Microsoft

LESSON 2.2

10754 Microsoft .NET Fundamentals

Understand Object-Oriented Concepts in the .NET Framework

LESSON 2.2

10754 Microsoft .NET Fundamentals

Lesson Overview

How does object-oriented programming (OOP) work within the Microsoft® .NET Framework?

In this lesson, you will explore:

- Using inheritance to extend classes
- Polymorphism
- Using interfaces in application design

Guiding Questions

1. How does inheritance work within the .NET Framework?
2. What are the advantages of using polymorphism?
3. What is the role of interfaces in object-oriented design?

Anticipatory Set

1. Describe one of your favorite video games.
2. What are the basic objects that are part of the game?
3. For each object, list a few actions that that object does in the game.

Object-Oriented Programming (OOP)

- OOP is a programming paradigm or model in which classes and objects work together to create a program or application.
- It enables developers to think about a problem in a way that is similar to how we look at the real world—many objects that interact with each other.
- The fundamental building blocks of object-oriented programs are **classes** and **objects**.

Classes and Objects

- Classes are like blueprints or recipes for objects.
  - You can create multiple houses from one blueprint and multiple batches of cookies from one recipe.
  - Each house (or each batch of cookies) is referred to as an **instance**.
- Classes and objects are not the same thing; a class is not an object, just as a recipe is not a batch of cookies.

Inheritance

- Inheritance is a technique in which a class *inherits* the attributes and behaviors of another class.
- Allows the developer to reuse, extend, and modify an existing class.

Example:

- The *Aircraft* class might define the basic characteristics and behaviors of a flying vehicle. Methods in that class might include *Fly(), TakeOff(),* and *Land()*.
- The *Helicopter* class could add a method called *Hover(),* which might replace or "override" the *TakeOff()* method so that takeoffs are vertical.

Inheritance Vocabulary

- The class whose members are inherited is called the **base class.**
- The class that inherits those members is called the **derived class.**
  - In our previous example, *Aircraft* is the base class; *Helicopter* is the derived class.
- Some developers refer to a base class as a *superclass* or a *parent class*; a derived class could then be called a *subclass* or a *child class*.
- We can test the validity of an inheritance relationship with the so-called "Is-A" test. "A helicopter **is an** aircraft," so that is an appropriate use of inheritance.
  - However, the *Car* class should not inherit from *Aircraft* because "a car **is an** aircraft" doesn't make sense.

Inheritance in .NET Languages

- In Microsoft C#, inheritance is established using the colon (:)

```
public class Helicopter : Aircraft
{
}
```

- In Microsoft Visual Basic®, inheritance is indicated with the `Inherits` keyword:

```
Public Class Helicopter Inherits Aircraft
End Class
```

Polymorphism

- Polymorphism is a language feature that allows a derived class to be used interchangeably with its base class.
  - In our example, a *Helicopter* can be treated like an *Aircraft* when the developer writes code. If you developed a *JumboJet* class, it could also be used like an *Aircraft*.
- Polymorphism is especially useful when working with collections, such as lists or arrays. A collection of *Aircraft* could be populated with *Helicopter* instances and *JumboJet* instances, making the code much more flexible.

Interfaces

- An interface defines a set of properties, methods, and events, but it does *not* provide any implementation.
- It is essentially a contract that dictates how any class implementing the interface must be implemented.
  - Consider a power outlet in the wall, which is similar to an interface. It dictates that any appliance that wants to use that power must follow some basic rules: how many prongs are required, the shape and spacing of those prongs, and so on.
- Any class that implements an interface but does not include the specified members (properties, methods, and events) will not compile.
- An object cannot be instantiated ("constructed") from an interface.

Lesson Review

1. Explain the relationship between a base class and a derived class.

2. Why do developers use polymorphism?

3. How is an interface like a contract?