**МОДУЛЬНА РОБОТА № 1**

з дисципліни

«Системне программування»

Шейко Р. О.

Групи КІ-21-2

**Питання 1. (19)**. Серіалізація колекцій об'єктів

Серіалізація колекцій об'єктів в С# - це процес перетворення колекції об'єктів в формат, який може бути збережений або переданий на інший комп'ютер. Зазвичай цей процес використовується для збереження даних у файлі або передачі їх через мережу.

В С# є кілька способів серіалізації колекцій об'єктів. Один з найпоширеніших - використання класу XmlSerializer з простору імен System.Xml.Serialization. Цей клас дозволяє серіалізувати об'єкти в формат XML.

Для серіалізації колекції об'єктів потрібно створити клас, який буде включати в себе цю колекцію. Наприклад, якщо у нас є клас Person і ми хочемо серіалізувати колекцію цих об'єктів, ми можемо створити новий клас People, який буде містити колекцію об'єктів типу Person:

[Serializable]

public class People

{

public List<Person> PersonList { get; set; }

}

Після цього можна створити екземпляр цього класу та заповнити його колекцією об'єктів Person:

People people = new People();

people.PersonList = new List<Person>();

people.PersonList.Add(new Person("John", "Doe", 25));

people.PersonList.Add(new Person("Jane", "Doe", 30));

Для серіалізації цієї колекції об'єктів в формат XML можна використовувати XmlSerializer:

XmlSerializer serializer = new XmlSerializer(typeof(People));

using (TextWriter writer = new StreamWriter("people.xml"))

{

serializer.Serialize(writer, people);

}

Цей код створить файл "people.xml", який містить серіалізовану колекцію об'єктів People.

Окрім XmlSerializer, в С# є також BinaryFormatter, який дозволяє серіалізувати об'єкти в бінарний формат. Цей підхід може бути корисним, якщо необхідно передавати дані через мережу, оскільки бінарний формат зазвичай займає менше місця в порівнянні з форматом XML. Проте BinaryFormatter має кілька обмежень, включаючи невозможність використання з різними версіями програми, а також небезпеку з небезпечними даними.

Узагалі, серіалізація колекцій об'єктів є важливою складовою розробки програмного забезпечення, оскільки дозволяє зберігати та передавати дані між компонентами системи. У С# є кілька способів серіалізації колекцій об'єктів, які можна використовувати в залежності від конкретних потреб додатку.

**Питання 2. (19)**. Обробка виняткових ситуацій. Властивості та методи класу Exception

Обробка виняткових ситуацій є важливою частиною програмування в будь-якій мові, в тому числі і в C#. Ця мова має вбудовану підтримку обробки виняткових ситуацій через клас Exception.

Клас Exception є базовим класом для всіх виключень у C#. Він містить властивості та методи, які дозволяють дізнатись про причину винятку і взаємодіяти з ним.

Одна з основних властивостей класу Exception - Message, яка містить рядок, що пояснює причину винятку. Якщо виняток створено з використанням конструктора, що приймає рядок як параметр, то цей рядок буде збережено у властивості Message.

Крім того, клас Exception містить властивість StackTrace, яка містить інформацію про стек викликів на момент виникнення винятку. Це дозволяє знайти точку в програмі, де сталося виняткова ситуація.

Клас Exception містить також кілька методів, які дозволяють додавати додаткову інформацію про виняток. Метод ToString() повертає рядок, який містить інформацію про виняток, включаючи тип винятку і його повідомлення. Метод GetType() повертає тип винятку. Метод GetBaseException() повертає базовий виняток у випадку, коли виняток було породжено кількома іншими винятками.

Окрім базового класу Exception, у C# є також спеціалізовані класи винятків, які успадковуються від нього. Наприклад, клас DivideByZeroException використовується для обробки випадку, коли в програмі виконується ділення на нуль.

Усі винятки можна обробляти за допомогою оператора try-catch. Він дозволяє відслідковувати винятки, які виникають в певній частині коду, і обробляти їх, не дозволяючи програмі крашитися. Код, який може викликати винятки, розміщується у блокі try, а код, який обробляє винятки, розміщується у блоках catch. У разі виникнення винятку, виконання програми переходить до блоку catch, де можна обробити виняток і вирішити, що робити далі.

Наприклад, наступний код ділить два числа, але враховує випадок ділення на нуль:

int a = 10;

int b = 0;

try

{

int c = a/b;

Console.WriteLine(c);

}

catch (DivideByZeroException e)

{

Console.WriteLine("Помилка: " + e.Message);

}

У цьому прикладі, якщо змінна b дорівнює нулю, то виникне виняток DivideByZeroException, і виконання програми перейде до блоку catch, де виведеться повідомлення про помилку.

Отже, клас Exception та спеціалізовані класи винятків дозволяють дізнатись про причину винятку та взаємодіяти з ним. Оператор try-catch дозволяє обробити винятки та уникнути краху програми. Розуміння обробки виняткових ситуацій є важливим навичкою для будь-якого програміста на мові C#.