Представляет книгу в библиотеке.

public class Book

{

Уникальный идентификатор книги.

Используется для различения книг в системе.

public string Id { get; set; }

Название книги.

Содержит название, под которым книга известна.

public string Title { get; set; }

Автор книги.

Указывает на имя или псевдоним автора, который написал книгу.

public string Author { get; set; }

Год издания книги.

Содержит год, в котором была впервые опубликована книга.

public int YearPublished { get; set; }

}

Менеджер для управления списком книг.

Позволяет добавлять, удалять и получать книги.

public class BookManager

{

Список книг, хранящихся в менеджере.

Инициализируется пустым списком при создании экземпляра класса.

public List<Book> Books { get; private set; } = new List<Book>();

Добавляет новую книгу в список.

<param name="book">Книга, которую нужно добавить.</param>

<exception cref="ArgumentException">Выдается, если книга уже существует или является невалидной.</exception>

public void AddBook(Book book)

{

Проверяем, является ли книга нулевой и существует ли книга с таким же Id.

if (book == null || Books.Any(b => b.Id == book.Id))

{

throw new ArgumentException("Книга уже существует или невалидная.");

}

Добавляем книгу в список.

Books.Add(book);

}

Удаляет книгу из списка по ее уникальному идентификатору.

<param name="bookId">Уникальный идентификатор книги, которую нужно удалить.</param>

<exception cref="KeyNotFoundException">Выдается, если книга не найдена в списке.</exception>

public void RemoveBook(string bookId)

{

Ищем книгу по уникальному идентификатору.

var book = Books.SingleOrDefault(b => b.Id == bookId);

if (book == null)

{

throw new KeyNotFoundException("Книга не найдена.");

}

Удаляем книгу из списка.

Books.Remove(book);

}

Получает книгу по ее уникальному идентификатору.

<param name="bookId">Уникальный идентификатор книги.</param>

<returns>Возвращает книгу, соответствующую заданному идентификатору.</returns>

<exception cref="KeyNotFoundException">Выдается, если книга не найдена в списке.</exception>

public Book GetBook(string bookId)

{

Ищем книгу по уникальному идентификатору.

var book = Books.SingleOrDefault(b => b.Id == bookId);

if (book == null)

{

throw new KeyNotFoundException("Книга не найдена.");

}

Возвращаем найденную книгу.

return book;

}

Получает список всех книг.

<returns>Список всех книг в менеджере.</returns>

public List<Book> GetAllBooks()

{

Возвращаем весь список книг.

return Books;

}

}

Класс, представляющий читателя.

internal class Reader

{

Уникальный идентификатор читателя.

public string Id { get; set; }

Имя читателя.

public string Name { get; set; }

Эл. почта читателя.

public string Email { get; set; }

}

Менеджер для управления списком читателей.

Позволяет добавлять, удалять и получать читателей.

internal class ReaderManager

{

Список читателей, хранящихся в менеджере.

Инициализируется пустым списком при создании экземпляра класса.

public List<Reader> Readers { get; private set; } = new List<Reader>();

Добавляет нового читателя в список.

<param name="reader">Читатель, которого нужно добавить.</param>

<exception cref="ArgumentException">Выдается, если читатель уже существует или является невалидным.</exception>

public void AddReader(Reader reader)

{

Проверяем, является ли читатель нулевым и существует ли читатель с таким же Id.

if (reader == null || Readers.Any(r => r.Id == reader.Id))

{

throw new ArgumentException("Читатель уже существует или невалидный.");

}

Добавляем читателя в список.

Readers.Add(reader);

}

Удаляет читателя из списка по его уникальному идентификатору.

<param name="readerId">Уникальный идентификатор читателя, которого нужно удалить.</param>

<exception cref="KeyNotFoundException">Выдается, если читатель не найден в списке.</exception>

public void RemoveReader(string readerId)

{

Ищем читателя по уникальному идентификатору.

var reader = Readers.SingleOrDefault(r => r.Id == readerId);

if (reader == null)

{

throw new KeyNotFoundException("Читатель не найден.");

}

Удаляем читателя из списка.

Readers.Remove(reader);

}

Получает читателя по его уникальному идентификатору.

<param name="readerId">Уникальный идентификатор читателя.</param>

<returns>Возвращает читателя, соответствующего заданному идентификатору.</returns>

<exception cref="KeyNotFoundException">Выдается, если читатель не найден в списке.</exception>

public Reader GetReader(string readerId)

{

Ищем читателя по уникальному идентификатору.

var reader = Readers.SingleOrDefault(r => r.Id == readerId);

if (reader == null)

{

throw new KeyNotFoundException("Читатель не найден.");

}

Возвращаем найденного читателя.

return reader;

}

Получает список всех читателей.

<returns>Список всех читателей в менеджере.</returns>

public List<Reader> GetAllReaders()

{

Возвращаем весь список читателей.

return Readers;

}

}

Главный класс программы, отвечающий за взаимодействие с пользователем.

internal class Program

{

Главный метод, точка входа в программу.

<param name="args">Аргументы командной строки (не используются).</param>

static void Main(string[] args)

{

// Создаем экземпляры менеджеров книг и читателей.

var bookManager = new BookManager();

var readerManager = new ReaderManager();

// Бесконечный цикл для выбора действий пользователем.

while (true)

{

// Выводим меню выбора действий.

Console.WriteLine("Введите действие: 1 - Управление книгами, 2 - Управление читателями, 0 - Выход");

string choice = Console.ReadLine();

// Если пользователь выбрал 0, выходим из программы.

if (choice == "0")

break;

// Обработка выбора пользователя для управления книгами или читателями.

if (choice == "1")

{

ManageBooks(bookManager);

}

else if (choice == "2")

{

ManageReaders(readerManager);

}

else

{

// Если введено неизвестное действие, сообщаем пользователю.

Console.WriteLine("Неизвестное действие.");

}

}

}

Метод для управления книгами.

<param name="bookManager">Менеджер книг, используемый для операций с книгами.</param>

static void ManageBooks(BookManager bookManager)

{

Бесконечный цикл для управления книгами.

while (true)

{

Выводим меню выбора действий для управления книгами.

Console.WriteLine("1 - Добавить книгу, 2 - Удалить книгу, 3 - Показать все книги, 0 - Назад");

string choice = Console.ReadLine();

Если пользователь выбрал 0, возвращаемся в предыдущее меню.

if (choice == "0") break;

try

{

Обработка выбора пользователя для добавления, удаления или отображения книг.

switch (choice)

{

case "1":

Создаем новый объект книги.

var book = new Book();

Console.Write("ID книги: ");

book.Id = Console.ReadLine();

Console.Write("Название книги: ");

book.Title = Console.ReadLine();

Console.Write("Автор книги: ");

book.Author = Console.ReadLine();

Console.Write("Год издания: ");

Читаем год издания и преобразуем к целочисленному типу.

book.YearPublished = int.Parse(Console.ReadLine());

Добавляем книгу через менеджер.

bookManager.AddBook(book);

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

break;

case "2":

Запрашиваем ID книги для удаления.

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

string bookIdToRemove = Console.ReadLine();

Удаляем книгу через менеджер.

bookManager.RemoveBook(bookIdToRemove);

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

break;

case "3":

Получаем список всех книг и отображаем их.

var books = bookManager.GetAllBooks();

foreach (var b in books)

{

Console.WriteLine($"ID: {b.Id}, Название: {b.Title}, Автор: {b.Author}, Год издания: {b.YearPublished}");

}

break;

default:

Если выбрано неизвестное действие, сообщаем пользователю.

Console.WriteLine("Неизвестное действие.");

break;

}

}

catch (Exception ex)

{

Обрабатываем все исключения и выводим сообщение об ошибке.

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

}

}

}

Метод для управления читателями.

<param name="readerManager">Менеджер читателей, используемый для операций с читателями.</param>

static void ManageReaders(ReaderManager readerManager)

{

Бесконечный цикл для управления читателями.

while (true)

{

Выводим меню выбора действий для управления читателями.

Console.WriteLine("1 - Добавить читателя, 2 - Удалить читателя, 3 - Показать всех читателей, 0 - Назад");

string choice = Console.ReadLine();

Если пользователь выбрал 0, возвращаемся в предыдущее меню.

if (choice == "0") break;

try

{

Обработка выбора пользователя для добавления, удаления или отображения читателей.

switch (choice)

{

case "1":

Создаем новый объект читателя.

var reader = new Reader();

Console.Write("ID читателя: ");

reader.Id = Console.ReadLine();

Console.Write("Имя читателя: ");

reader.Name = Console.ReadLine();

Console.Write("Электронная почта: ");

reader.Email = Console.ReadLine();

Добавляем читателя через менеджер.

readerManager.AddReader(reader);

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

break;

case "2":

Запрашиваем ID читателя для удаления.

Console.Write("ID читателя для удаления: ");

string readerIdToRemove = Console.ReadLine();

Удаляем читателя через менеджер.

readerManager.RemoveReader(readerIdToRemove);

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

break;

case "3":

Получаем список всех читателей и отображаем их на экране.

var readers = readerManager.GetAllReaders();

foreach (var r in readers)

{

Выводим информацию о каждом читателе.

Console.WriteLine($"ID: {r.Id}, Имя: {r.Name}, Email: {r.Email}");

}

break;

default:

Если выбрано неизвестное действие, сообщаем пользователю.

Console.WriteLine("Неизвестное действие.");

break;

}

}

catch (Exception ex)

{

Обрабатываем все исключения и выводим сообщение об ошибке.

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

}

}

}