.NET Technology

(503112)

By Mai Van Manh

Lab 03: Advanced C# Programming

Mỗi bài học thực hành đều đi kèm với một số tập tin PDF hướng dẫn các kiến thức có liên quan đến các bài tập. Sinh viên cần xem các tập tin hướng dẫn (và video nếu có) trước khi thực hiện các bài tập bên dưới. Sinh viên chỉ nộp lại source code (.cs, .txt; và static contents, media) của từng bài tập, không nộp toàn bộ project. Với bài tập giao diện (Forms, Web) sinh viên cần cung cấp thêm ảnh screenshot từng giao diện của bài tập. Dữ liệu mỗi bài tập cần được đặt trong một thư mục riêng (trừ khi bài tập chỉ có 1 file duy nhất), toàn bộ bài tập cần được đặt trong thư mục có dạng 520H1234_NguyenVanA và được nén lại dưới dạng zip/rar trước khi nộp. Bài nộp không có thông tin sinh viên hoặc nộp toàn bộ source code của project sẽ không được tính.

Exercise 1: Simple Lambda Expressions

Create lambda expressions for basic scenarios:

1. Write a lambda expression that takes two integers and returns their sum.
2. Write a lambda expression that takes a string and returns its length.
3. Write a lambda expression that checks if an integer is even.
4. Write a lambda expression that filters a list of integers to select only even numbers.

Exercise 2: Basic Delegate Usage

Create a simple console application that demonstrates the basic usage of delegates:

1. Define a delegate named MathOperation that takes two int parameters and returns an int.
2. Implement three methods: Add, Subtract, and Multiply, each matching the signature of the MathOperation delegate. These methods should perform the corresponding mathematical operations.
3. In your Main method, create instances of the MathOperation delegate and associate them with the Add, Subtract, and Multiply methods.
4. Use these delegate instances to perform mathematical operations and display the results.

Exercise 3: Basic Multithreading

Create a simple multithreading exercise to understand the basics of creating and managing threads:

1. Write a program that calculates the sum of integers from 1 to N, where N is a large number (e.g., 1 million).
2. Implement two versions of this program: one using a single thread and another using multiple threads. In the version using multithreading, you need to divide the table sum calculation into several parts, each part is processed by one thread.
3. Measure the execution time of both versions and compare the performance.

Exercise 4: Custom Generic List

Create a custom generic list class:

1. Implement a generic class called CustomList<T> that can store elements of any data type.
2. Include methods for adding, removing, and retrieving items from the list.
3. Implement an indexer to access elements by index.
4. Make the list dynamically resizable (e.g., by doubling its size when it reaches capacity).
5. Test your CustomList<T> class by creating instances of it and performing various operations like adding, removing, and retrieving items.

HOMEWORK

Exercise 5: Lambda Expressions with LINQ

Practice using lambda expressions with LINQ:

1. Create a list of custom objects (e.g., Person with properties Name and Age).
2. Use lambda expressions to filter the list and select objects that meet certain criteria (e.g., people older than 30).
3. Use lambda expressions to order the list by a specific property (e.g., sorting people by age).
4. Use lambda expressions to project data, selecting specific properties (e.g., creating a list of names from a list of Person objects).

Exercise 6: Lambda Expressions for Event Handling

Practice using lambda expressions for event handling:

1. Create a custom class with an event (e.g., a Button class with a Click event).
2. Subscribe to the event using a lambda expression instead of a traditional event handler method.
3. Implement event handling logic within the lambda expression.
4. Raise the event and observe the lambda expression-based event handler in action.

Exercise 7: Multicast Delegates

Explore the concept of multicast delegates by performing multiple operations with a single delegate:

1. Create a delegate named StringOperation that takes a string parameter and returns a string.
2. Implement two methods: ToUpper and ToLower, both matching the StringOperation delegate's signature.
3. In your Main method, create an instance of the StringOperation delegate and associate it with the ToUpper method.
4. Use the delegate to convert a string to uppercase.
5. Use the += operator to add the ToL ToLower ower method to the same delegate.

6. Use the delegate again to convert the string to lowercase, and observe how both methods are called in sequence.

Exercise 8: Custom Event Handling

Simulate a custom event handling system using delegates:

1. Define a delegate named EventHandler that takes no parameters and returns void.
2. Create a class called EventManager with the following methods:
    - AddHandler(EventHandler handler): Adds an event handler delegate to a list.
    - RemoveHandler(EventHandler handler): Removes an event handler delegate from the list.
    - RaiseEvent(): Invokes all the event handler delegates in the list.
3. In your Main method, create an instance of the EventManager.
4. Define one or more methods as event handlers and associate them with the EventManager using AddHandler.
5. Call the RaiseEvent method of the EventManager to trigger the event handlers and observe the output.

Exercise 9: Producer-Consumer Problem

Implement the classic producer-consumer problem using threads:

1. Create a program that simulates a producer thread that generates items and a consumer thread that consumes these items.
2. Use a shared buffer (queue) between the producer and consumer threads to exchange data.
3. Implement synchronization mechanisms (e.g., locks, semaphores) to ensure that the producer and consumer threads can work concurrently without conflicts.