|  |  |
| --- | --- |
| 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. Интерфейсы

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

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

Москва, 2023

**Задания**

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

package lab4

/\*\*

\* 3. Создать класс Mobile с внутренним классом,

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

\*

\*

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

\* 3. interface Сотрудник <- class Инженер <- class Руководитель.

\* 4. interface Здание <- abstract class Общественное Здание <- class Театр.

\*/

class Mobile(

val models: HashSet<Model> = HashSet(),

) {

enum class OS(val n: String) {

ANDROID("Android"),

IOS("IOS");

override fun toString(): String {

return this.n

}

}

fun getOSByName(name: String): OS? {

return models.firstOrNull() { it.name == name }?.os

}

fun getCPUByName(name: String): String? {

return models.firstOrNull() { it.name == name }?.cpu

}

fun getModelsByOS(os: OS): HashSet<Model> {

return models.filter { it.os == os }.toHashSet()

}

inner class Model(

var brand: String,

val name: String,

val os: OS,

val cpu: String,

) {

init {

models.add(this)

}

override fun toString(): String {

return "$brand $name: ${os.n} on CPU $cpu"

}

}

}

fun main() {

val mobile = Mobile()

mobile.Model("Apple", "iPhone 14 Pro MAX", Mobile.OS.IOS, "A16 Bionic")

mobile.Model("Google", "Pixel 7 Pro", Mobile.OS.ANDROID, "Google Tensor G2")

mobile.Model("Samsung", "Galaxy S22 Ultra", Mobile.OS.ANDROID, "Snapdragon 8 Gen 1")

println(mobile.getModelsByOS(Mobile.OS.ANDROID))

}

4. Создать класс Художественная Выставка с внутренним классом, с помощью объектов которого можно хранить информацию о картинах, авторах и времени проведения выставок.

package lab4

import java.util.\*

/\*\*

\* 4. Создать класс Художественная Выставка с внутренним классом,

\* с помощью объектов которого можно хранить информацию о картинах, авторах и времени проведения выставок.

\*/

class ArtExhibition {

private val exhibitions: HashSet<Exhibition> = HashSet()

inner class Exhibition(val name: String, val datetime: Date) {

init {

exhibitions.add(this)

}

private val artworks: HashSet<Artwork> = HashSet()

fun addArtwork(author: String, name: String) {

artworks.add(Artwork(author, name))

}

fun getArtworkByName(name: String): Artwork? {

return artworks.firstOrNull { it.name == name }

}

fun getArtworksByAuthor(author: String): HashSet<Artwork> {

return artworks.filter { it.author == author }.toHashSet()

}

fun getArtworks(): HashSet<Artwork> {

return artworks

}

inner class Artwork(val author: String, val name: String)

}

}

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

3. interface Сотрудник <- class Инженер <- class Руководитель.

package lab4

/\*\*

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

\* 3. interface Сотрудник <- class Инженер <- class Руководитель.

\*/

internal interface Employee {

fun doTask()

fun getGrade(): String

}

internal abstract class Engineer : Employee {

override fun doTask() {

println("Task done.")

}

abstract fun solveProblem()

}

internal class Boss : Engineer() {

override fun getGrade(): String {

return "25 Grade"

}

override fun solveProblem() {

println("Problem solved")

}

}

4. interface Здание <- abstract class Общественное Здание <- class Театр.

package lab4

/\*\*

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

\* 4. interface Здание <- abstract class Общественное Здание <- class Театр.

\*/

internal interface Building {

fun addToFavorites()

fun getInfo(): String

}

internal abstract class PublicPlace : Building {

override fun addToFavorites() {

println("Added to favorites")

}

abstract fun visit()

}

internal class Theater : PublicPlace() {

override fun getInfo(): String {

return "Theater"

}

override fun visit() {

println("Theater visited")

}

}

**Вывод:** В этой лабораторной работе мы изучили концепции внутренних классов, интерфейсов и абстрактных классов в Kotlin. В целом, эта лабораторная работа обеспечила четкое понимание концепций внутренних классов, интерфейсов и абстрактных классов в Kotlin. Это важные концепции, которые должен освоить любой Kotlin-программист, поскольку они позволяют нам писать модульный код многократного использования, который можно легко расширять и поддерживать с течением времени.