Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Программирование мобильных информационных систем

Отчёт

по лабораторной работе №4

на тему

**Объектно-ориентированное программирование**

Выполнил: Проверил:

ст. гр. 214302 Усенко Ф.В.

Коршов М.И.

Минск 2024

Вариант 15. Система управления библиотекой с поддержкой аренды: Напишите систему управления библиотекой, включающую классы *Book*, *LibraryMember*, и *Rental*. Реализуйте методы для управления каталогом книг, учета аренды и возврата книг, а также для расчета штрафов за просрочку

Листинг кода:

data class Book(

private val \_id: Int,

private val \_title: String,

private val \_author: String,

private var \_isAvailable: Boolean = true

) {

val id: Int

get() = \_id

val title: String

get() = \_title

val author: String

get() = \_author

var isAvailable: Boolean

set(value) {

\_isAvailable = value

}

get() = \_isAvailable

override fun toString(): String {

return "$\_id: $\_title by $\_author; available: $\_isAvailable"

}

}

import java.time.LocalDate

class Library {

private val books: MutableList<Book> = mutableListOf()

private val members: MutableList<LibraryMember> = mutableListOf()

private var \_lastBookIdUsed = 0

private var \_lastLibraryMemberIdUsed = 0

var lastBookIdUsed

get() = \_lastBookIdUsed

set(value) {

\_lastBookIdUsed = value

}

var lastLibraryMemberIdUsed

get() = \_lastLibraryMemberIdUsed

set(value) {

\_lastLibraryMemberIdUsed = value

}

fun getMemberById(memberId: Int): LibraryMember {

return members.find { it.id == memberId }!!

}

fun getBookById(bookId: Int): Book {

return books.find { it.id == bookId }!!

}

fun areLibraryMembers(): Boolean {

return members.isNotEmpty()

}

fun areLibraryMembersWithRentals(): Boolean {

return members.any(LibraryMember::hasRentals)

}

fun areLibraryMembersWithActiveRentals(): Boolean {

return members.any(LibraryMember::hasActiveRentals)

}

fun areBooks(): Boolean {

return books.isNotEmpty()

}

fun areAvailableBooks(): Boolean {

return books.any(Book::isAvailable)

}

fun addBook(book: Book) {

books.add(book)

}

fun registerMember(member: LibraryMember): Boolean {

if (members.find { it.id == member.id } == null) {

members.add(member)

return true

}

return false

}

fun isSuchMember(id: Int): Boolean {

return members.find { it.id == id } != null

}

fun isSuchBook(id: Int): Boolean {

return books.find { it.id == id } != null

}

fun printLibraryMembers() {

if (members.isEmpty())

println("No members in library")

else

members.forEach { println(it) }

}

fun printLibraryMembersWithRentals() {

members.filter { it.hasRentals() }.forEach { println(it) }

}

fun printLibraryMembersWithActiveRentals() {

members.filter { it.hasActiveRentals() }.forEach { println(it) }

}

fun printBooks() {

if (books.isEmpty())

println("No books in library")

else

books.forEach { println(it) }

}

fun printAvailableBooks() {

books.forEach { if (it.isAvailable) println(it) }

}

fun rentBook(memberId: Int, bookId: Int, rentalDate: LocalDate = LocalDate.now().minusWeeks(3)): Boolean {

val member = members.find { it.id == memberId }

val book = books.find { it.id == bookId }

if (member != null && book != null) {

val rental = Rental(book, rentalDate)

member.rentedBooks.add(rental)

book.isAvailable = false

return true

} else {

return false

}

}

fun returnBook(memberId: Int, bookId: Int, returnDate: LocalDate): Boolean {

val member = members.find { it.id == memberId }

val rental = member?.rentedBooks?.find { it.book.id == bookId && it.returnDate == null }

if (member != null && rental != null) {

rental.returnDate = returnDate

rental.book.isAvailable = true

return true

} else {

return false

}

}

fun calculateFineForMember(memberId: Int, currentDate: LocalDate, finePerDay: Double = 5.0): Double {

val member = members.find { it.id == memberId }

return member?.rentedBooks?.sumOf { it.calculateFineForRental(currentDate, finePerDay) } ?: 0.0

}

fun calculateFineForLibrary(currentDate: LocalDate, finePerDay: Double = 5.0): Double {

var sum = 0.0

members.forEach { member ->

sum += member.rentedBooks.sumOf {

it.calculateFineForRental(

currentDate,

finePerDay

)

}

}

return sum

}

}

data class LibraryMember(

val id: Int,

val name: String,

val rentedBooks: MutableList<Rental> = mutableListOf(),

) {

override fun toString(): String {

return "$id: $name"

}

fun printRentals() {

rentedBooks.forEach { println(it) }

}

fun printActiveRentals() {

rentedBooks.forEach { if (it.returnDate == null) println(it) }

}

fun hasRentals(): Boolean {

return rentedBooks.isNotEmpty()

}

fun hasActiveRentals(): Boolean {

return rentedBooks.any { it.returnDate == null }

}

}

import java.time.LocalDate

data class Rental(

private val \_book: Book,

private val \_rentalDate: LocalDate,

private var \_returnDate: LocalDate? = null,

) {

val book: Book

get() = \_book

val rentalDate: LocalDate

get() = \_rentalDate

var returnDate: LocalDate?

set(value) {

\_returnDate = value

}

get() = \_returnDate

fun calculateFineForRental(currentDate: LocalDate, finePerDay: Double): Double {

val dueDate = \_rentalDate.plusWeeks(2)

val overdueDate = \_returnDate ?: currentDate

if (overdueDate.isAfter(dueDate)) {

val daysLate = overdueDate.toEpochDay() - dueDate.toEpochDay()

return daysLate \* finePerDay

} else {

return 0.0

}

}

override fun toString(): String {

return "$\_book; rentalDate=$\_rentalDate; returnDate=$\_returnDate"

}

}

import java.time.LocalDate

fun main() {

val library = Library()

while (true) {

println("Choose action:\n1)Add book\n2)Register library member\n3)Add rental\n4)Remove rental\n5)Calculate total fine\n6)Print books\n7)Print library members\n8)Print library member rentals\n9)Exit")

var choice: Int?

try {

choice = readlnOrNull()?.toInt()

if (choice != null && (choice < 1 || choice > 9)) {

println("No such option")

continue

}

} catch (e: NumberFormatException) {

println("Not a number")

continue

}

when (choice) {

null -> continue

1 -> addBook(library)

2 -> addLibraryMember(library)

3 -> addRental(library)

4 -> removeRental(library)

5 -> calculateTotalFine(library)

6 -> library.printBooks()

7 -> library.printLibraryMembers()

8 -> printMemberRentals(library)

9 -> break

}

}

}

fun addBook(library: Library) {

val id = library.lastBookIdUsed

library.lastBookIdUsed++

var title: String?

while (true) {

println("Input book title")

title = readlnOrNull()

if (title.isNullOrEmpty()) {

println("Input a valid book title")

continue

}

break

}

var author: String?

while (true) {

println("Input book author")

author = readlnOrNull()

if (author.isNullOrEmpty()) {

println("Input a valid book author")

continue

}

break

}

library.addBook(Book(id, title.toString(), author.toString()))

}

fun addLibraryMember(library: Library) {

val id = library.lastLibraryMemberIdUsed

library.lastLibraryMemberIdUsed++

var name: String?

while (true) {

println("Input library member name")

name = readlnOrNull()

if (name.isNullOrEmpty()) {

println("Input a library member name")

continue

}

break

}

library.registerMember(LibraryMember(id, name.toString()))

}

fun addRental(library: Library) {

println("Current library members:")

if (library.areLibraryMembers())

library.printLibraryMembers()

else {

println("No library members")

return

}

println("Current available library books:")

if (library.areAvailableBooks())

library.printAvailableBooks()

else {

println("No available books in library")

return

}

var memberId: Int

while (true) {

println("Choose library member id")

try {

val inputId = readlnOrNull()?.toInt()

if (inputId == null || !library.isSuchMember(inputId)) {

println("No such member")

continue

}

memberId = inputId.toInt()

} catch (e: NumberFormatException) {

println("Not a valid id")

continue

}

break

}

val libraryMember = library.getMemberById(memberId)

var bookId: Int

while (true) {

println("Choose book")

try {

val inputId = readlnOrNull()?.toInt()

if (inputId == null || !library.isSuchBook(inputId)) {

println("No such book")

continue

}

bookId = inputId.toInt()

} catch (e: NumberFormatException) {

println("Not a valid id")

continue

}

val bookToRent = library.getBookById(bookId)

if (!bookToRent.isAvailable) {

println("This book isn't available")

continue

}

break

}

library.rentBook(memberId, bookId)

println("The book was successfully rented!")

libraryMember.printRentals()

}

fun removeRental(library: Library) {

println("Current library members with active rentals:")

if (library.areLibraryMembersWithActiveRentals())

library.printLibraryMembersWithActiveRentals()

else {

println("No library members with active rentals")

return

}

var memberId: Int

var libraryMember: LibraryMember

while (true) {

println("Choose library member id")

try {

val inputId = readlnOrNull()?.toInt()

if (inputId == null || !library.isSuchMember(inputId)) {

println("No such member")

continue

}

memberId = inputId.toInt()

} catch (e: NumberFormatException) {

println("Not a valid id")

continue

}

libraryMember = library.getMemberById(memberId)

if (!libraryMember.hasActiveRentals()) {

println("Not a valid member")

continue

}

break

}

println("Chosen member active rentals:")

libraryMember.printActiveRentals()

var bookId: Int

while (true) {

println("Choose book to return")

try {

val inputId = readlnOrNull()?.toInt()

if (inputId == null || libraryMember.rentedBooks.find {it.book.id == inputId && it.returnDate == null} == null) {

println("No such book rented")

continue

}

bookId = inputId.toInt()

} catch (e: NumberFormatException) {

println("Not a valid id")

continue

}

break

}

library.returnBook(memberId, bookId, LocalDate.now())

println("The book was successfully returned!")

libraryMember.printRentals()

}

fun printMemberRentals(library: Library) {

println("Current library members with rentals:")

if (library.areLibraryMembersWithRentals())

library.printLibraryMembersWithRentals()

else {

println("No library members with rentals")

return

}

var memberId: Int

var libraryMember: LibraryMember

while (true) {

println("Choose library member id")

try {

val inputId = readlnOrNull()?.toInt()

if (inputId == null || !library.isSuchMember(inputId)) {

println("No such member")

continue

}

memberId = inputId.toInt()

} catch (e: NumberFormatException) {

println("Not a valid id")

continue

}

libraryMember = library.getMemberById(memberId)

break

}

println("Chosen member rentals:")

libraryMember.printRentals()

println("Chosen member fine: ${library.calculateFineForMember(memberId, LocalDate.now())}")

}

fun calculateTotalFine(library: Library) {

println("Total fine for library:")

println(library.calculateFineForLibrary(LocalDate.now()))

}

Контрольные вопросы:

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

Класс в *Kotlin* – это шаблон для создания объектов, который может содержать свойства (данные) и методы (функции). Класс объявляется с использованием ключевого слова *class*.

1. Как создать объект класса в *Kotlin*? Приведите пример.

Объекты создаются с помощью вызова конструктора класса.

Пример:

val myCar = Car("Toyota", "Corolla", 2020)

1. Что такое свойства класса, и как их объявить в *Kotlin*?

Свойства класса – это переменные, которые хранят состояние объекта. Свойства класса могут быть изменяемыми (*var*) или неизменяемыми (*val*) и объявляются внутри класса. Они могут быть инициализированы сразу или в конструкторе.