Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Дисциплина: Программирование мобильных информационных систем

Отчёт

К лабораторной работе № 4

на тему:

**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ (ООП)**

Проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ф.В. Усенко

(подпись)

Выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Шубин

(подпись) 214302

Минск, 2024

**Цель работы:** Изучить принципы объектно-ориентированного программирования (ООП) в Kotlin, включая классы, объекты, наследование, полиморфизм, интерфейсы и абстрактные классы. Научиться создавать и использовать собственные классы, а также применять принципы ООП на практике.

**Задание для самостоятельной работы (Вариант 8).**

**Реализация системы бронирования:** Разработайте систему для бронирования мест в отеле. Создайте классы Room, Guest, Reservation, и реализуйте методы для управления бронированием, обработки запросов, изменения брони, и расчета стоимости проживания с учетом дополнительных услуг.

**Код программы.**

var *reservation* = Reservation(*mutableListOf*())  
var *guest*: Guest = Guest(*reservation*)  
  
fun main() {  
  
 val listOfHotelRooms = *mutableListOf*(  
 Room(11110, "стандартный", 1, 120.0, 0, *mutableListOf*()),  
 Room(11111, "улучшенный/супериор", 2, 175.5, 0, *mutableListOf*()),  
 Room(11112, "со спальной комнатой", 2, 200.75, 0, *mutableListOf*()),  
 Room(11113, "люкс", 2, 310.0,0, *mutableListOf*()),  
 Room(11114, "люкс", 5, 410.5,0, *mutableListOf*()))  
  
 *println*("Добро пожаловать в систему бронирования мест в отеле!!!")  
 *userMenu*(listOfHotelRooms)  
}  
  
fun userMenu(listOfHotelRooms: MutableList<Room>){  
  
 while(true) {  
  
 *println*(  
 "\nВыберите пункт меню:" +  
 "\n1. Забронировать номер\n2. Мои номера" +  
 "\n3. Изменить бронь" +  
 "\n4. Просмотр стоимости бронирования" +  
 "\nВЫХОД - любая другая клавиша"  
 )  
  
 val input = *readlnOrNull*()?.*toIntOrNull*()  
  
 when (input) {  
 1 -> {  
 *roomBooking*(listOfHotelRooms)  
 }  
 2 ->{  
 if(*guest*.reservation.myRooms.*isNullOrEmpty*()){  
 *println*("Вы еще не забронировали ни одного номера")  
 }else{  
 for(room in *guest*.reservation.myRooms!!){  
 *println*(  
 "\n№ комнаты: ${room.roomNumb}; тип номера: ${room.roomType}; вместимость номера: ${room.capacity} +" +  
 " чел.; стоимость (за одну ночь с учетом доп. услуг): ${room.cost} бел.руб."  
 )  
 *println*("Включенные дополнительные услуги:")  
 var i = 1  
 for(service in room.additionalService!!){  
 *println*(  
 "${i}) ${service.name}"  
 )  
 i++  
 }  
 }  
 }  
 }  
 3 -> {  
 var i = 1  
 if(*guest*.reservation.myRooms.*isNullOrEmpty*()){  
 *println*("Вы еще не забронировали ни одного номера")  
 }else {  
 *alterBookings*(listOfHotelRooms)  
 }  
 }  
 4 -> {  
 var generalCost = 0.0  
 for (room in *guest*.reservation.myRooms!!){  
 generalCost += room.cost \* room.nightsAmount  
 }  
 *println*("Общая стоимость проживания с учетом дополнительных услуг: ${generalCost} бел.руб.")  
 }  
 else -> {  
 break  
 }  
 }  
 }  
}  
  
fun roomBooking(listOfHotelRooms: MutableList<Room>){  
 while (true) {  
 val additionalServiceList = *mutableListOf*(  
 AdditionalService("Не интересуют дополнительные услуги", 0.0),  
 AdditionalService("Двухразовое питание в отеле (завтрак, ужин)", 50.0),  
 AdditionalService("Трехразовое питание в отеле", 75.9),  
 AdditionalService("Пользование сауной", 150.0),  
 AdditionalService("Пользование спортивным залом", 95.5),  
 AdditionalService("Пользование камерой хранения или сейфом", 30.0)  
 )  
 *println*("Доступные номера для бронирования:")  
 var i = 1  
 for (room in listOfHotelRooms) {  
 *println*(  
 "${i}) № комнаты: ${room.roomNumb}; тип номера: ${room.roomType}; вместимость номера: ${room.capacity}" +  
 "чел.; стоимость (за одну ночь): ${room.cost} бел.руб."  
 )  
 i++  
 }  
 val roomChoice = *readlnOrNull*()?.*toIntOrNull*()  
 if (roomChoice != null) {  
 if (roomChoice < 1 || roomChoice > listOfHotelRooms.size) {  
 *println*("Ошибка ввода!!! Выберите доступный номер для бронирования")  
 continue  
 } else {  
 while(true){  
 *println*("Пожалуйста, выберите интересующие дополнительные услуги" +  
 "\nДоступные доп. услуги:")  
 var j = 1  
 for (service in additionalServiceList) {  
 if(service == additionalServiceList.get(0)){  
 continue  
 }  
 *println*(  
 "${j}) ${service.name} (стоимость за ночь: ${service.price} бел.руб.)"  
 )  
 j++  
 }  
 *println*(  
 "0) ${additionalServiceList.get(0).name}"  
 )  
 val serviceChoice = *readlnOrNull*()?.*toIntOrNull*()  
 if(serviceChoice != null){  
 if (serviceChoice < 0 || serviceChoice > additionalServiceList.size - 1) {  
 *println*("Ошибка ввода!!! Выберите доступную дополнительную услугу")  
 continue  
 } else {  
 val bookedRoom = listOfHotelRooms.get(roomChoice - 1)  
 bookedRoom.addService(additionalServiceList.get(serviceChoice))  
 additionalServiceList.remove(additionalServiceList.get(serviceChoice))  
 *print*("Желаете добавить другие доп. услуги?\n1.Да\n НЕТ - любая другая клавиша\n")  
 val input = *readlnOrNull*()?.*toIntOrNull*()  
 if(input != 1){  
 *reservation*.book(bookedRoom)  
 listOfHotelRooms.remove(listOfHotelRooms.get(roomChoice - 1))  
 break  
 }  
 }  
 }  
 }  
 while (true) {  
 try {  
 *println*("\nВведите желаемое количество ночей, на которое хотите забронировать выбранный номер: ")  
 val input = *readlnOrNull*() ?: throw IllegalArgumentException("Введено пустое значение.")  
 val nightsAmount = input.*toIntOrNull*() ?: throw NumberFormatException("Введено не число.")  
 if (nightsAmount <= 0) {  
 throw IllegalArgumentException("Количество ночей должно быть больше нуля.")  
 }  
 *reservation*.myRooms!![*reservation*.myRooms!!.size - 1].nightsAmount = nightsAmount  
 *guest*.addReservation(*reservation*)  
 *println*("Вы забронировали номер на $nightsAmount ночей.")  
 break  
 } catch (e: IllegalArgumentException) {  
 *println*("Ошибка: ${e.message}")  
 continue  
 } catch (e: NumberFormatException) {  
 *println*("Ошибка: Введите действительное число.")  
 continue  
 }  
 }  
 *print*("Желаете продолжить бронирование?\n1.Да\n НЕТ - любая другая клавиша\n")  
 val input = *readlnOrNull*()?.*toIntOrNull*()  
 if(input != 1)break  
 }  
 }  
 }  
}  
  
fun alterBookings(listOfHotelRooms: MutableList<Room>){  
 *println*("Для изменения брони необходимо отменить текущую бронь и забронировать номер заново\n" +  
 "Выберите номер, бронирование которого хотели бы отменить:")  
 var i = 1  
 for (room in *guest*.reservation.myRooms!!) {  
 *println*(  
 "${i}) № комнаты: ${room.roomNumb}; тип номера: ${room.roomType}; вместимость номера: ${room.capacity}" +  
 " чел.; стоимость (за одну ночь с учетом доп. услуг): ${room.cost} бел.руб."  
 )  
 i++  
 }  
 while(true) {  
 try {  
 var roomIndex = 0  
 var flag = false  
 val input = *readlnOrNull*() ?: throw IllegalArgumentException("Введено пустое значение.")  
 val roomChoice = input.*toIntOrNull*() ?: throw NumberFormatException("Введено не число.")  
 for (room in *guest*.reservation.myRooms!!) {  
 if(room.roomNumb == *guest*.reservation.myRooms!!.get(roomChoice - 1).roomNumb){  
 flag = true  
 break  
 }  
 roomIndex++  
 }  
 if(!flag){  
 *println*("Ошибка ввода. Невозможно изменить бронь комнаты с заданным номером")  
 }  
 val bookedRoomToRemove = *guest*.reservation.myRooms!!.get(roomIndex)  
 listOfHotelRooms.add(bookedRoomToRemove)  
 listOfHotelRooms.*sortBy* **{ it**.roomNumb **}** *guest*.reservation.myRooms!!.remove(bookedRoomToRemove)  
 break  
 } catch (e: IllegalArgumentException) {  
 *println*("Ошибка: ${e.message}")  
 continue  
 } catch (e: NumberFormatException) {  
 *println*("Ошибка: Введите действительное число.")  
 continue  
 }  
 }  
 *roomBooking*(listOfHotelRooms)  
}

**Контрольные вопросы.**

1. Что такое класс в Kotlin, и как он объявляется?

Класс в Kotlin — это основная строительная единица объектов, которая позволяет определять типы и структуры данных. Классы определяют свойства (переменные) и методы (функции), которые принадлежат объектам этого класса. Класс в Kotlin объявляется с использованием ключевого слова class. Класс может содержать свойства и методы, а также быть шаблоном для создания объектов.

1. Как объявить и использовать метод класса? Приведите пример.

В Kotlin методы класса объявляются внутри тела класса. Метод — это функция, которая принадлежит классу и может быть вызвана на объектах класса:

class Car(val brand: String, val model: String, var year: Int) {

fun printInfo() {

println("Brand: $brand, Model: $model, Year: $year")

}

fun updateYear(newYear: Int) {

year = newYear

}

}

fun main() {

val myCar = Car("Toyota", "Camry", 2015)

// Вызов метода printInfo

myCar.printInfo()

// Вызов метода updateYear

myCar.updateYear(2021)

}

1. Как в Kotlin создать вторичный конструктор, и зачем он может понадобиться?

Вторичные конструкторы используются для предоставления дополнительных способов инициализации объектов. Они объявляются с использованием ключевого слова constructor:

class Car {

val brand: String

val model: String

var year: Int

constructor(brand: String, model: String) {

this.brand = brand

this.model = model

this.year = 2021

}

}

1. Что такое абстрактный класс, и как объявить абстрактный метод?

Абстрактные классы используются, когда необходимо создать базовый класс, который нельзя инстанцировать, но который может содержать как абстрактные, так и не абстрактные методы:

abstract class Animal {

abstract fun sound()

fun sleep() {

println("Animal is sleeping")

}

}

class Dog : Animal() {

override fun sound() {

println("Dog barks")

}

}

1. Что такое компаньон-объект, и как его использовать для создания единственного экземпляра класса?

Компаньон-объект — это специальный объект внутри класса, который ведет себя как статический метод или поле в Java. Компаньон-объекты позволяют обращаться к методам и свойствам класса без создания экземпляра:

class MyClass {

companion object {

fun create(): MyClass = MyClass()

}

}

val instance = MyClass.create()

Компаньон-объекты полезны, когда нужно предоставить функции для работы с классом, которые не зависят от конкретного объекта.

1. В чем разница между статическим методом в Java и методом компаньон-объекта в Kotlin?

В Java статический метод определяется с помощью ключевого слова static. Эти методы принадлежат самому классу, а не его экземпляру, поэтому они могут быть вызваны без создания объекта этого класса.

В Kotlin нет прямого аналога статических методов. Вместо этого используется компаньон-объект (companion object). Методы, объявленные внутри компаньон-объекта, также могут быть вызваны без создания экземпляра класса.