МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Розрахунково-графічна робота № 1**

з дисципліни « Архітектура програмного забезпечення .Net »

*назва дисципліни*

на тему: «ШАБЛОНИ ПРОЕКТУВАННЯ»

Виконав: студент 2 курсу групи № 622п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Зайченко Ярослав Ігорович

(прізвище й ініціали студента)

Прийняв: к. т. н., доцент

Лучшев Павло Олександрович

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2023

**ЗМІСТ**

[Постановка завдання 3](#_Toc159492133)

[Порядок виконання роботи 3](#_Toc159492134)

[Виконання роботи 5](#_Toc159492135)

[Висновок 11](#_Toc159492136)

Мета роботи: Вивчення стандартних ситуацій у процесі розробки складних програмних проектів та застосування шаблонів проектування (Design patterns) для їх вирішення.

Таблиця 1 – виконання вимог до роботи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Складність | Вимоги до роботи | Бали | Оцінка |
|  | Базовий рівень | Текстовий опис шаблону, його основних складових частин, їх призначення (з посиланням на першоджерело! ) | 5 | 5 |
|  | UML модель шаблону у вигляді Mermaid діаграми | 5 | 5 |
|  | Програмний проект на С# , що демонструє специфіку шаблону проектування (кожен шаблон окремо) | 5 | 5 |
|  | Реалізація шаблону проектування додатковою (будь-якою) мовою програмування | 5 | 5 |
|  | Підвищений рівень | Створення шаблону проектування (design pattern) у вигляді шаблону проекту або елемента (project / item template) для Visual Studio (zip -файл) | 5 | 5 |

Постановка завдання

Самостійно знайти в мережі Інтернет (відповідно до варіанта) опис шаблонів проектування (Design patterns) наступних типів:

* що породжує (Creational patterns);
* структурного (Structural patterns);
* поведінкового (Behavioral pattern);
* паралельних обчислень (Concurrency pattern).

У репозиторії GitHub створити файл ReadMe.md і на підставі зібраного матеріалу сформувати текстовий опис шаблону та його графічне подання у вигляді відповідних UML-діаграм:

* статичної моделі (діаграма класів та/або діаграма модулів);
* динамічної моделі (діаграма взаємодії та/або стану);

Для побудови діаграм використовувати інструмент візуалізації Mermaid, який формує зображення з текстового опису на основі мови Markdown .

На практичному етапі для кожного шаблону проектування розробити програмний проект, який демонструє особливості застосування заданих шаблонів проектування практично.

Для завдання підвищеного рівня складності необхідний шаблон проектування (design pattern) оформити у вигляді zip-файлу, який є шаблоном проекту або елемента ( Project / Item Template ) для середовища розробки Visual Studio.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Група . № | Creational  pattern | Structural pattern | Behavioral pattern | Concurrency  pattern |
| 622п.10 | Multiton | Adapter or Wrapper | Interpreter | Lock |

Склад розрахунково-графічної роботи:

1. ***GitHub*** репозиторій:

* файл ***ReadMe.md*** з ***UML*** моделями та текстовим описом;
* програмні проекти із демонстрацією шаблонів;
* проєкт (Project / Item Template) для завдання підвищеної складності.

1. Звіт, що здається без архівації, наприклад: ***РГР 621 П'яточкінПП.docx***

* Титульна сторінка.
* Варіант та постановка задачі.
* Теоретичні відомості (з файлу ReadMe.md, окремо по кожному шаблону) з посиланням на джерело нформації:
* призначення та застосування шаблону проектування.
* опис шаблону проектування ( UML -діаграма).
* опис основних структурних елементів.
* Лістинг програми (кожний шаблон).
* Таблиця з позначкою виконання вимог на роботу (табл.1).

Виконання роботи

Посилання на *GitHub* репозиторій:

[*https://github.com/Yarik-Za/APPZ.Net\_RGR1*](https://github.com/Yarik-Za/APPZ.Net_RGR1)

Вміст файлу *ReadMe.md* з *UML* моделями та текстовим описом

1. **Multiton (Creational Pattern)**

Multiton - це шаблон проектування, який узагальнює шаблон Singleton. Відмінність полягає в тому, що Singleton дозволяє створити лише один екземпляр класу, а Multiton дозволяє контрольовано створювати кілька екземплярів, які він керує за допомогою карти. Цей шаблон спрощує отримання спільних об’єктів в додатку.

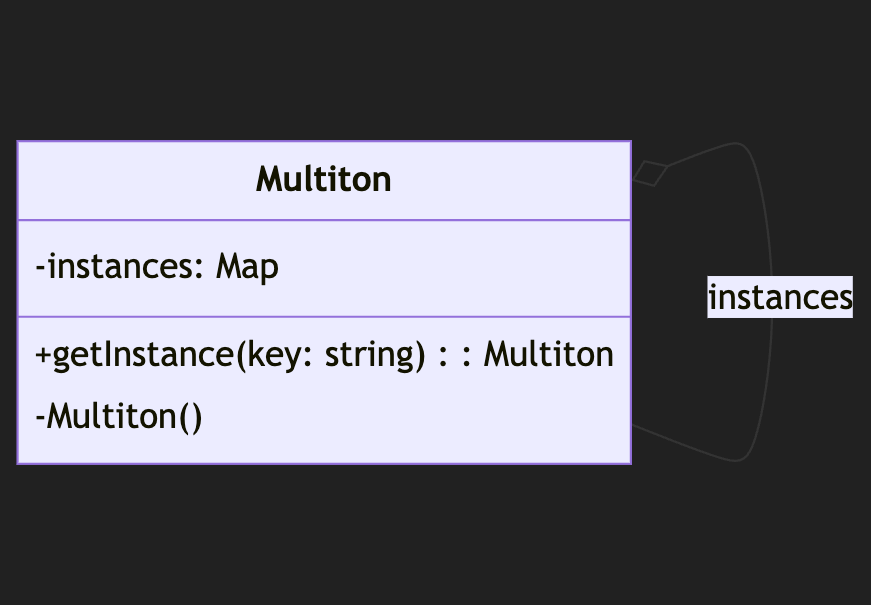


Рисунок 1 – UML діаграма шаблону дизайну Multiton

**Основні складові:**

* Ключ: Ідентифікатор, який використовується для отримання екземпляра Multiton.
* Екземпляр: Об’єкт, який створюється і зберігається в Multiton.
* Карта: Структура даних, яка використовується для зберігання екземплярів Multiton.

**Переваги:** Дозволяє обмежити кількість екземплярів класу, дозволяючи при цьому створювати кілька екземплярів. Забезпечує централізований доступ до одного каталогу multitons, де кожен екземпляр multiton в пулі може існувати, маючи свій власний стан.

**Недоліки:** Цей шаблон, як і шаблон Singleton, ускладнює модульне тестування, оскільки він вводить глобальний стан в додаток. З garbage collected мовами він може стати джерелом витоку пам’яті, оскільки він вводить глобальні сильні посилання на об’єкти.

**Приклади використання:** Multiton - це шаблон проектування, який забезпечує лише один об’єкт, що буде створений для ключа в багатопоточному середовищі.

1. **Adapter або Wrapper (Structural Pattern)**

Adapter - це шаблон проектування, який дозволяє використовувати інтерфейс існуючого класу як інший інтерфейс. Це часто використовується для того, щоб зробити існуючі класи сумісними з іншими без зміни їх вихідного коду5. Цей шаблон дозволяє “перекладати” один інтерфейс на інший.

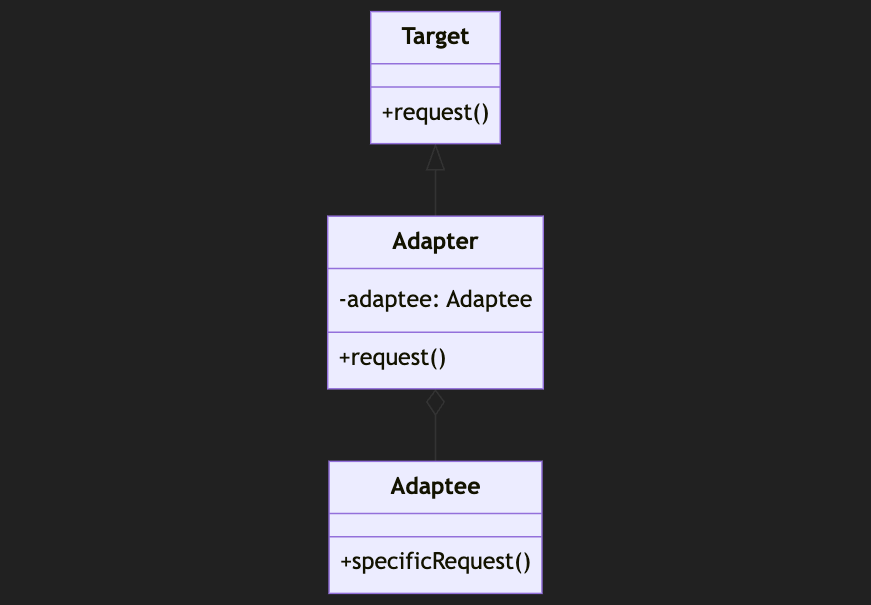


Рисунок 2 – UML діаграма шаблону дизайну Adapter

**Основні складові:**

* Цільовий інтерфейс: Інтерфейс, який очікується клієнтським кодом.
* Adaptee: Існуючий клас або система з несумісним інтерфейсом, який потрібно інтегрувати в нову систему.
* Adapter: Клас, який реалізує цільовий інтерфейс і внутрішньо використовує екземпляр Adaptee, щоб зробити його сумісним з цільовим інтерфейсом.

**Переваги:** Дозволяє використовувати існуючі класи з іншими, не змінюючи їх вихідний код. Адаптери дозволяють вам обмежити кількість екземплярів класу, при цьому дозволяючи створювати кілька екземплярів.

**Недоліки:** Введення адаптерів може додати додатковий шар складності до кодової бази. Використання адаптерів надмірно може призвести до надмірно складної та заплутаної архітектури.

**Приклади використання:** Шаблон адаптера часто використовується в програмних мовах, де він використовується для аналізу та оцінки виразів. Він також використовується в інших програмних додатках, яким потрібна оцінка складних виразів, таких як математичні або логічні вирази.

1. **Interpreter (Behavioral Pattern)**

Interpreter - це шаблон проектування, який визначає спосіб інтерпретації та оцінки мовних граматик або виразів. Він надає механізм для оцінки речень в мові, представляючи їх граматику як набір класів. Кожен клас представляє правило або вираз в граматиці, і шаблон дозволяє ці класи компонувати ієрархічно для інтерпретації складних виразів.

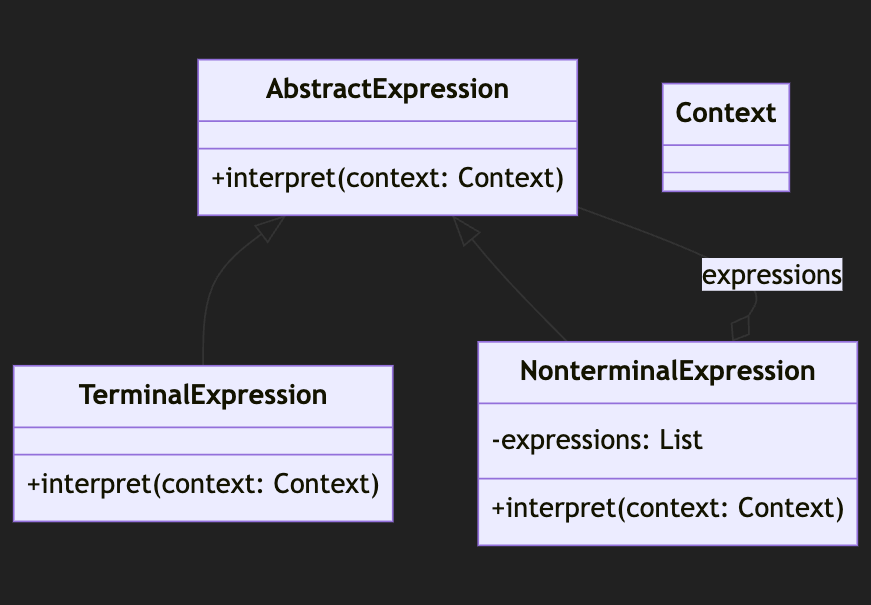


Рисунок 3 – UML діаграма шаблону дизайну Interpreter

**Основні складові:**

* Абстрактний вираз: Це абстрактний клас або інтерфейс, який оголошує абстрактний метод інтерпретації.
* Термінальний вираз: Це конкретні класи, які реалізують інтерфейс абстрактного виразу.
* Нетермінальний вираз: Це також конкретні класи, які реалізують інтерфейс абстрактного виразу.

**Переваги:** Легко змінювати та розширювати граматику. Реалізація граматики є прямолінійною8.

**Недоліки:** Складні граматики важко підтримувати, оскільки буде багато термінальних класів.

**Приклади використання:** Шаблон Interpreter часто використовується в програмних мовах, де він використовується для аналізу та оцінки виразів. Він також використовується в інших програмних додатках, яким потрібна оцінка складних виразів, таких як математичні або логічні вирази.

1. **Lock (Concurrency Pattern)**

Lock - це шаблон паралельних обчислень, який використовується для контролю доступу до спільних ресурсів в базі даних. Цей протокол допомагає підтримувати послідовність даних та цілісність даних серед кількох користувачів.

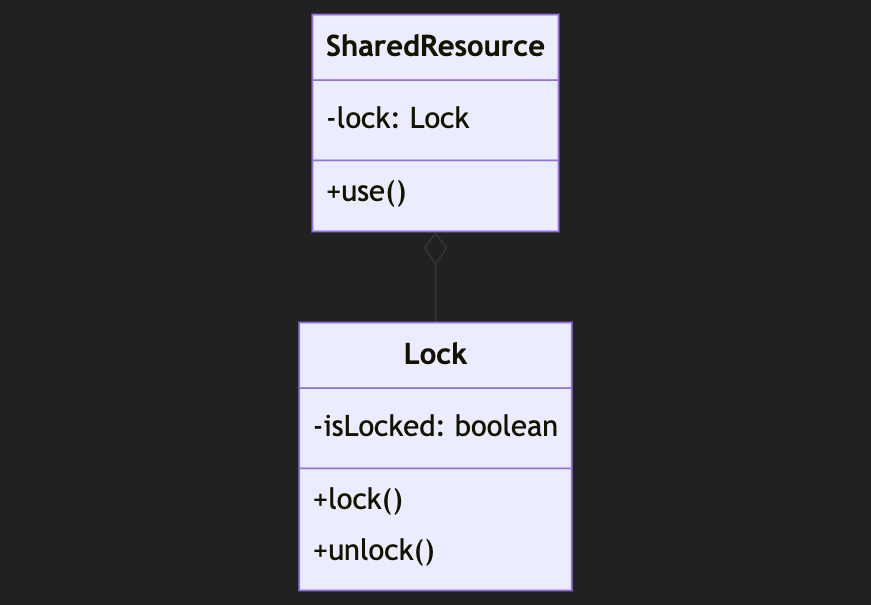


Рисунок 1 – UML діаграма шаблону дизайну Lock

**Основні складові частини Lock:**

* Ключ: Ідентифікатор, який використовується для отримання доступу до ресурсу.
* Ресурс: Об’єкт, до якого потрібно отримати доступ.
* Блокування: Механізм, який використовується для контролю доступу до ресурсу.

**Переваги:** Легко змінювати та розширювати граматику. Реалізація граматики є прямолінійною.

**Недоліки:** Якщо конкуренція даних висока, повторні транзакції можуть призвести до голодування і значно вплинути на продуктивність.

**Приклади використання:** Для надання контролю за конкуренцією та запобігання неконтрольованому доступу до даних, менеджер бази даних ставить блокування на буферні пули.

**Таблиця посилань на першоджерела**

| **Дизайн паттерн** | **Посилання** |
| --- | --- |
| Мультитон | [Medium](https://justgokus.medium.com/what-is-the-multiton-design-pattern-eeeb5dd8bc7d), [Wikipedia](https://en.wikipedia.org/wiki/Multiton_pattern), [Stack Overflow](https://stackoverflow.com/questions/9631087/multiton-real-world-example), [Code Chuckle](https://medium.com/@codechuckle/demystifying-design-patterns-multiton-pattern-859b50638a9f), [Code Project](https://www.codeproject.com/Articles/1178694/Singleton-and-Multiton-Pattern) |
| Адаптер | [DZone](https://dzone.com/articles/adapter-design-pattern-in-java), [Pentalog](https://www.pentalog.com/blog/design-patterns/adapter-design-pattern/), [Stackify](https://stackify.com/design-patterns-explained-adapter-pattern-with-code-examples/), [GeeksforGeeks](https://www.geeksforgeeks.org/adapter-pattern/), [OODesign](https://www.oodesign.com/adapter-pattern), [The Coded Self](https://www.thecodedself.com/The-Difference-Between-an-Adapter-and-a-Wrapper/), [Stack Interface](https://stackinterface.com/adapter-design-pattern/), [FreeCodeCamp](https://www.freecodecamp.org/news/design-pattern-for-modern-backend-development-and-use-cases/) |
| Інтерпретатор | [Visual Paradigm](https://tutorials.visual-paradigm.com/interpreter-pattern-tutorial/), [Simple Tech Talks](https://simpletechtalks.com/interpreter-design-pattern-explained-with-simple-example/), [GeeksforGeeks](https://www.geeksforgeeks.org/interpreter-design-pattern/), [Dot Net Tutorials](https://dotnettutorials.net/lesson/interpreter-design-pattern/), [InfoWorld](https://www.infoworld.com/article/3243279/how-to-work-with-the-interpreter-design-pattern.html), [TutorialsPoint](https://www.tutorialspoint.com/design_pattern/interpreter_pattern.htm) |
| Багаторівневе блокування | [Educative](https://www.educative.io/courses/deep-dive-into-the-internals-of-the-database/optimistic-vs-pessimistic-concurrency-control), [IBM](https://www.ibm.com/docs/en/db2/10.5?topic=management-locks-concurrency-control), [OSTEP](https://pages.cs.wisc.edu/~remzi/OSTEP/threads-locks-usage.pdf), [GeeksforGeeks](https://www.geeksforgeeks.org/two-phase-locking-protocol/), [Wikipedia](https://en.wikipedia.org/wiki/Lock_%28computer_science%29) |
| Модифікація розблокування | [Hacking with Rust](https://www.hackingwithrust.net/2024/01/22/simplifying-concurrency-easy-implementation-of-double-checked-locking-pattern-in-rust/), [Oracle Blogs](https://blogs.oracle.com/maa/post/from-chaos-to-order-the-importance-of-concurrency-control-within-the-database-2-of-6) |

Програмні проєкти із демонстрацією шаблонів

Код програм подано у додатках А-Г.

Успішно створені шаблони були додані в список Item Templates у середовищі розробки. Див. рисунок 5.

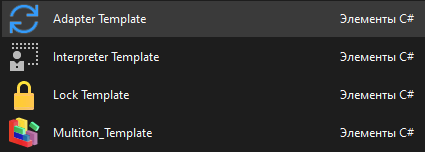


Рисунок 5 – список шаблонів файлів у середовищі розробки

Висновок

У ході виконання завдання було знайдено та описано шаблони проектування кожного типу: породжуючі (Creational), структурні (Structural), поведінкові (Behavioral) та паралельні обчислення (Concurrency). Для кожного з цих шаблонів був створений відповідний текстовий опис та наведено графічне подання у вигляді UML-діаграм, використовуючи інструмент візуалізації Mermaid.

Також було розроблено програмний проект для кожного шаблону проектування з метою продемонструвати його особливості та застосування на практиці. Кожен з цих проектів ілюструє використання відповідного шаблону проектування в реальних ситуаціях розробки програмного забезпечення.

Для завдання підвищеного рівня складності було створено шаблон проектування у вигляді zip-файлу, який може бути використаний як шаблон проекту або елемент (Project / Item Template) для середовища розробки Visual Studio.

*Додаток А*Машинний лістинг файлу **Adapter.cs**

Реалізацція C#:

using System;

public interface IDateParser

{

DateTime ParseDate(string dateString);

}

public class DateAdapter : IDateParser

{

public DateTime ParseDate(string dateString)

{

return DateTime.ParseExact(dateString, "yyyy-MM-dd", null);

}

}

class Program

{

static void Main(string[] args)

{

// Використання адаптера

IDateParser dateParser = new DateAdapter();

DateTime parsedDate = dateParser.ParseDate("2024-03-28");

Console.WriteLine(parsedDate);

Console.ReadKey();

}

}

Реалізацція Kotlin:

import java.time.LocalDate

import java.time.format.DateTimeFormatter

interface DateParser {

fun parseDate(dateString: String): LocalDate

}

class DateAdapter : DateParser {

override fun parseDate(dateString: String): LocalDate {

return LocalDate.parse(dateString, DateTimeFormatter.ofPattern("yyyy-MM-dd"))

}

}

fun main() {

val dateParser: DateParser = DateAdapter()

val parsedDate = dateParser.parseDate("2024-03-20")

println(parsedDate)

}

*Додаток Б*Машинний лістинг файлу **Interpreter.cs**

Реалізацція C#:

using System;

using System.Linq;

public interface IExpression

{

int Interpret();

}

public class NumberExpression : IExpression

{

private int \_number;

public NumberExpression(int number)

{

\_number = number;

}

public int Interpret()

{

return \_number;

}

}

public class AddExpression : IExpression

{

private readonly IExpression \_left;

private readonly IExpression \_right;

public AddExpression(IExpression left, IExpression right)

{

\_left = left;

\_right = right;

}

public int Interpret()

{

return \_left.Interpret() + \_right.Interpret();

}

}

class Program

{

static void Main(string[] args)

{

// Використання інтерпретатора

IExpression expression = new AddExpression(new NumberExpression(10), new NumberExpression(5));

Console.WriteLine(expression.Interpret()); // Результат: 15

Console.ReadKey();

}

}

Реалізацція Kotlin:

interface Expression {

fun interpret(): Int

}

class NumberExpression(private val number: Int) : Expression {

override fun interpret(): Int {

return number

}

}

class AddExpression(private val left: Expression, private val right: Expression) : Expression {

override fun interpret(): Int {

return left.interpret() + right.interpret()

}

}

fun main() {

val expression: Expression = AddExpression(NumberExpression(10), NumberExpression(5))

println(expression.interpret()) // Результат: 15

}

*Додаток В*Машинний лістинг файлу **Multiton.cs**

Реалізацція C#:

using System;

using System.Collections.Generic;

public class Multiton

{

private static readonly Dictionary<string, Multiton> instances = new Dictionary<string, Multiton>();

private Multiton() { }

public static Multiton GetInstance(string key)

{

if (!instances.ContainsKey(key))

{

instances[key] = new Multiton();

}

return instances[key];

}

}

class Program

{

static void Main(string[] args)

{

// Використання Multiton

Multiton instance1 = Multiton.GetInstance("key1");

Multiton instance2 = Multiton.GetInstance("key2");

Multiton instance3 = Multiton.GetInstance("key1");

Multiton instance4 = Multiton.GetInstance("кеу1");

Console.WriteLine(instance1.GetHashCode()); // Виведе хеш першого екземпляру

Console.WriteLine(instance2.GetHashCode()); // Виведе хеш другого екземпляру

Console.WriteLine(instance3.GetHashCode()); // Виведе такий же хеш, як у першого екземпляру (тобто він вже створений)

Console.WriteLine(instance4.GetHashCode());

Console.ReadKey();

}

}

Реалізацція Kotlin:

class Multiton private constructor() {

companion object {

private val instances = mutableMapOf<String, Multiton>()

fun getInstance(key: String): Multiton {

return instances.getOrPut(key) { Multiton() }

}

}

}

fun main() {

val instance1 = Multiton.getInstance("key1")

val instance2 = Multiton.getInstance("key2")

val instance3 = Multiton.getInstance("key1")

println(instance1.hashCode()) // Виведе хеш першого екземпляру

println(instance2.hashCode()) // Виведе хеш другого екземпляру

println(instance3.hashCode()) // Виведе інший хеш, оскільки це новий екземпляр

println(instance1 === instance3) // Поверне true, оскільки це один і той же екземпляр

}

*Додаток Г*Машинний лістинг файлу **Lock.cs**

Реалізацція C#:

using System;

using System.Threading;

public class SharedResource

{

private int \_counter = 0;

private readonly object \_lockObject = new object();

public void Increment()

{

lock (\_lockObject)

{

\_counter++;

}

}

public int GetValue()

{

lock (\_lockObject)

{

return \_counter;

}

}

}

class Program

{

static void Main(string[] args)

{

SharedResource resource = new SharedResource();

// Створюємо два потоки, які збільшують значення ресурсу

Thread thread1 = new Thread(() =>

{

for (int i = 0; i < 10000; i++)

{

resource.Increment();

}

});

Thread thread2 = new Thread(() =>

{

for (int i = 0; i < 10000; i++)

{

resource.Increment();

}

});

thread1.Start();

thread2.Start();

thread1.Join();

thread2.Join();

Console.WriteLine("Final value: " + resource.GetValue());

Console.ReadKey();

}

}

Реалізацція Kotlin:

import java.util.concurrent.locks.ReentrantLock

class SharedResource {

private var counter = 0

private val lock = ReentrantLock()

fun increment() {

lock.lock()

try {

counter++

} finally {

lock.unlock()

}

}

fun getValue(): Int {

lock.lock()

try {

return counter

} finally {

lock.unlock()

}

}

}

fun main() {

val resource = SharedResource()

val thread1 = Thread {

repeat(10000) { resource.increment() }

}

val thread2 = Thread {

repeat(10000) { resource.increment() }

}

thread1.start()

thread2.start()

thread1.join()

thread2.join()

println("Final value: ${resource.getValue()}")

}