Microsoft

LESSON 2.6

10754 Microsoft .NET Fundamentals

# Understand Generics

LESSON 2.6

10754 Microsoft .NET Fundamentals

# Lesson Overview

Understand generics.

In this lesson, you will:

- Learn about generics terminology.
- Learn how generics help manage collections.
- Use generics in a simple application.

# Guiding Questions

1. What programming needs do generics fill?
2. What is a type parameter, and how is it used?
3. How are generic collections used in the Microsoft® .NET Framework?

# Anticipatory Set

Consider the following code:

```
int i = 10;
float f = 1.5F;
i = (int)f;
```

With a partner, discuss the value of *i* after the code executes—and why that is the value.

# Generics

- Generics are types that let you tailor a method, class, structure, or interface to the precise data type it acts upon.
- Although generics have other uses, they are used most commonly in collections.
  - In collections, using generics eliminates the need to cast objects prior to processing or using them.
- In short, a generic type is a type that uses *generic type parameters*.

# The Need for Generics

- Collection classes such as `ArrayList` are *untyped*, meaning that they can contain any object derived from `System.Object`—which is to say, almost anything.
- However, the flexibility of an untyped collection is offset by a big limitation: The contents of the collection must be cast to the type that you wish to use.
  - Developers then have to take steps to ensure that only the correct objects get stored in the object; even then, it may be cumbersome to use the objects in the collection.
  - Some developers choose to implement their own *strongly typed* collection methods, but this requires additional coding.
- Generics allow developers to specify types when objects are created.

# Generic Type Parameters

- A generic type parameter is the placeholder or template that a generic type uses.
- Generic type parameters are specified with angle brackets ( < >)
- We will look at the `List` class, which is the generic equivalent `ArrayList`. To declare a `List` object, use a generic type parameter as follows:

```
List<String> myListOfStrings;
```

- In this example, the generic type parameter indicates that the collection will store `String` objects.
- Objects in this `List` can be handled as `Strings` without the need for casting. Note that it uses the same index syntax as arrays:

```
Console.WriteLine(myListOfStrings[2].Substring(2,3));
```

# Using Generics

1. Create a Windows Forms project with a `Button` (named `btnTest`) and a `Label` (`lblOutput`); create a `Click` event handler for `btnTest`.
2. In the event handler, add the following code:

```
List<String> myList = new List<String>();

myList.Add("Microsoft");
myList.Add("Visual");
myList.Add("Studio");

lblOutput.Text = myList[2];
```

3. Run the application and click the `Button`. The `Label` should display "Studio."

# Lesson Review

- Create the example application from the instructions available at ***http://msdn.microsoft.com/en-us/library/6sh2ey19.aspx***.
- This application is a Console application, not a Windows Forms application.
- Use CTRL-F5 or "Run Without Debugging" to keep the command prompt window open.