ЛАБОРАТОРНА РОБОТА № 4

Створення на аналіз моделей розгортання хмарних рішень «Microsoft Cloud", алгоритмів збору сміття для вивільнення пам’яті та механізмів взаємодії виконуваного та не виконуваного коду

Мета роботи: ознайомитися з концепцією «Microsoft Cloud» та її застосуванням для галузевих рішень. Вивчити різні моделі розгортання хмарних рішень. Зрозуміти принципи роботи збирача сміття (Garbage Collector) в .NET Framework. Навчитися очищати ресурси, що не керуються, у .NET програмах. Ознайомитися з проміжною мовою (Intermediate Language) та процесом виконання коду в .NET. Зрозуміти механізми взаємодії виконуваного та не виконуваного коду.

Завдання: проаналізувати "Microsoft Cloud", моделі розгортання, принцип роботи збору сміття, очищення ресурсів, процес компіляції коду (проміжна мова) та виконання, дослідити процеси взаємодії виконуваного та не виконуваного коду

1. **Огляд "Microsoft Cloud":**
   - описати концепцію "Microsoft Cloud" та її складові (Azure, Microsoft 365, Dynamics 365);
   - пояснити переваги використання хмарних технологій для розробки галузевих рішень;
   - запропонувати приклади використання "Microsoft Cloud" у різних галузях (охорона здоров'я, фінанси, виробництво тощо).
2. **Моделі розгортання:**
   - описати різні моделі розгортання хмарних рішень (публічна хмара (Public Cloud), приватна хмара (Private Cloud) та гібридна хмара (Hybrid Cloud);
   - порівняти ці моделі за різними критеріями (безпека, вартість, масштабованість тощо).
   - запропонувати приклади використання кожної моделі для галузевих рішень.
3. **Збір сміття:**
   - пояснити принцип роботи збирача сміття в .NET Framework.
   - описати алгоритми, які використовуються для збору сміття.
   - Запропонувати приклад у якому зобразити як збирач сміття вивільняє пам'ять від об'єктів, що вже не використовуються програмою.
4. **Очищення ресурсів, що не керуються:**
   - пояснити, які ресурси вважаються такими, що не керуються (наприклад, файли, з'єднання з базами даних, мережеві сокети);

- описати методи очищення таких ресурсів: використання блоку `using`. Реалізація інтерфейсу `IDisposable`. Використання методів `Close` та `Dispose`;
- навести приклади очищення ресурсів різних типів.

5. **Проміжна мова та виконання:**
   - описати процес компіляції коду C# в проміжну мову (Intermediate Language, IL);
   - пояснити роль Common Language Runtime (CLR) у виконанні IL коду;
   - описати процес Just-In-Time (JIT) компіляції IL коду в машинний код.
6. **Взаємодія виконуваного та не виконуваного коду:**
   - пояснити як .NET програми можуть взаємодіяти з кодом, написаним на інших мовах програмування (наприклад, C++);
   - описати механізми Platform Invocation (P/Invoke) та COM Interop для взаємодії з не виконуваним кодом;
   - навести приклади використання цих механізмів для вирішення конкретних завдань.

Приклад виконання лабораторної роботи:

Крок 1. Огляд "Microsoft Cloud".

Опишіть "Microsoft Cloud" та її складові. Наведіть приклади використання Azure для зберігання та обробки медичних даних, Microsoft 365 для спільної роботи над фінансовими документами, Dynamics 365 для управління виробничими процесами.

Крок 2. Моделі розгортання.

Порівняйте моделі розгортання на прикладі медичної інформаційної системи. Публічна хмара може використовуватися для зберігання анонімних статистичних даних, приватна хмара - для зберігання персональних медичних даних пацієнтів, гібридна хмара - для інтеграції локальної медичної системи з хмарними сервісами.

Крок 3. Збір сміття.

Створіть простий клас "Пацієнт" та продемонструйте, як збирач сміття звільняє пам'ять від об'єктів цього класу, коли вони стають недоступними.

```
public class Пацієнт
{
    public string ПІБ { get; set; }
    public int Вік { get; set; }

    public Пацієнт(string піб, int вік)
    {
        ПІБ = піб;
        Вік = вік;
    }
}

// ...

Пацієнт пацієнт1 = new Пацієнт("Іванов Іван Іванович", 30);
пацієнт1 = null; // Об'єкт стає недоступним, збирач сміття звільнить пам'ять
```

Рисунок 4.1 – Приклад класу "Пацієнт" звільнення пам'яті від об'єктів цього класу

Крок 4. Очищення ресурсів, що не керуються.

Напишіть код, який працює з файлом та базою даних, та продемонструйте використання блоку `using` та інтерфейсу `IDisposable` для очищення ресурсів.

```
// Робота з файлом
using (StreamWriter writer = new StreamWriter("data.txt"))
{
    writer.WriteLine("Some data");
} // Файл буде автоматично закрито

// Робота з базою даних
public class DatabaseConnection : IDisposable
{
    // ...

    public void Dispose()
    {
        // Закриття з'єднання з базою даних
    }
}

using (DatabaseConnection connection = new DatabaseConnection())
{
    // ...
```

Рисунок 4.2 – Приклад очищення ресурсів, що не керуються

Крок 5. Проміжна мова та виконання.

Опишіть процес компіляції коду C# в IL та виконання IL коду CLR.

Крок 6. Взаємодія виконуваного та не виконуваного коду.

Наведіть приклад використання P/Invoke для виклику функції з бібліотеки DLL, написаної на C++. Приклад використання P/Invoke для виклику функції з бібліотеки DLL, написаної на C#:

Крок 1. Створення бібліотеки DLL на C++

Створимо просту бібліотеку DLL на C++, яка містить функцію для додавання двох чисел.

```cpp
// MyLibrary.h
#pragma once

extern "C" __declspec(dllexport) int Add(int a, int b);

// MyLibrary.cpp
#include "MyLibrary.h"

int Add(int a, int b)
{
    return a + b;
}
```

Рисунок 4.3 – Приклад створення бібліотеки DLL на C++

Скомпілюйте цей код у DLL-бібліотеку (MyLibrary.dll).

Крок 2. Використання P/Invoke в C#

Створимо консольний застосунок на C#, який буде використовувати P/Invoke для виклику функції Add з DLL-бібліотеки.

```cs
using System;
using System.Runtime.InteropServices;

namespace PInvokeExample
{
    class Program
    {
        // Оголошуємо зовнішню функцію за допомогою DllImport
        [DllImport("MyLibrary.dll", CallingConvention = CallingConvention.Cdecl)]
        public static extern int Add(int a, int b);

        static void Main(string[] args)
        {
            // Викликаємо функцію Add з DLL-бібліотеки
            int result = Add(5, 3);

            // Виводимо результат на консоль
            Console.WriteLine($"Результат додавання: {result}");

            Console.ReadKey();
        }
    }
}
```

Рисунок 4.4 – Приклад використання P/Invoke в C#

Пояснення:

- `DllImport("MyLibrary.dll")` вказує, з якої DLL-бібліотеки імпортується функція.
- `CallingConvention = CallingConvention.Cdecl` вказує на угоду про виклик функції (Cdecl використовується для C++).
- `public static extern int Add(int a, int b)` оголошує зовнішню функцію Add, яка приймає два цілих числа та повертає їх суму.

Запуск програми:

Запустіть консольний застосунок на C#. Він викличе функцію Add з DLL-бібліотеки та виведе результат на консоль.

Важливо:

- DLL-бібліотека (MyLibrary.dll) повинна знаходитися в тому ж каталозі, що й виконуваний файл C# програми, або в одному з каталогів, вказаних у змінній середовища PATH.
- Необхідно правильно вказати ім'я DLL-бібліотеки та угоду про виклик функції в атрибуті `DllImport`.
- Типи даних в C# та C++ повинні відповідати один одному.

Цей приклад демонструє простий спосіб використання P/Invoke для виклику функцій з DLL-бібліотек, написаних на C++. P/Invoke є потужним механізмом, який дозволяє .NET програмам взаємодіяти з кодом, написаним на інших мовах програмування, та використовувати можливості, які недоступні безпосередньо в .NET Framework.
Також запропонований приклад допоможе краще зрозуміти концепцію "Microsoft Cloud" та набути практичних навичок роботи з різними аспектами .NET Framework, включаючи управління пам'яттю, очищення ресурсів та взаємодію з не виконуваним кодом.

Звіт:

Звіт повинен містити:

- короткий опис виконаної роботи;
- код програми з детальними коментарями;
- пояснення щодо роботи з "Microsoft Cloud", моделями розгортання, збирачем сміття, очищенням ресурсів, проміжною мовою та взаємодією з не виконуваним кодом;
- висновки щодо отриманих результатів та набутих навичок.

Контрольні питання:

1. Що таке "Microsoft Cloud" та які її основні складові?
2. Які існують моделі розгортання хмарних рішень?
3. Як працює збирач сміття в .NET Framework?
4. Які методи очищення ресурсів, що не керуються, ви знаєте?
5. Що таке проміжна мова та як вона виконується CLR?
6. Які механізми використовуються для взаємодії виконуваного та не виконуваного коду?