop> props;

public Department(String name) {

this.name = name;

this.props = new ArrayList<>();

}

public void addSubDir(String name, String property) {

Prop prop = new Prop(name, property);

props.add(prop);

}

public Prop getProp(String name) {

for (Prop prop : props) {

if (prop.getName().equals(name)) {

return prop;

}

}

return null;

}

public String getName() {

return name;

}

public List<Prop> getProps() {

return props;

}

public class Prop {

private String name;

private String property;

public Prop(String name, String property) {

this.name = name;

this.property = property;

}

public String getName() {

return name;

}

public String getProperty() {

return property;

}

}

}

}

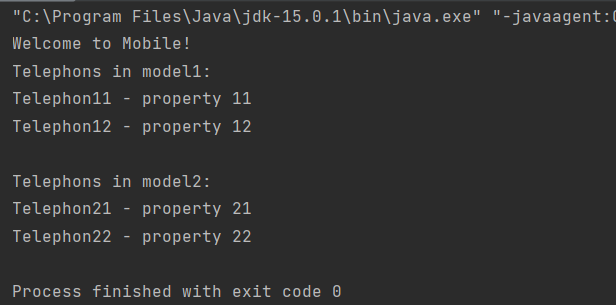


Рисунок 2 – Результат работы программы

**Вариант 2**

Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов  
2. interface Абитуриент <- abstract class Студент <- class Студент-Заочник. Листинг 3 – Код программы

public class Main {

/\*\*

\* Вариант 2. Задача 2.

\* Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов:

\* interface Абитуриент <- abstract class Студент <- class Студент-Заочник

\*/

public static void main(String[] args) {

// Example of polymorphism

Enrollee enrollee = new CorrStud();

enrollee.sleep();

// Example of inheritance

Student student = new CorrStud();

student.sleep();

student.study();

}

}

// Enrollee interface

interface Enrollee {

void sleep();

}

// Student abstract class that implements Enrollee interface

abstract class Student implements Enrollee {

public void sleep() {

System.out.println("student is sleeping.");

}

// Abstract method for study

abstract void study();

}

// CorrStud class that extends Student

class CorrStud extends Student {

public void study() {

System.out.println("correspondence student is studying.");

}

}

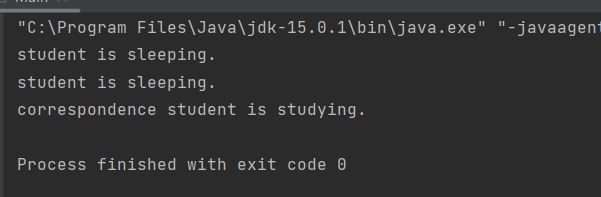


Рисунок 3 – Результат работы программы

3. interface Сотрудник <- class Инженер <- class Руководитель.  
Листинг 4 – Код программы

public class Main {

/\*\*

\* Вариант 2. Задача 3.

\* Реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов:

\* interface Сотрудник <- class Инженер <- class Руководитель.

\*/

public static void main(String[] args) {

// Example of polymorphism

Employee employee = new Supervisor();

employee.sleep();

// Example of inheritance

Engineer engineer = new Supervisor();

engineer.sleep();

engineer.work();

}

}

// Employee interface

interface Employee {

void sleep();

}

// Engineer abstract class that implements Employee interface

abstract class Engineer implements Employee {

public void sleep() {

System.out.println("Engineer is sleeping.");

}

// Abstract method for work

abstract void work();

}

// Supervisor class that extends Engineer

class Supervisor extends Engineer {

public void work() {

System.out.println("Supervisor is working.");

}

}

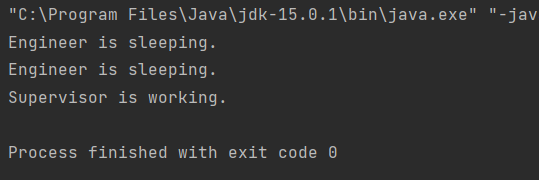


Рисунок 4 – Результат работы программы

**Вывод:** приобретен навык работы с внутренними классами и интерфейсами.

**Ссылка на репозиторий с программами:** [**https://github.com/nargi3/BigData**](https://github.com/nargi3/BigData)