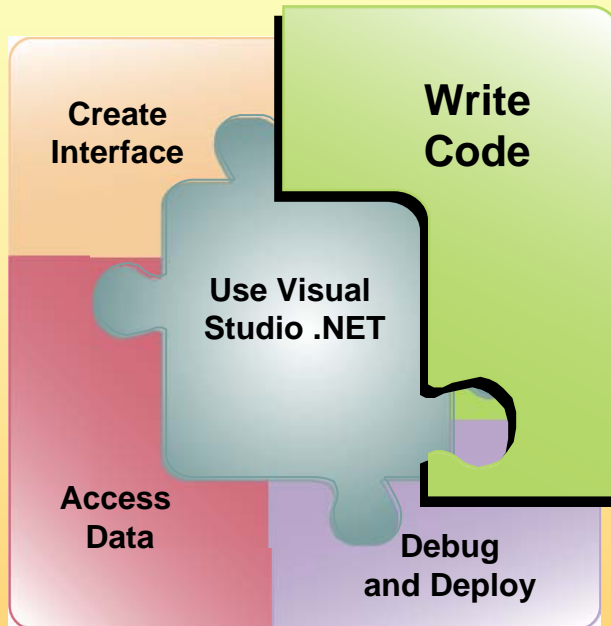


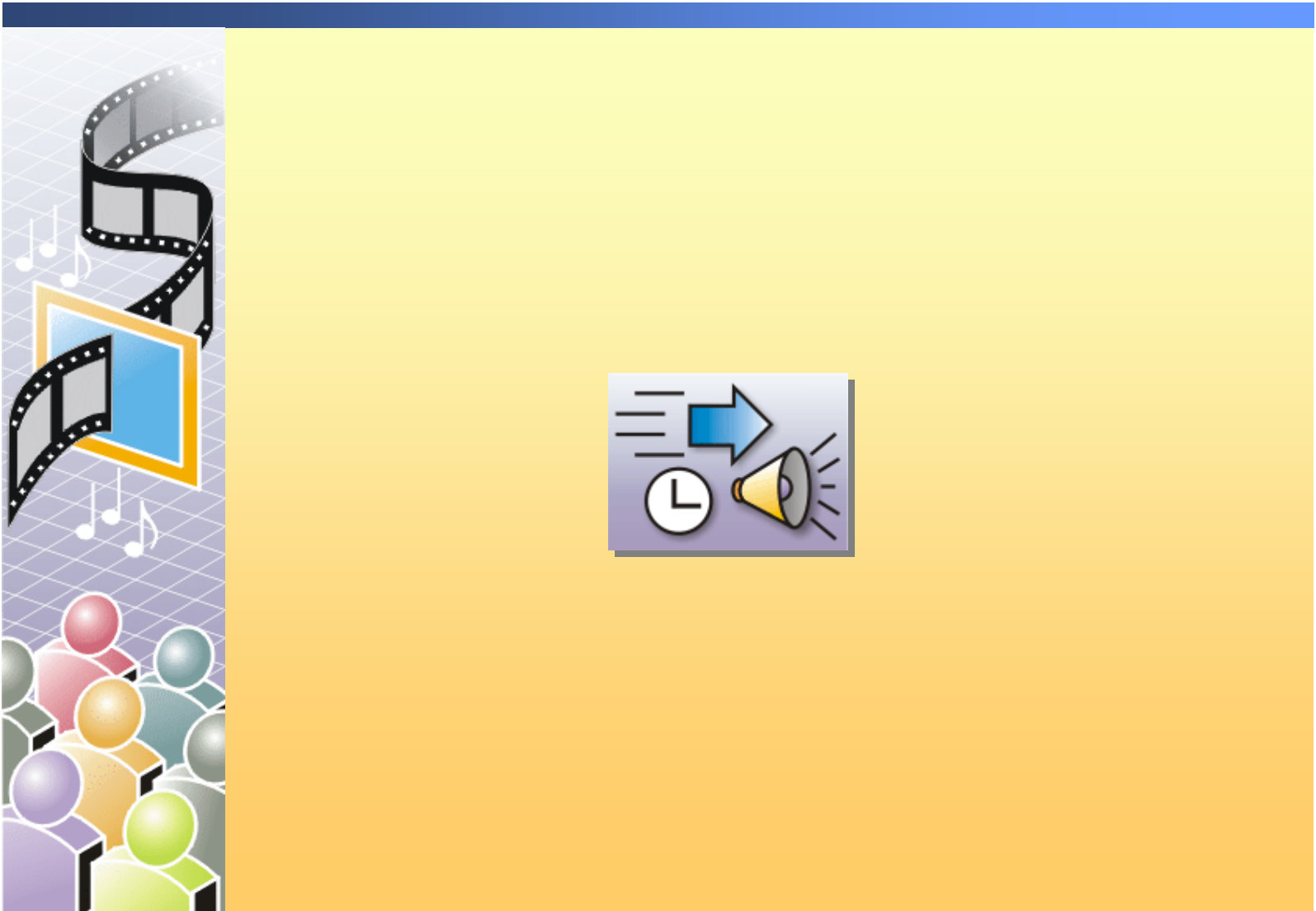
**Module 7: Object-
Oriented Programming
in Visual Basic .NET**

Overview



- Understanding Classes
- Working with Classes
- Using Shared Members
- Inheritance, Polymorphism, and Namespaces

Multimedia: Introduction to Object-Oriented Concepts



Lesson: Understanding Classes

- abstraction
- class
- encapsulation
- object

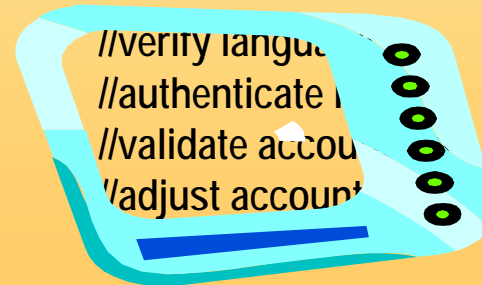
What Is a Class?

- A *class* is a blueprint that describes an object and defines attributes and operations for the object
- Classes use *abstraction* to make available only the elements essential to defining the object
- Classes use *encapsulation* to enforce an abstraction

What the user sees:

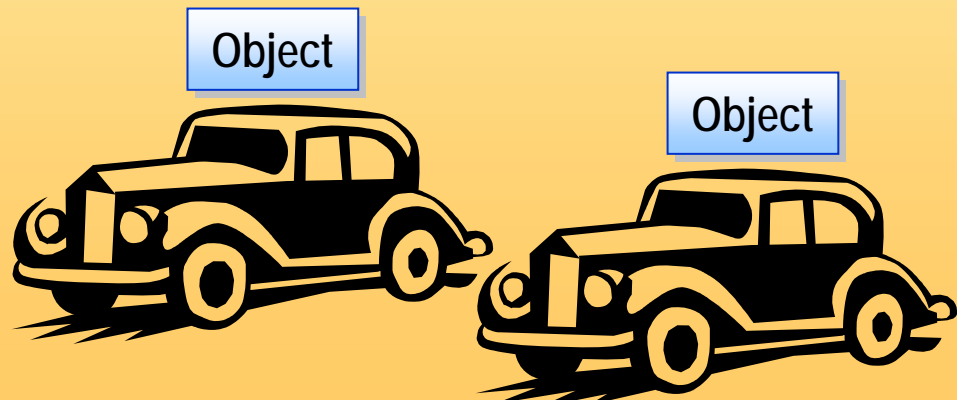
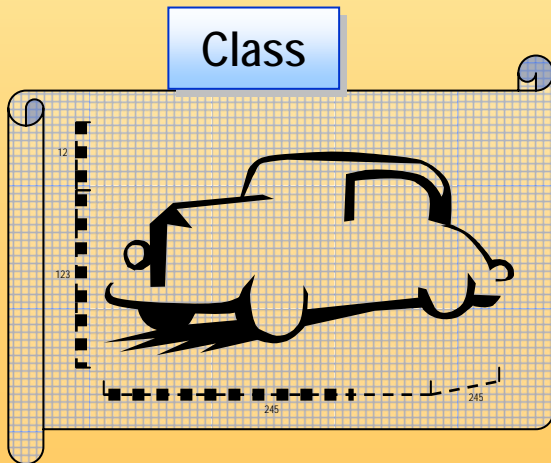


What is encapsulated:



What Is an Object?

- An object is an instance of a class
- Objects have the following qualities:
 - Identity: Objects are distinguishable from one another
 - Behavior: Objects can perform tasks
 - State: Objects store information that can vary over time



How to Use the Object Browser

The image shows the Visual Studio Object Browser window. It is divided into three main sections:

- Objects Pane:** Located on the left, it displays a tree view of the project's components. The `AccessibleObject` class is selected and highlighted in blue.
- Members Pane:** Located on the right, it displays the members of the selected `AccessibleObject` class. The members listed are:
 - `DoDefaultAction()`
 - `GetChildCount()`
 - `GetChild(Integer)`
 - `GetFocused()`
 - `GetHelpTopic(String)`
 - `GetSelected()`
 - `HitTest(Integer, Integer)`
 - `Navigate(System.Windows.Forms.Ac...`
 - `New()`
- Description Pane:** Located at the bottom, it provides details about the selected class. It shows the class signature `<System.Runtime.InteropServices.ComVisibleAttribute(True)>`, the class name `Public Class AccessibleObject`, its inheritance from `System.MarshalByRefObject`, and its membership in `System.Windows.Forms`. A **Summary:** section follows, stating: "Provides information that accessibility applications use to adjust an application's UI for users with impairments."

Three callout boxes with pink arrows point to these sections: "Objects Pane" points to the tree view, "Members Pane" points to the list of methods, and "Description Pane" points to the class details and summary.

Lesson: Working with Classes

- How to Create a New Class
- How to Add Instance Data Members
- How to Add Methods
- How to Add Properties
- How to Create an Instance of a Class
- How to Use Constructors
- How to Use Destructors

How to Create a New Class

- Create a new class by using the Add Class command on the Project menu
- Example of new class named BankAccount:

```
Public Class BankAccount  
  
End Class
```

How to Add Instance Data Members

- Adding a data member named *balance*

```
Public Class BankAccount
    Private balance As Double
End Class
```

Keyword	Definition
Public	Accessible everywhere
Private	Accessible only within the type itself
Protected	Accessible only by classes which inherit from the class

How to Add Methods

- Adding a method named Deposit

```
Public Class BankAccount  
  
    Private balance As Double  
  
    Public Sub Deposit(ByVal amount As Double)  
        balance += amount  
    End Sub  
  
End Class
```

- Overloaded methods: two or more methods with the same name but different signatures
Example: `MessageBox.Show`

How to Add Properties

- Adding a property:

```
Public Class BankAccount
    Private customerName As String

    Public Property Name( ) As String
        Get
            Return customerName
        End Get
        Set(ByVal Value As String)
            customerName = Value
        End Set
    End Property

End Class
```

How to Create an Instance of a Class

- Using the New keyword to create an instance of the BankAccount class:

```
Module Bank

    Sub Main
        Dim account As New BankAccount( )
        account.Deposit(500.00)
    End Sub

End Module
```

Practice: Creating a Class



- In this practice, you will create a `BankAccount` class with methods and properties

How to Use Constructors

- Executes code when object is instantiated

```
Public Sub New( )  
    ' Perform simple initialization  
    value = 1  
End Sub
```

- Can overload, but does not use **Overloads** keyword

```
Public Sub New(ByVal i As Integer)  
    ' Overloaded without Overloads keyword  
    ' Perform more complex initialization  
    value = i  
End Sub
```

How to Use Destructors

- Used for resource cleanup
- Called by runtime before destroying object
 - Important: Destruction might not happen immediately

```
Protected Overrides Sub Finalize( )
```

```
    ' Can close connections or other resources
```

```
    conn.Close
```

```
End Sub
```

Lesson: Using Shared Members

- How to Use Shared Data Members
- How to Use Shared Methods

How to Use Shared Data Members

- Shared data members allow multiple class instances to refer to a single class-level variable

```
Class SavingsAccount
  Public Shared InterestRate As Double
  Public Name As String, Balance As Double
  . . .
End Class
```

```
SavingsAccount.InterestRate = 0.03
```

How to Use Shared Methods

- Can be used without declaring a class instance
- Can only access shared data

```
' TestClass code  
Public Shared Function GetComputerName( ) As String  
    ...  
End Function
```

```
' Client code  
MessageBox.Show(TestClass.GetComputerName( ))
```

Practice: Creating Shared Methods



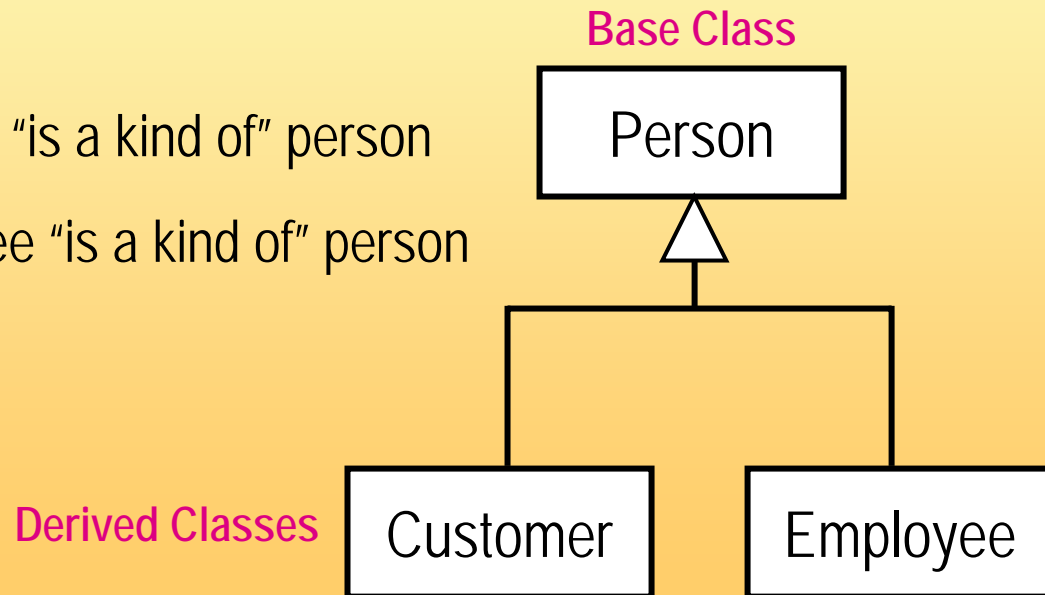
- In this practice, you will:
 - Create a class
 - Add shared methods
 - Use shared methods

Lesson: Inheritance, Polymorphism, and Namespaces

- inheritance
- polymorphism
- structures and classes
- namespaces

What Is Inheritance?

- Inheritance specifies an “is-a-kind-of” relationship
- Multiple classes share the same attributes and operations, allowing efficient code reuse
- Examples:
 - A customer “is a kind of” person
 - An employee “is a kind of” person

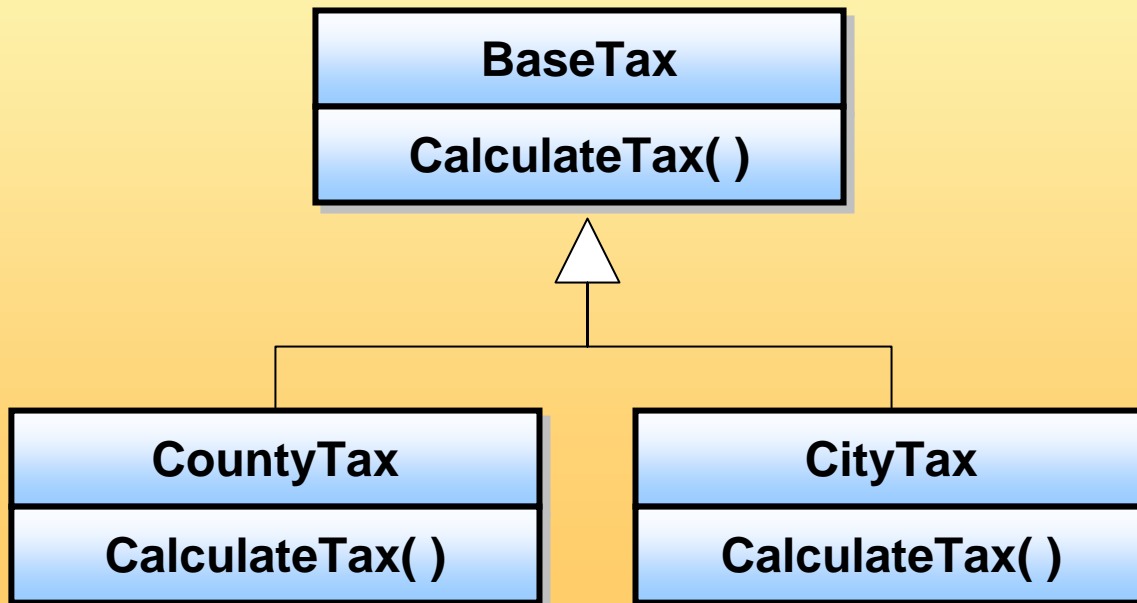


How to Inherit from a Class

- A derived class inherits from a base class
- Properties, methods, data members, events, and event handlers can be inherited (dependent on scope)
- Keywords
 - **Inherits** – inherits from a base class
 - **NotInheritable** – cannot be inherited from
 - **MustInherit** – instances of the class cannot be created; must be inherited from as a base class

What Is Polymorphism?

- The method name resides in the base class
- The method implementations reside in the derived classes



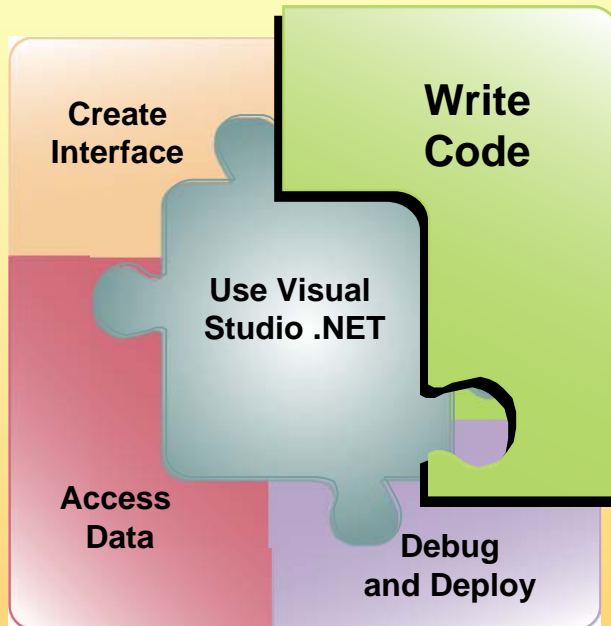
Comparing Classes to Structures

Classes	Structures
Can define data members, properties, and methods	Can define data members, properties, and methods
Support constructors and member initialization	No default constructor or member initialization
Support Finalize method	Do not support Finalize method
Extensible by inheritance	Do not support inheritance
Reference type	Value type

How to Organize Classes into Namespaces

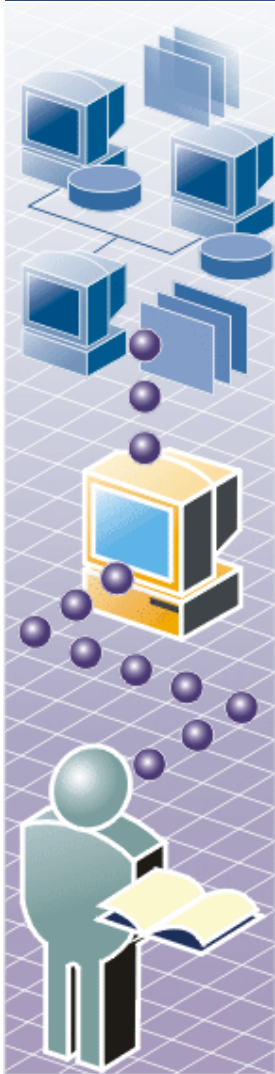
- Namespaces are an organizational system
- Namespaces provide fully qualified names for classes
 - Example: `System.Windows.Forms.Button`
- To import a namespace:
 - At the project level, add a reference to the DLL that contains the namespace
 - Use the **Imports** keyword

Review



- Understanding Classes
- Working with Classes
- Using Shared Members
- Inheritance, Polymorphism, and Namespaces

Lab 7.1: Creating a Derived Class



- Exercise 1: Creating a Derived Form Class