Session: 14

Advanced Methods and Types

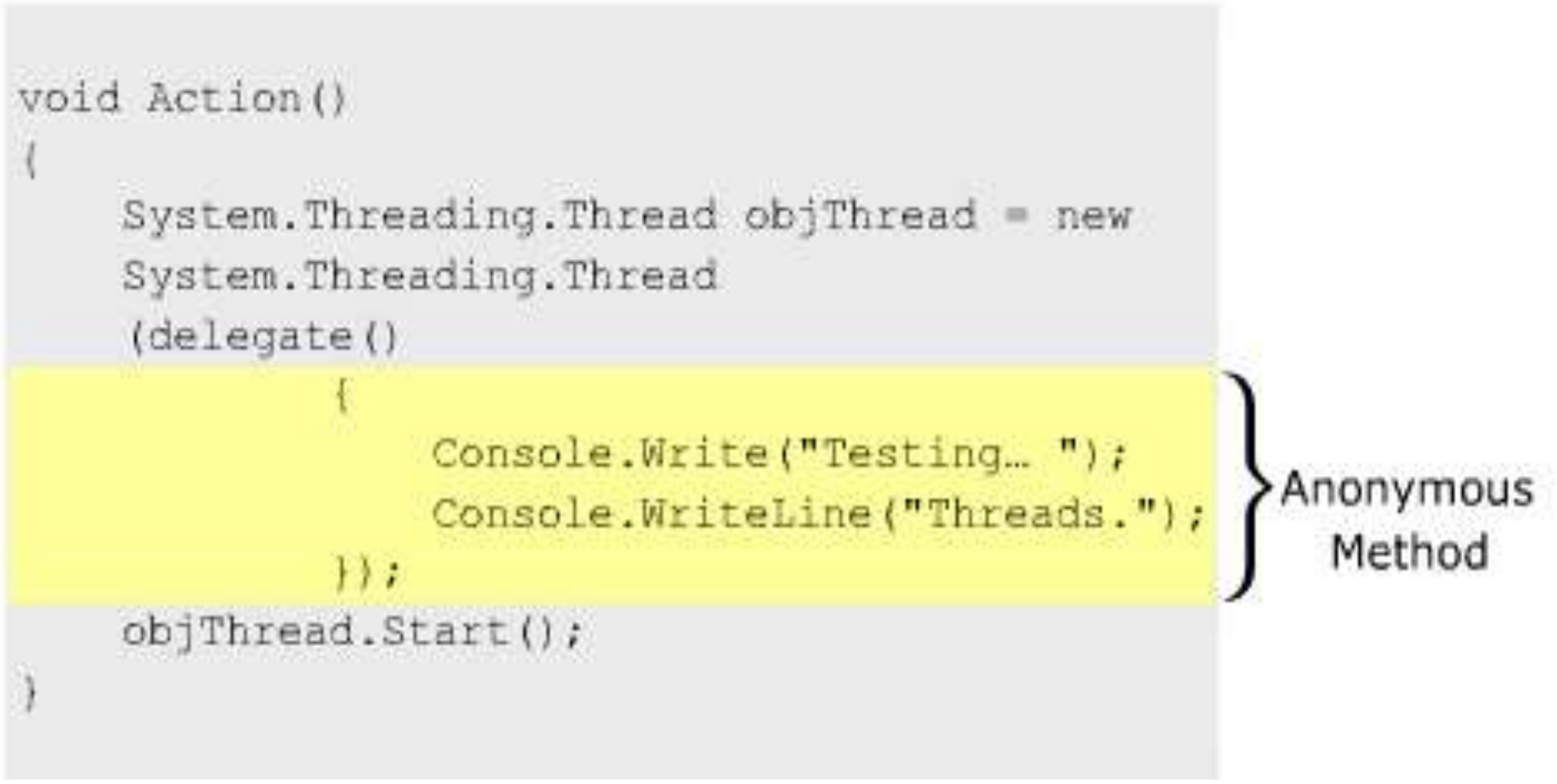
Objectives

- Describe anonymous methods
- Define extension methods
- Explain anonymous types
- Explain partial types
- Explain nullable types

Anonymous Methods

- An anonymous method is an inline nameless block of code that can be passed as a delegate parameter.

Example



```
void Action()
{
    System.Threading.Thread objThread = new
    System.Threading.Thread
    (delegate()
            {
                Console.Write("Testing… ");
                Console.WriteLine("Threads.");
            });
    objThread.Start();
}
```

Creating Anonymous Methods

- Define a delegate :

```
<acc-modifier> delegate <ret-type> Delegate_name(parameters);
```

- Instantiate the delegate by using anonymous method:

```
<Delegate_name> obj = delegate(parameters) {
    /*      .   .   .      */
}
```

Snippet

```
class AnonymousMethods {
   delegate void Display();

   static void Main(string[] args) {
       //using anonymous methods
       Display objDisp = delegate()  {
           Console.WriteLine("This is an anonymous method");
       };
       objDisp();
   }
}
```

Referencing Multiple Anonymous Methods

- C# allows a delegate that can reference multiple anonymous methods.
- The += operator is used to add additional references to either named or anonymous methods after instantiating the delegate.

```
class Program
{
  public delegate void AnoMethod(int a, int b);
  static void Main(string[] args)
  {
    int a = 9, b = 3;
    AnoMethod add ;

    add = delegate(int x, int y)  {
      Console.WriteLine("{0} + {1} = {2}", x, y, x + y);
    };

    add += delegate(int x, int y)  {
      Console.WriteLine("{0} - {1} = {2}", x, y, x - y);
    };
    add(a, b);
  }
```

Extension Methods 1-2

- allow to extend an existing type with new functionality without directly modifying those types.
- are **static** methods that have to be declared in a **static** class.
- declared by specifying the first parameter with the **`this`** keyword, identifies the type of objects in which the method can be called.
- The object that you use to invoke the method is automatically passed as the first parameter.

Syntax

```
static return-type MethodName (this type-obj, param-list)
```

Extension Methods 2-2

- The following code creates an extension method for a string and converts the first character of the string to lowercase:

Snippet

```
using System;
static class ExtensionExample
{
  // Extension Method to convert the 1st char to lowercase
   public static string FirstLetterLower(this string result)
   {
        if (result.Length > 0){
           char[] s = result.ToCharArray();
           s[0] = char.ToLower(s[0]);
           return new string(s);
        }
         return result;
    }
}
```

Anonymous Types

- Anonymous type:
  - Is basically a class with no name and is not explicitly defined in code.
  - Uses object initializers to initialize properties and fields. Since it has no name, you need to declare an implicitly typed variable to refer to it.

Syntax

```
new { identifierA = valueA, identifierB = valueB, … }
```

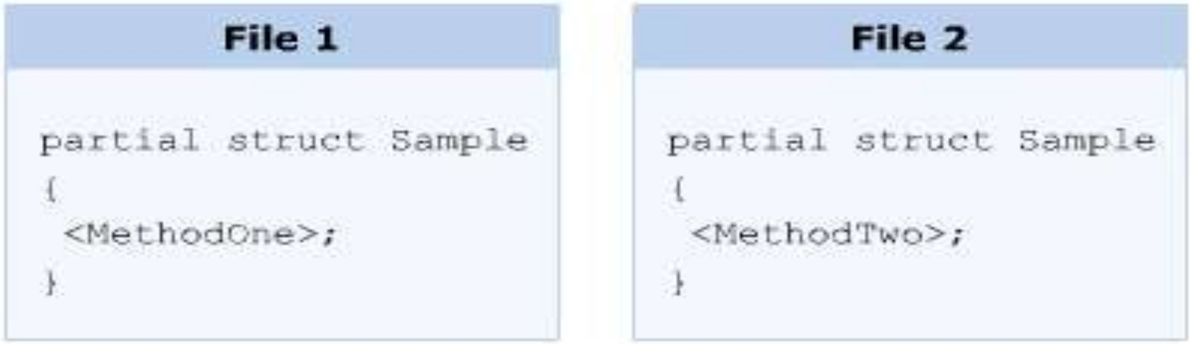
```
using System;
/// <summary>
/// Class AnonymousTypeExample to demonstrate anonymous type
/// </summary>
class AnonymousTypeExample
{
   public static void Main(string[] args)
   {
       // Anonymous Type with three properties.
       var stock = new { Name = "Michgan Enterprises", Code = 1301,
        Price = 35056.75 };
       Console.WriteLine("Stock Name: " + stock.Name);
       Console.WriteLine("Stock Code: " + stock.Code);
       Console.WriteLine("Stock Price: " + stock.Price);
   }
}
```

Using Partial Types

- A large project in an organization involves creation of multiple structures, classes, and interfaces.
- If these types are stored in a single file, their modification and maintenance becomes very difficult.
- In addition, multiple programmers working on the project cannot use the file at the same time for modification.
- Thus, partial types can be used to split a type over separate files, allowing the programmers to work on them simultaneously.
- Partial types are also used with the code generator in Visual Studio 2012.

Features of Partial Types

- facilitates the definition of classes, structures, and interfaces over multiple files.
- Benefits of Partial Type
  - separate the generator code from the application code.
  - help in easier development and maintenance of the code.
  - make the debugging process easier.
  - prevent programmers from accidentally modifying the existing code.
- The following figure displays an example of a partial type:



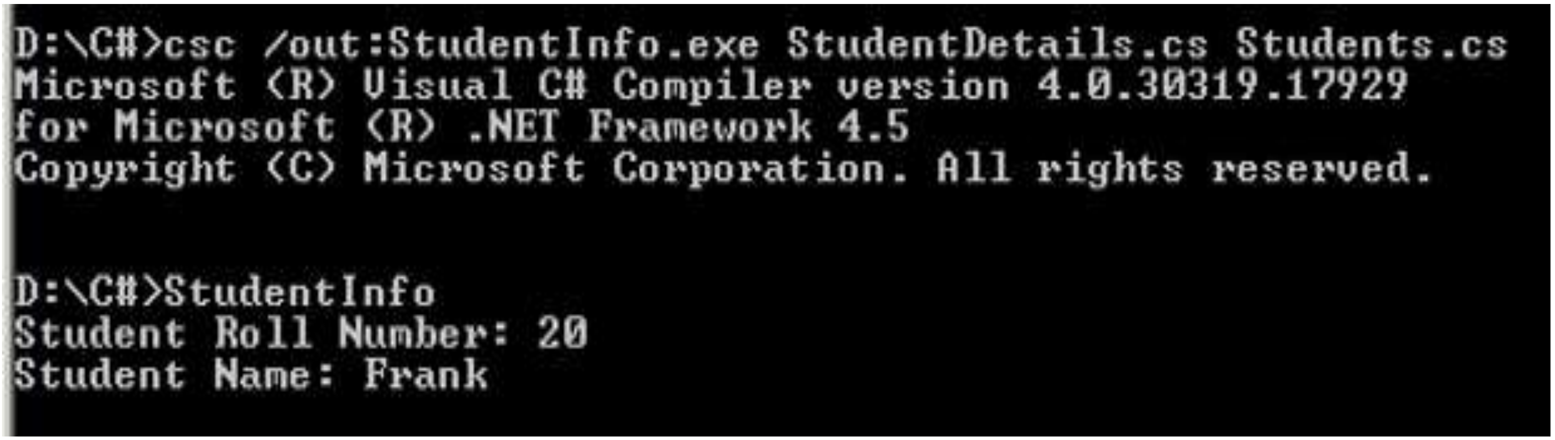
```
partial struct Sample
{
 <MethodOne>;
}
```

```
partial struct Sample
{
 <MethodTwo>;
}
```

Merged Elements during Compilation

- The members of partial classes, partial structures, partial interfaces declared & stored at different locations are combined together at the time of compilation.
- These members can include:
  - XML comments & Interfaces
  - Generic-type parameters & Class variables
  - Local variables & Methods
  - Properties
- A partial type can be compiled at the Developer Command Prompt for VS2012. The command to compile a partial type is:

```
csc /out:<FileName>.exe <CSFileName1>.cs <CSFileName2>.cs
```



```
D:\C#>csc /out:StudentInfo.exe StudentDetails.cs Students.cs
Microsoft (R) Visual C# Compiler version 4.0.30319.17929
for Microsoft (R) .NET Framework 4.5
Copyright (C) Microsoft Corporation. All rights reserved.


D:\C#>StudentInfo
Student Roll Number: 20
Student Name: Frank
```

Partial Methods

- is a method whose signature is included in a partial type.
- may be optionally implemented in another part of the partial class or type or same part of the class or type.

Snippet

```
namespace PartialTest
{
/// <summary>
/// Class Shape is a partial class and defines a partial method.
/// </summary>

    public partial class Shape
    {
        partial void Create();
    }
}
```

Implementing Nullable Types 1-2

- A nullable type can include any range of values that is valid for the data type to which the nullable type belongs.
- For example, a bool type that is declared as a nullable type can be assigned the values true, false, or null.
- Nullable types have two public read-only properties that can be implemented to check the validity of nullable types and to retrieve their values.

  These are as follows:

  - **The `HasValue` property**: `HasValue` is a `bool` property that determines validity of the value in a variable.
    The `HasValue` property returns a true if the value of the variable is `not null`, else it returns false.
  - **The `Value` property:** The `Value` property identifies the value in a nullable variable. When the `HasValue` evaluates to true, the `Value` property returns the value of the variable, otherwise it returns an exception.

Implementing Nullable Types 2-2

- The following code displays the employee’s name, ID, and role using the nullable types:

Snippet

```
using System;
class Employee {
    static void Main(string[] args)  {
      int empId = 10;
      string empName = “Patrick”;
      char? role = null;
      Console.WriteLine(“Employee ID: “ + empId);
      Console.WriteLine(“Employee Name: “ + empName);
      if (role.HasValue == true) {
        Console.WriteLine(“Role: “ + role.Value);
      }
      else {
        Console.WriteLine(“Role: null”);
      }
    }
}
```

The ?? Operator

- When a nullable type contains a null value and you assign this nullable type to a non-nullable type, the complier generates an exception called `System.InvalidOperationException`.
- To avoid this problem, you can specify a default value for the nullable type that can be assigned to a non-nullable type by using the `??` operator.
- If the nullable type contains a null value, the `??` operator returns the default value.
- The following code demonstrates the use of `??` operator:

Snippet

```
using System;
class Salary {
   static void Main(string[] args) {
         double? actualValue = null;
         double marketValue = actualValue ?? 0.0;
         actualValue = 100.20;
         Console.WriteLine("Value: " + actualValue);
         Console.WriteLine("Market Value: " + marketValue);
   }
}
```

Summary

- Anonymous methods allow you to pass a block of unnamed code as a parameter to a delegate.
- Extension methods allow you to extend different types with additional static methods.
- You can create an instance of a class without having to write code for the class beforehand by using a new feature called anonymous types.
- Partial types allow you to split the definitions of classes, structs, and interfaces to store them in different C# files.
- You can define partial types using the partial keyword.
- Nullable types allow you to assign null values to the value types.
- Nullable types provide two public read-only properties, HasValue and Value.