**Московский государственный технический**

**университет им. Н.Э. Баумана**

**Факультет ИУ**

**Кафедра ИУ5**

**Курс «Парадигмы и конструкции языков программирования»**

**Отчет по домашнему заданию**

Выполнил студент группы ИУ5-33Б:

Костяев В.Д.

Подпись и дата:

Проверил преподаватель каф.:

Гапанюк Ю. Е.

Подпись и дата:

Москва, 2024 г

**Задание:**

1. **Выберите язык программирования (который Вы ранее не изучали) и (1) напишите по нему реферат с примерами кода или (2) реализуйте на нем небольшой проект (с детальным текстовым описанием).**
2. **Реферат (проект) может быть посвящен отдельному аспекту (аспектам) языка или содержать решение какой-либо задачи на этом языке.**
3. **Необходимо установить на свой компьютер компилятор (интерпретатор, транспилятор) этого языка и произвольную среду разработки.**
4. **В случае написания реферата необходимо разработать и откомпилировать примеры кода (или модифицировать стандартные примеры).**
5. **В случае создания проекта необходимо детально комментировать код.**
6. **При написании реферата (создании проекта) необходимо изучить и корректно использовать особенности парадигмы языка и основных конструкций данного языка.**
7. **Приветствуется написание черновика статьи по результатам выполнения ДЗ. Черновик статьи может быть подготовлен группой студентов, которые исследовали один и тот же аспект в нескольких языках или решили одинаковую задачу на нескольких языках.**

**Текст программы:**

**Book.cs**

using System;

namespace LibraryManagementSystem

{

// Класс, представляющий книгу в библиотеке.

public class Book

{

// Уникальный идентификатор книги

public int Id { get; set; }

// Название книги

public string Title { get; set; }

// Автор книги

public string Author { get; set; }

// Год издания

public int Year { get; set; }

// Доступна ли книга для выдачи

public bool IsAvailable { get; set; }

// Конструктор по умолчанию

public Book() { }

// Конструктор с параметрами

public Book(int id, string title, string author, int year)

{

Id = id;

Title = title;

Author = author;

Year = year;

IsAvailable = true;

}

// Переопределение метода ToString для удобного вывода информации о книге

public override string ToString()

{

return $"ID: {Id}, Название: {Title}, Автор: {Author}, Год: {Year}, Доступна: {IsAvailable}";

}

}

}

**User.cs**

using System;

namespace LibraryManagementSystem

{

// Класс, представляющий пользователя библиотеки.

public class User

{

// Уникальный идентификатор пользователя

public int Id { get; set; }

// Имя пользователя

public string Name { get; set; }

// Электронная почта

public string Email { get; set; }

// Конструктор по умолчанию

public User() { }

// Конструктор с параметрами

public User(int id, string name, string email)

{

Id = id;

Name = name;

Email = email;

}

// Переопределение метода ToString для удобного вывода информации о пользователе

public override string ToString()

{

return $"ID: {Id}, Имя: {Name}, Email: {Email}";

}

}

}

**BorrowRecord.cs**

using System;

namespace LibraryManagementSystem

{

// Класс, представляющий запись о выдаче книги пользователю.

public class BorrowRecord

{

// Уникальный идентификатор записи

public int RecordId { get; set; }

// Идентификатор книги

public int BookId { get; set; }

// Идентификатор пользователя

public int UserId { get; set; }

// Дата выдачи книги

public DateTime BorrowDate { get; set; }

// Дата возврата книги

public DateTime? ReturnDate { get; set; }

// Конструктор по умолчанию

public BorrowRecord() { }

// Конструктор с параметрами

public BorrowRecord(int recordId, int bookId, int userId)

{

RecordId = recordId;

BookId = bookId;

UserId = userId;

BorrowDate = DateTime.Now;

ReturnDate = null;

}

// Переопределение метода ToString для удобного вывода информации о записи

public override string ToString()

{

string returnDateStr = ReturnDate.HasValue ? ReturnDate.Value.ToString() : "Не возвращена";

return $"Record ID: {RecordId}, Book ID: {BookId}, User ID: {UserId}, Borrow Date: {BorrowDate}, Return Date: {returnDateStr}";

}

}

}

**Library.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using Newtonsoft.Json;

namespace LibraryManagementSystem

{

// Класс, управляющий библиотекой, включая книги, пользователей и записи о выдаче.

public class Library

{

// Списки книг, пользователей и записей о выдаче

public List<Book> Books { get; set; }

public List<User> Users { get; set; }

public List<BorrowRecord> BorrowRecords { get; set; }

// Пути к файлам для сохранения данных

private readonly string booksFile = Path.Combine("data", "books.json");

private readonly string usersFile = Path.Combine("data", "users.json");

private readonly string borrowRecordsFile = Path.Combine("data", "borrowRecords.json");

// Конструктор

public Library()

{

Books = new List<Book>();

Users = new List<User>();

BorrowRecords = new List<BorrowRecord>();

LoadData();

}

// Загрузка данных из файлов.

private void LoadData()

{

try

{

if (File.Exists(booksFile))

{

string booksJson = File.ReadAllText(booksFile);

Books = JsonConvert.DeserializeObject<List<Book>>(booksJson) ?? new List<Book>();

}

if (File.Exists(usersFile))

{

string usersJson = File.ReadAllText(usersFile);

Users = JsonConvert.DeserializeObject<List<User>>(usersJson) ?? new List<User>();

}

if (File.Exists(borrowRecordsFile))

{

string recordsJson = File.ReadAllText(borrowRecordsFile);

BorrowRecords = JsonConvert.DeserializeObject<List<BorrowRecord>>(recordsJson) ?? new List<BorrowRecord>();

}

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при загрузке данных: {ex.Message}");

}

}

// Сохранение данных в файлы.

public void SaveData()

{

try

{

// Убедимся, что директория data существует

Directory.CreateDirectory("data");

string booksJson = JsonConvert.SerializeObject(Books, Formatting.Indented);

File.WriteAllText(booksFile, booksJson);

string usersJson = JsonConvert.SerializeObject(Users, Formatting.Indented);

File.WriteAllText(usersFile, usersJson);

string recordsJson = JsonConvert.SerializeObject(BorrowRecords, Formatting.Indented);

File.WriteAllText(borrowRecordsFile, recordsJson);

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при сохранении данных: {ex.Message}");

}

}

// Добавление новой книги в библиотеку.

public void AddBook(Book book)

{

Books.Add(book);

SaveData();

Console.WriteLine("Книга успешно добавлена.");

}

// Удаление книги по ID.

public void RemoveBook(int bookId)

{

Book book = Books.FirstOrDefault(b => b.Id == bookId);

if (book != null)

{

if (book.IsAvailable)

{

Books.Remove(book);

SaveData();

Console.WriteLine("Книга успешно удалена.");

}

else

{

Console.WriteLine("Невозможно удалить книгу, так как она выдана пользователю.");

}

}

else

{

Console.WriteLine("Книга с таким ID не найдена.");

}

}

// Поиск книги по названию или автору.

public List<Book> SearchBooks(string query)

{

return Books.Where(b => b.Title.Contains(query, StringComparison.OrdinalIgnoreCase) ||

b.Author.Contains(query, StringComparison.OrdinalIgnoreCase)).ToList();

}

// Отображение всех книг в библиотеке.

public void DisplayAllBooks()

{

if (Books.Count == 0)

{

Console.WriteLine("В библиотеке нет книг.");

return;

}

foreach (var book in Books)

{

Console.WriteLine(book);

}

}

// Добавление нового пользователя.

public void AddUser(User user)

{

Users.Add(user);

SaveData();

Console.WriteLine("Пользователь успешно добавлен.");

}

// Удаление пользователя по ID.

public void RemoveUser(int userId)

{

User user = Users.FirstOrDefault(u => u.Id == userId);

if (user != null)

{

// Проверим, нет ли у пользователя выданных книг

bool hasBorrowedBooks = BorrowRecords.Any(r => r.UserId == userId && !r.ReturnDate.HasValue);

if (!hasBorrowedBooks)

{

Users.Remove(user);

SaveData();

Console.WriteLine("Пользователь успешно удален.");

}

else

{

Console.WriteLine("Невозможно удалить пользователя, так как у него есть выданные книги.");

}

}

else

{

Console.WriteLine("Пользователь с таким ID не найден.");

}

}

// Отображение всех пользователей.

public void DisplayAllUsers()

{

if (Users.Count == 0)

{

Console.WriteLine("В библиотеке нет зарегистрированных пользователей.");

return;

}

foreach (var user in Users)

{

Console.WriteLine(user);

}

}

// Выдача книги пользователю.

public void BorrowBook(int bookId, int userId)

{

Book book = Books.FirstOrDefault(b => b.Id == bookId);

User user = Users.FirstOrDefault(u => u.Id == userId);

if (book == null)

{

Console.WriteLine("Книга с таким ID не найдена.");

return;

}

if (user == null)

{

Console.WriteLine("Пользователь с таким ID не найден.");

return;

}

if (!book.IsAvailable)

{

Console.WriteLine("Книга в настоящее время недоступна для выдачи.");

return;

}

// Создаем новую запись о выдаче

int newRecordId = BorrowRecords.Count > 0 ? BorrowRecords.Max(r => r.RecordId) + 1 : 1;

BorrowRecord record = new BorrowRecord(newRecordId, bookId, userId);

BorrowRecords.Add(record);

// Обновляем статус книги

book.IsAvailable = false;

SaveData();

Console.WriteLine("Книга успешно выдана пользователю.");

}

// Возврат книги.

public void ReturnBook(int bookId, int userId)

{

// Находим соответствующую запись о выдаче

BorrowRecord record = BorrowRecords.FirstOrDefault(r => r.BookId == bookId && r.UserId == userId && !r.ReturnDate.HasValue);

if (record == null)

{

Console.WriteLine("Соответствующая запись о выдаче не найдена.");

return;

}

// Устанавливаем дату возврата

record.ReturnDate = DateTime.Now;

// Обновляем статус книги

Book book = Books.FirstOrDefault(b => b.Id == bookId);

if (book != null)

{

book.IsAvailable = true;

}

SaveData();

Console.WriteLine("Книга успешно возвращена.");

}

// Отображение всех записей о выдаче.

public void DisplayAllBorrowRecords()

{

if (BorrowRecords.Count == 0)

{

Console.WriteLine("Нет записей о выдаче книг.");

return;

}

foreach (var record in BorrowRecords)

{

Console.WriteLine(record);

}

}

}

}

**Program.cs**

using System;

namespace LibraryManagementSystem

{

// Класс Program - точка входа в приложение.

class Program

{

static void Main(string[] args)

{

Library library = new Library();

bool exit = false;

Console.WriteLine("=== Система Управления Библиотекой ===");

while (!exit)

{

ShowMenu();

Console.Write("Выберите опцию: ");

string choice = Console.ReadLine();

switch (choice)

{

case "1":

AddBook(library);

break;

case "2":

RemoveBook(library);

break;

case "3":

SearchBooks(library);

break;

case "4":

library.DisplayAllBooks();

break;

case "5":

AddUser(library);

break;

case "6":

RemoveUser(library);

break;

case "7":

library.DisplayAllUsers();

break;

case "8":

BorrowBook(library);

break;

case "9":

ReturnBook(library);

break;

case "10":

library.DisplayAllBorrowRecords();

break;

case "0":

exit = true;

break;

default:

Console.WriteLine("Неверный выбор. Пожалуйста, попробуйте снова.");

break;

}

Console.WriteLine();

}

Console.WriteLine("Спасибо за использование системы! До свидания.");

}

// Отображение меню опций.

static void ShowMenu()

{

Console.WriteLine("======================================");

Console.WriteLine("1. Добавить книгу");

Console.WriteLine("2. Удалить книгу");

Console.WriteLine("3. Поиск книги");

Console.WriteLine("4. Показать все книги");

Console.WriteLine("5. Добавить пользователя");

Console.WriteLine("6. Удалить пользователя");

Console.WriteLine("7. Показать всех пользователей");

Console.WriteLine("8. Выдать книгу пользователю");

Console.WriteLine("9. Принять возврат книги");

Console.WriteLine("10. Показать все записи о выдаче");

Console.WriteLine("0. Выход");

Console.WriteLine("======================================");

}

// Добавление новой книги.

static void AddBook(Library library)

{

Console.WriteLine("=== Добавление Книги ===");

try

{

Console.Write("Введите ID книги (целое число): ");

int id = int.Parse(Console.ReadLine());

// Проверка на уникальность ID

if (library.Books.Exists(b => b.Id == id))

{

Console.WriteLine("Книга с таким ID уже существует.");

return;

}

Console.Write("Введите название книги: ");

string title = Console.ReadLine();

Console.Write("Введите автора книги: ");

string author = Console.ReadLine();

Console.Write("Введите год издания: ");

int year = int.Parse(Console.ReadLine());

Book book = new Book(id, title, author, year);

library.AddBook(book);

}

catch (FormatException)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите правильные данные.");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

}

// Удаление книги.

static void RemoveBook(Library library)

{

Console.WriteLine("=== Удаление Книги ===");

try

{

Console.Write("Введите ID книги для удаления: ");

int id = int.Parse(Console.ReadLine());

library.RemoveBook(id);

}

catch (FormatException)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите правильный ID.");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

}

// Поиск книг.

static void SearchBooks(Library library)

{

Console.WriteLine("=== Поиск Книг ===");

Console.Write("Введите название или автора для поиска: ");

string query = Console.ReadLine();

var results = library.SearchBooks(query);

if (results.Count == 0)

{

Console.WriteLine("Книги по заданному запросу не найдены.");

}

else

{

Console.WriteLine($"Найдено {results.Count} книг(а):");

foreach (var book in results)

{

Console.WriteLine(book);

}

}

}

// Добавление нового пользователя.

static void AddUser(Library library)

{

Console.WriteLine("=== Добавление Пользователя ===");

try

{

Console.Write("Введите ID пользователя (целое число): ");

int id = int.Parse(Console.ReadLine());

// Проверка на уникальность ID

if (library.Users.Exists(u => u.Id == id))

{

Console.WriteLine("Пользователь с таким ID уже существует.");

return;

}

Console.Write("Введите имя пользователя: ");

string name = Console.ReadLine();

Console.Write("Введите email пользователя: ");

string email = Console.ReadLine();

User user = new User(id, name, email);

library.AddUser(user);

}

catch (FormatException)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите правильные данные.");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

}

// Удаление пользователя.

static void RemoveUser(Library library)

{

Console.WriteLine("=== Удаление Пользователя ===");

try

{

Console.Write("Введите ID пользователя для удаления: ");

int id = int.Parse(Console.ReadLine());

library.RemoveUser(id);

}

catch (FormatException)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите правильный ID.");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

}

// Выдача книги пользователю.

static void BorrowBook(Library library)

{

Console.WriteLine("=== Выдача Книги ===");

try

{

Console.Write("Введите ID книги: ");

int bookId = int.Parse(Console.ReadLine());

Console.Write("Введите ID пользователя: ");

int userId = int.Parse(Console.ReadLine());

library.BorrowBook(bookId, userId);

}

catch (FormatException)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите правильные ID.");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

}

// Возврат книги пользователем.

static void ReturnBook(Library library)

{

Console.WriteLine("=== Возврат Книги ===");

try

{

Console.Write("Введите ID книги: ");

int bookId = int.Parse(Console.ReadLine());

Console.Write("Введите ID пользователя: ");

int userId = int.Parse(Console.ReadLine());

library.ReturnBook(bookId, userId);

}

catch (FormatException)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите правильные ID.");

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

}

}

}

**Экранные формы с примерами выполнения программы**





