SQL

SELECT fields

FROM table

WHERE search condition

GROUP BY grouping_columns

HAVING search_condition

ORDER BY sort_fields

SQL

SELECT *

SELECT Architecture

SELECT Architecture, Agriculture, Social

SQL

```
FROM Education
FROM Education, Lookup
FROM Cities, Countries
FROM Cities INNER JOIN Countries ON
Cities.country = Countries.country
```

SQL

WHERE country = ‘India’

WHERE pop2015 >= 20

WHERE city **LIKE** ‘D%’

WHERE Information < 400

WHERE monetaryUnit **LIKE** ‘_U%’

SQL

ORDER BY city

ORDER BY city **ASC**

ORDER BY city **DESC**

ORDER BY country, city **ASC**

SQL

```
SELECT * FROM Cities ORDER BY city ASC
```

```
SELECT city, monetaryUnit FROM Cities INNER
JOIN Countries
ON Cities.country = Countries.country
```

```
SELECT * FROM Cities WHERE country = ‘India’
```

```
SELECT * FROM Cities WHERE city LIKE ‘D%’
```

```
SELECT * FROM Cities WHERE pop2015 >= 20
```

Chapter 11

Chapter 11 – Object-Oriented Programming

11.1 Classes and Objects

11.2 Arrays of Objects; Events; Containment

11.3 Inheritance

What is Object Oriented Programming?

An object is like a black box.

The internal details are hidden.

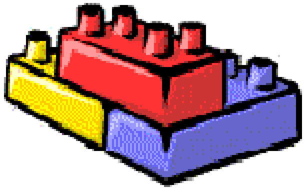
- Identifying *objects* and assigning *responsibilities* to these objects.
- Objects communicate to other objects by sending *messages*.
- Messages are received by the *methods* of an object

What is an object?

- Tangible Things as a car, printer, ...
- Roles as employee, boss, ...
- Incidents as flight, overflow, ...
- Interactions as contract, sale, ...
- Specifications as colour, shape, ...

WHY DO WE CARE ABOUT OBJECTS?

- Modularity - large software projects can be split up in smaller pieces.
- Reuseability - Programs can be assembled from pre-written software components.
- Extensibility - New software components can be written or developed from existing ones.



11.1 Classes and Objects

- **noun** A word used to denote or name a person, place, thing, quality, or act.

- **verb** That part of speech that expresses existence, action, or occurrence.

- **adjective** Any of a class of words used to modify a noun or other substantive by limiting, qualifying, or specifying.
  - *The American Heritage Dictionary of the English Language*

OOP Analogy

Classes → **noun** A word used to denote or name a person, place, thing, quality, or act.

Methods → **verb** That part of speech that expresses existence, action, or occurrence.

Properties → **adjective** Any of a class of words used to modify a noun or other substantive by limiting, qualifying, or specifying.

OOP Terminology

- An *object* is an *encapsulation* of data and procedures that act on that data

- *"data hiding"* prevents inadvertent data modification

Built In Objects

- *Control objects* – text boxes, list boxes, buttons, etc

- To create an *instance* of a control object, double-click on that control in the tool box.

- The control in the tool box is a template or blueprint of that control.

- You cannot set properties or invoke methods until you create an instance.

User Defined Objects

- *Code objects* – a specific instance of a user defined type called a *class*

```
Class ClassName
   statements
End Class
```

The statements define the properties, methods, and events for the class

- The user defined type represents the template or blueprint for the code object

- This user defined type is called a *class*

Instantiating a Code Object

- An object of a class can be declared with the statements:

```
Dim objectName As className
objectName = New className(arg1, arg2, ...)
```

  where the second statement must appear inside a procedure.

- The Dim statement sets up a reference to the new object.
- The object is actually created with the word *New*.

Instantiating a Code Object

- The pair of statements from the previous slide can be replaced with the following single statement, which can appear anywhere in a program.

```
Dim objectName As New
  className(arg1,arg2,...)
```

Common Tasks

Task	**Statement**
Assign a value to a property	***objectName.propertyName*** = ***value***
Assign the value of a property to a variable	***varName*** = ***objectName.propertyName***
Carry out a method	***objectName.methodName*(*arg1*, ...)**
Raise an event	**RaiseEvent *eventName***

Private Data

- Classes contain variables, called *member* or *instance* variables that are declared with a statement of the form

```
Private m_name As String
```

- The word "Private" is used to ensure that the variable cannot be accessed directly from outside the class

- Values are not assigned to or read from member variables directly, but rather through property blocks

Get and Set

Property block

```
Private m_name As String

Public Property Name() As String
  Get
    Return m_name
  End Get
  Set(ByVal value As String)
    m_name = value
  End Set
End Property
```

Public vs. Private

- Items declared with the keyword Private (instead of Dim) cannot be accessed from outside the class.

- Those declared as Public are accessible from both inside and outside the class.

Student Class: Member Variables

```
Private m_name As String
Private m_ssn As String
Private m_midterm As Double
Private m_final As Double
```

Student Class: Property Blocks

```
Public Property Name() As String
  Get
    Return m_name
  End Get
  Set(ByVal value As String)
    m_name = value
  End Set
End Property
```

Student Class: Property Blocks

```
Public Property SocSecNum() As String
  Get
    Return m_ssn
  End Get
  Set(ByVal value As String)
    m_ssn = value
  End Set
End Property
```

Student Class: WriteOnly Property Blocks

```
Public WriteOnly Property Midterm() As Double
  Set(ByVal value As String)
    m_midterm = value
  End Set
End Property

Public WriteOnly Property Final() As Double
  Set(ByVal value As String)
    m_final = value
  End Set
End Property
```

Two Notes

- ***Note 1:*** The last two Property blocks were WriteOnly. We will soon see why. A property block also can be specified as ReadOnly. If so, it consists only of a Get procedure

- ***Note 2:*** Methods are constructed with Sub and Function procedures.

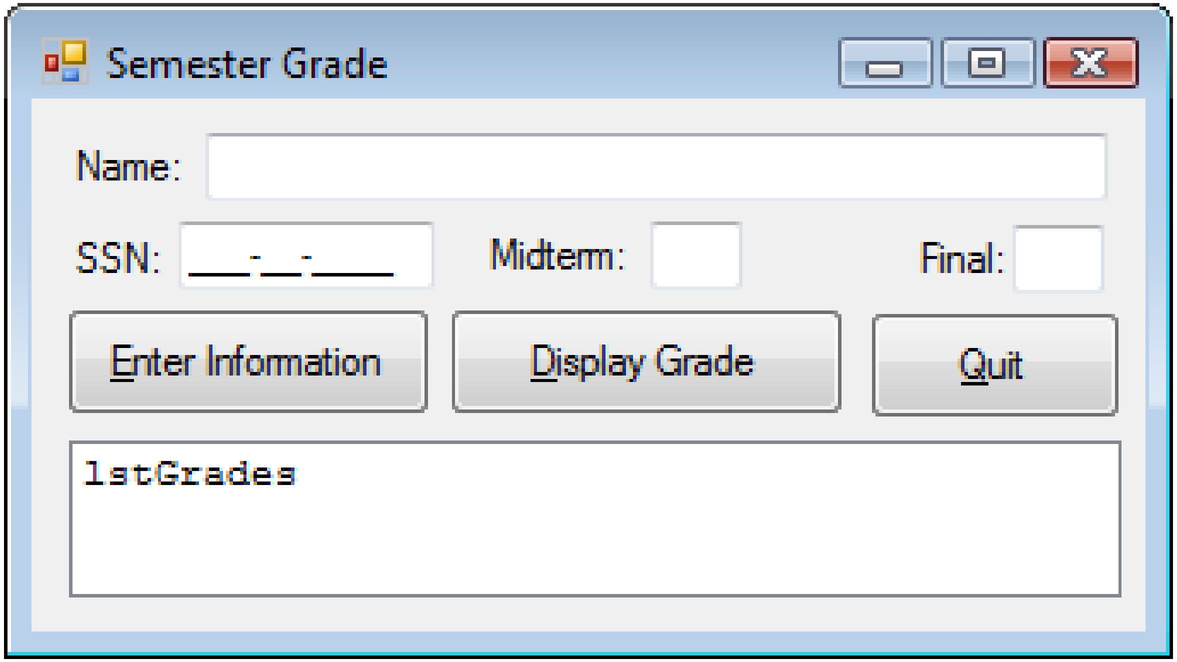
Student Class: Method

```
Function CalcSemGrade() As String
  Dim grade As Double
  grade = (m_midterm + m_final) / 2
  grade = Math.Round(grade)
  Select Case grade
    Case Is >= 90
      Return "A"
    Case Is >= 80
      Return "B"
:
End Function
```

Student Class

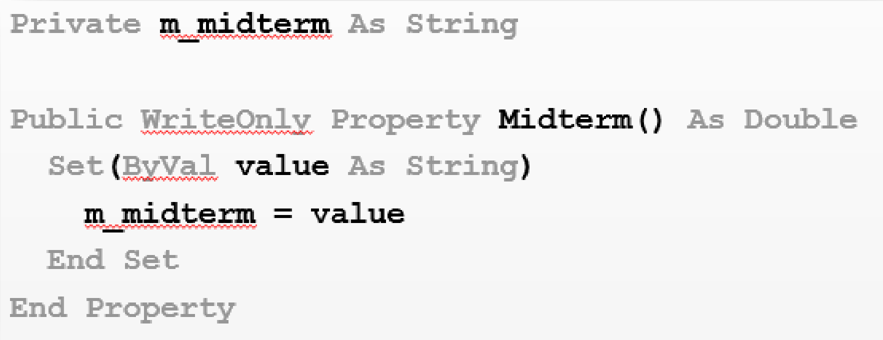
```
Class Student
   (Four Private Declaration statements)
   (Four Property Blocks)
   Function CalcSemGrade() As String
      :
   End Function
End Class    'Student
```

Example 1: Form



Example 1: Form Code

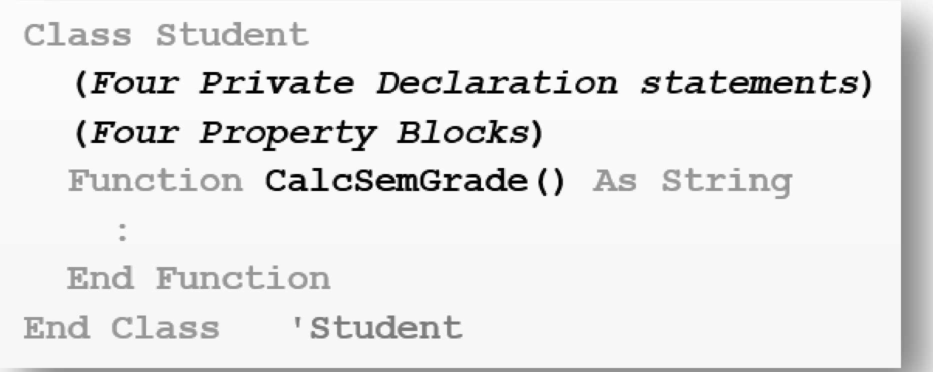
```
Dim pupil As Student

Private Sub btnEnter_Click(…) Handles btnEnter.Click
pupil = New Student()  'Create instance of
'Student
'Read the values stored in the text boxes
pupil.Name = txtName.Text
pupil.SocSecNum = mtxtSSN.Text
pupil.Midterm = CDbl(txtMidterm.Text)
pupil.Final = CDbl(txtFinal.Text)
lstGrades.Items.Clear()
lstGrades.Items.Add("Student Recorded.")
End Sub
```



```
Private m_midterm As String

Public WriteOnly Property Midterm() As Double
  Set(ByVal value As String)
    m_midterm = value
  End Set
End Property
```

EXAMPLE 1: FORM CODE CONTINUED

```
Private Sub btnDisplay_Click(...) Handles
  btnDisplay.Click
   Dim fmtStr As String = "{0,-20}{1,-15}{2,-4}"
   lstGrades.Items.Clear()
   lstGrades.Items.Add(String.Format(fmtStr, _
       pupil.Name, pupil.SocSecNum, _
       pupil.CalcSemGrade))
End Sub
Private Sub btnQuit_Click(...) Handles btnQuit.Click
   End
End Sub
```



```
Class Student
  (Four Private Declaration statements)
  (Four Property Blocks)
  Function CalcSemGrade() As String
     :
  End Function
End Class    'Student
```

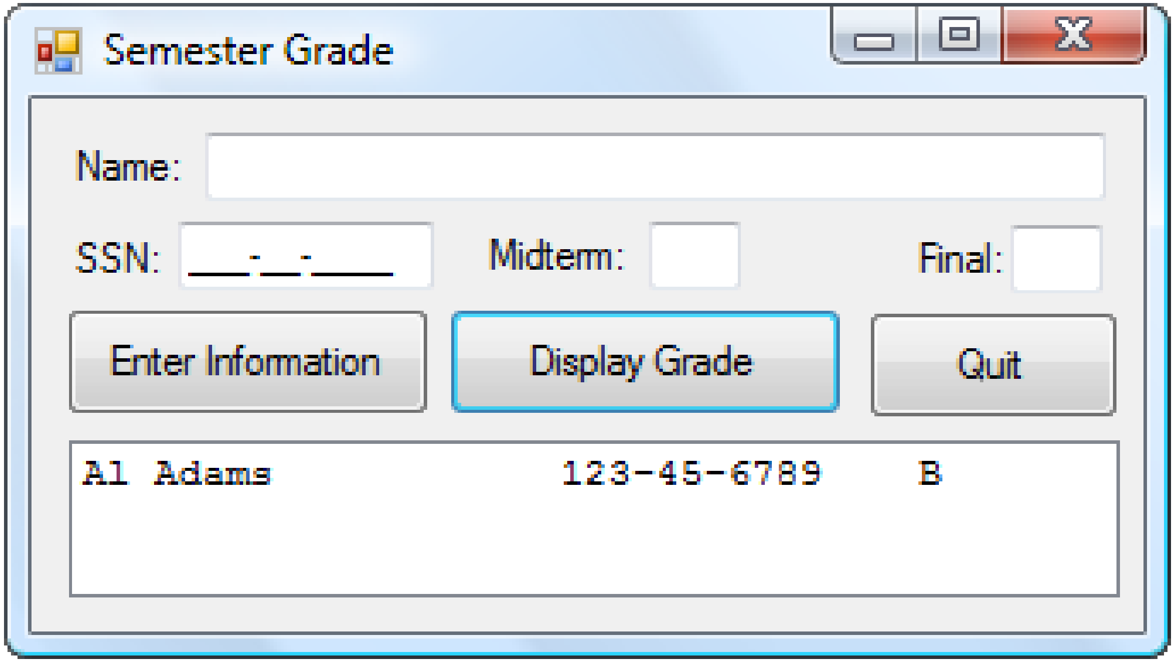
Example 1: Form Code continued

Calls the Get property procedure

```
lstGrades.Items.Add(String.Format(fmtStr, pupil.Name,
            pupil.SocSecNum, pupil.CalcSemGrade))
```

Calls the CalcSemGrade method

Example 1: Output



```
Al  Adams                   123-45-6789     B
```

Steps Used to Create a Class

1. Identify a *thing* in your program that is to become an object

2. Determine the properties and methods that you would like the object to have. (As a rule of thumb, properties should access data, and methods should perform operations.)

3. A class will serve as a template for the object. The code for the class is placed in a class block of the form

```
Class ClassName
    statements
End Class
```

Steps continued

4. For each of the properties in Step 2, declare a private member variable with a statement of the form

   ```
   Private m_variableName As DataType
   ```

5. For each of the member variables in Step 4, create a Property block with Get and/or Set procedures to retrieve and assign values of the variable.

6. For each method in Step 2, create a Sub procedure or Function procedure to carry out the task.

Example 2: PFStudent

- PF stands for Pass/Fail
- Example 2 has the same form and code as Example 1, except for the CalcSemGrade method.

PFStudent Class: Method

```
Function CalcSemGrade() As String
  Dim grade As Double
  grade = (m_midterm + m_final) / 2
  grade = Math.Round(grade)
  If grade >= 60  Then
    Return "Pass"
  Else
    Return "Fail"
End Function
```

OUTPUT:

```
Adams, Al    123-45-6789  Pass
```

OBJECT CONSTRUCTORS

- Each class has a special method called a **constructor** that is always invoked when the object is instantiated

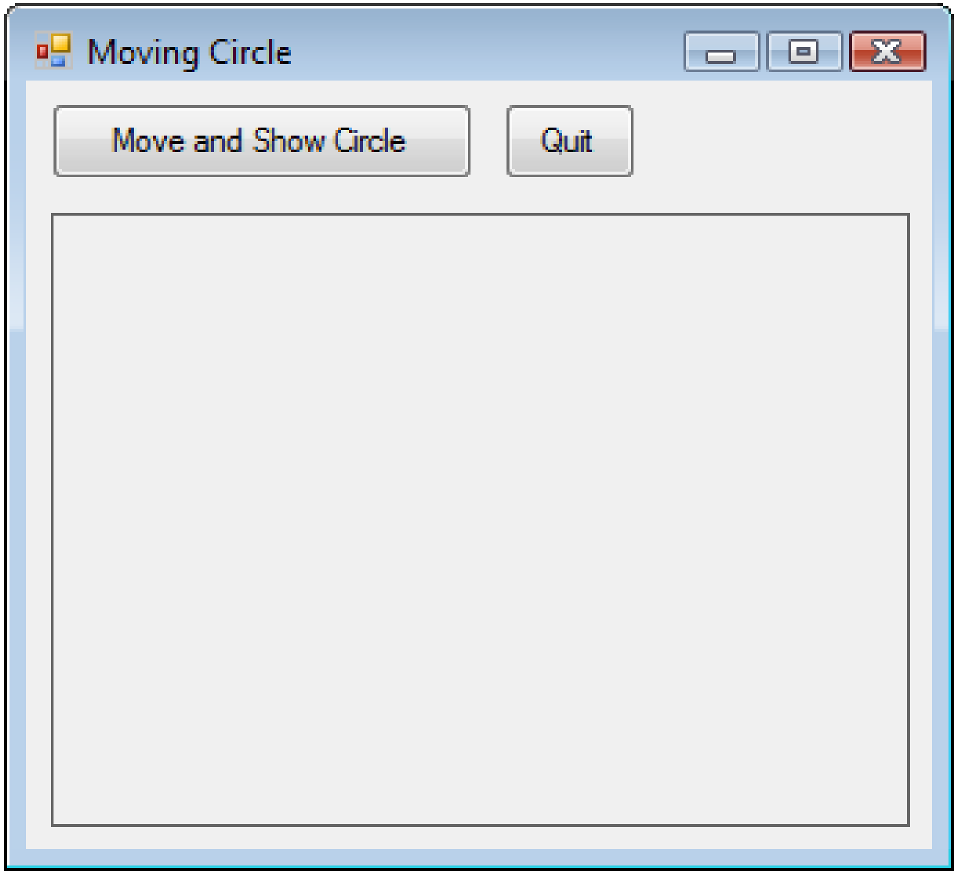
- The constructor may take arguments

- It is used to perform tasks to initialize the object

- The first line of the constructor has the form:

```
Public Sub New(ByVal par1 As dataType, ...)
```

Example 3: Form



Example 3: Circle Class Member Variables

```
Class Circle
  Private m_x As Integer
'Dist from left side _
'of picture box to circle

  Private m_y As Integer 'Distance from top
  '                of picture box to circle

  Private m_d As Integer 'Diameter of circle
```

Example 3: Property Block

```
Public Property Xcoord() As Integer
  Get
    Return m_x
  End Get
  Set(ByVal value As Integer)
    m_x = value
  End Set
End Property
```

Example 3: Property Block

```
Public Property Ycoord() As Integer
  Get
    Return m_y
  End Get
  Set(ByVal value As Integer)
    m_y = value
  End Set
End Property
```

Example 3: Property Block

```
Public Property Diameter() As Integer
   Get
      Return m_d
   End Get
   Set(ByVal value As Integer)
      m_d = value
   End Set
End Property
```

Example 3: Circle Class Constructor

```
Public Sub New()
  'Set the initial location of the
  'circle to the upper left corner of
  'the picture box, and set its
  'diameter to 40.
  Xcoord = 0
  Ycoord = 0
  Diameter = 40
End Sub
```

Example 3: Circle Class Methods

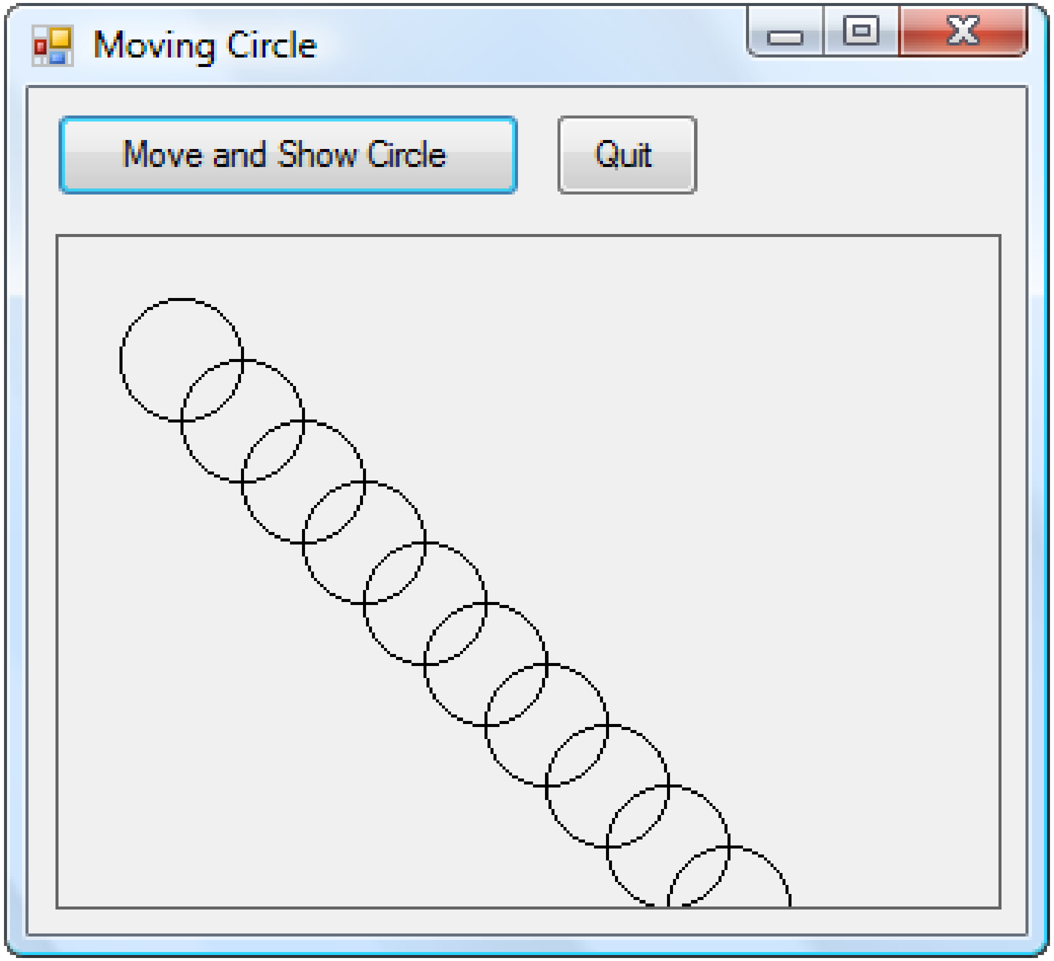
```
Sub Show(ByRef g As Graphics)
  'Draw a circle with given specifications
  g.DrawEllipse(Pens.Black, Xcoord, _
               Ycoord, Diameter, Diameter)
End Sub

Sub Move(ByVal distance As Integer)
  Xcoord += distance
  Ycoord += distance
End Sub
```

Example 3: Form's Code

```
Class frmCircle
  Dim round As New Circle()
  Private Sub btnMove_Click(...) Handles btnMove.Click
    round.Move(20)
    round.Show(picCircle.CreateGraphics)
  End Sub
  Private Sub btnQuit_Click(...) Handles btnQuit.Click
    End
  End Sub
End Class   'frmCircle
```

EXAMPLE 3: OUTPUT

Press the Move button ten times.



11.2 Arrays of Objects; Events; Containment

"An object without an event is like a telephone without a ringer."

-*Anonymous*

Arrays of Objects

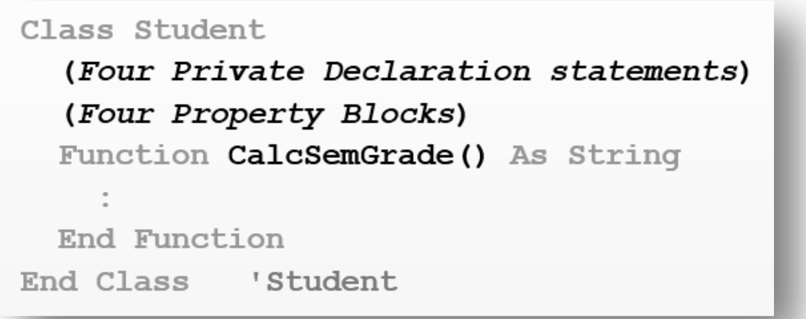
- Arrays have a data type
- That data type can be of User Defined Type
- Therefore, we can have arrays of objects

Example 1: Code

Uses an array of Student objects. Same form design as Example 1 of Section 11.1, but with the following code modifications.

```
Dim students(50) As Student                'Class-level
Dim lastStudentAdded As Integer = -1  'Class-level
```

```
Dim pupil As New Student()   'In btnEnter_Click
pupil.Name = txtName.Text
pupil.SocSecNum = txtSSN.Text
pupil.Midterm = CDbl(txtMidterm.Text)
pupil.Final = CDbl(txtFinal.Text)
'Add the student to the array
lastStudentAdded += 1
students(lastStudentAdded) = pupil
```



```
Class Student
  (Four Private Declaration statements)
  (Four Property Blocks)
  Function CalcSemGrade() As String
     :
  End Function
End Class    'Student
```

Events

- User-defined events can be created for classes.

- The statement for triggering an event is located in the class block

- The event is dealt with in the form's code.

User Defined Event

- Suppose that the event is named UserDefinedEvent and has the parameters *par1*, *par2*, and so on.

- In the class block, place the following statement in the Declarations section

```
Public Event UserDefinedEvent(ByVal par1 As _
     DataType1, ByVal par2 As DataType2, ...)
```

- The next statement should be placed at the locations in the class block code at which the event should be triggered

```
RaiseEvent UserDefinedEvent(arg1, arg2, ...)
```

Responding to Events

- When declaring an object variable, the keyword WithEvents must be added so that the object will respond to events:

```
Dim WithEvents object1 As ClassName
```

- The declaration line of an event procedure would be

```
Private Sub object1_UserDefinedEvent(ByVal par1 As _
      DataType1, ...) Handles object1.UserDefinedEvent
```

Example 2: Code

- Same form design as Example 3 of Section 11.1
- Addition of a text box called txtCaution
- Contains the event PositionChanged that is triggered whenever the circle moves
- The following code modifications are incorporated in the Declarations section of the Circle class, add

```
Public Event PositionChanged(ByVal x As Integer, _
           ByVal y As Integer, ByVal d As Integer)
```

In the Move Sub procedure of the Circle class, add the line

```
RaiseEvent PositionChanged(Xcoord, Ycoord, _
                           Diameter)
```

EXAMPLE 2: CODE CONTINUED

- In the Form's code, change the object's declaration statement to

```
Dim WithEvents round As New Circle()
```

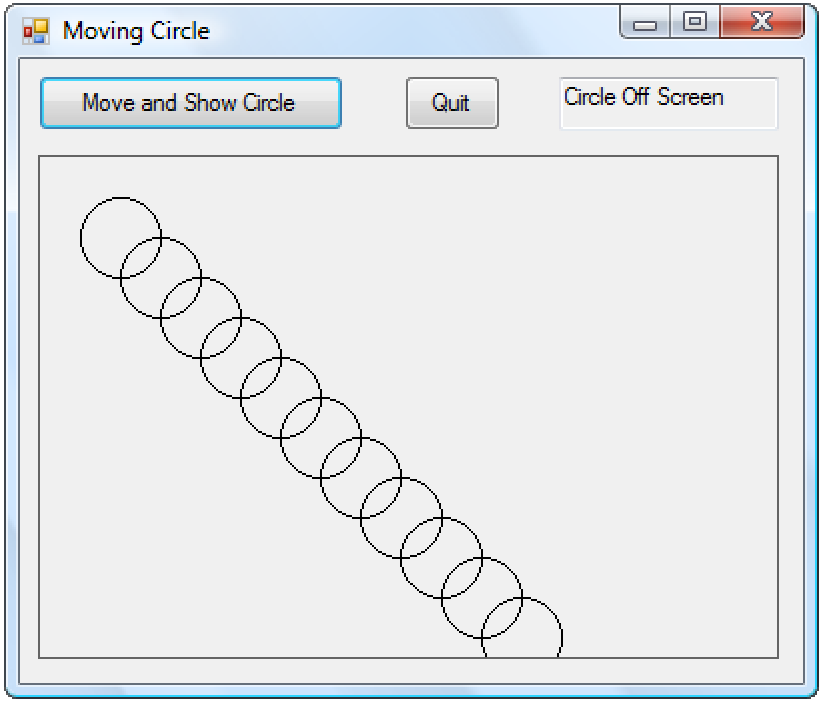
Example 2: Code continued

- Add the following event procedure:

```
Sub round_PositionChanged(ByVal x As Integer, _
      ByVal y As Integer, ByVal d As Integer) _
      Handles round.PositionChanged
  If (x + d > picCircle.Width) Or _
            (y + d > picCircle.Height) Then
    txtCaution.Text = "Circle Off Screen"
  End If
End Sub
```

EXAMPLE 2: OUTPUT

Press the Move button eleven times.



Circle Off Screen

11.3 Inheritance

- Inheritance
- Polymorphism and Overriding
- Abstract Properties, Methods, and Classes

11.3 Inheritance

- The three relationships between classes are “use,” “containment,” and “inheritance.”

- One class **uses** another class if it manipulates objects of that class.

- Class A **contains** class B when a member variable of class A makes use of an object of type class B.

Containment

- Class A **contains** class B when a member variable of class A is an object of type class B.

```
Class DeckOfCards
  Private m_deck(51) As Card
  'Class DeckOfCards contains class Card
```

INHERITANCE RELATIONSHIP

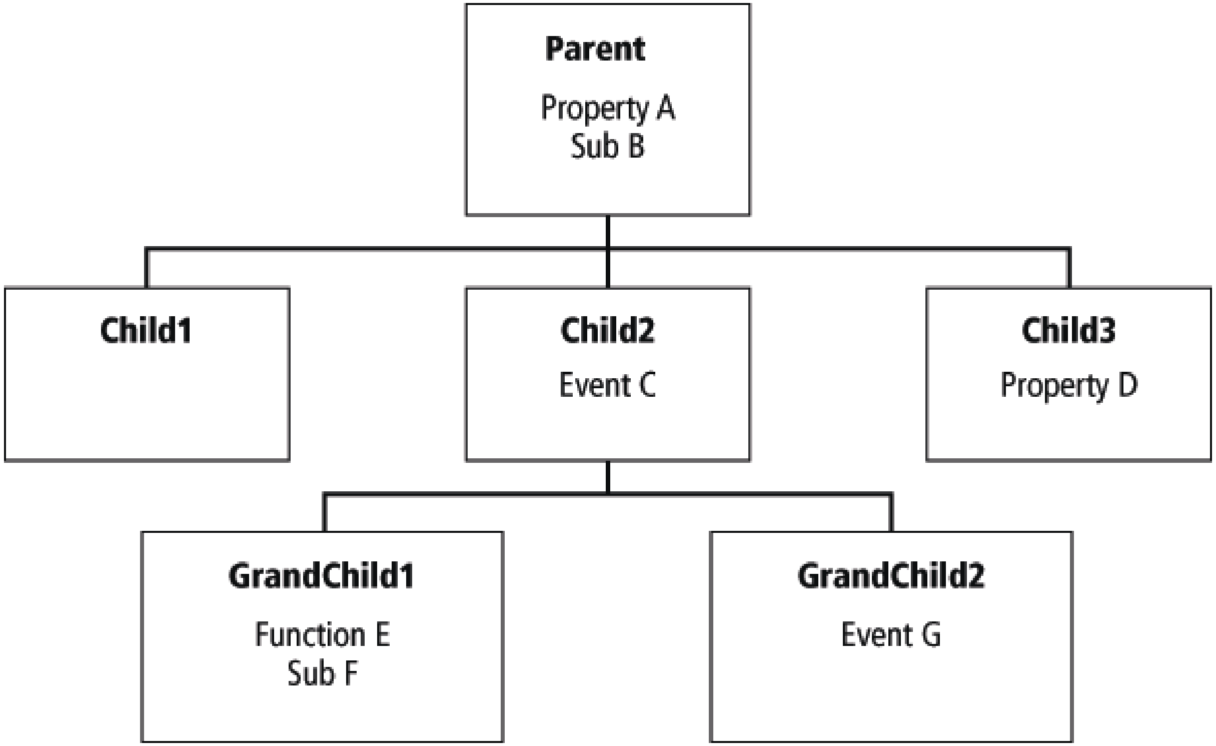
- **Inheritance** is a process by which one class (the **child** or **derived** class) inherits the properties, methods, and events of another class (the **parent** or **base** class).

- The child has access to all of its parent's properties, methods and events as well as to some of its own.

- If the parent is itself a child, then it and its children have access to all of its parent's properties, methods and events.

Inheritance Hierarchy

- *GrandChild1* has access to *Property A*, *Sub B*, and *Event C* from its parent and adds *Function E* and *Sub F*



Benefits of Inheritance

- Allows two or more classes to share some common features yet differentiate themselves on others.

- Supports **code reusability** by avoiding the extra effort required to maintain duplicate code in multiple classes.

Designing Objects

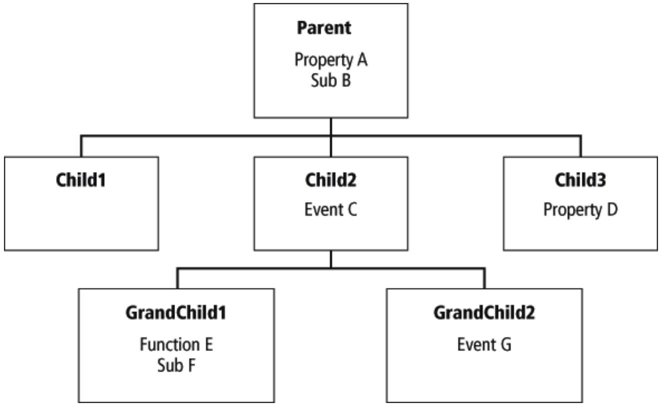
- Programmers need the ability to identify useful hierarchies of classes and derived classes

- Software engineers are still working on the guidelines for when and how to establish hierarchies

- The **ISA test**: If one class *is a* more specific case of another class, the first class should inherit from the second class

INHERITS

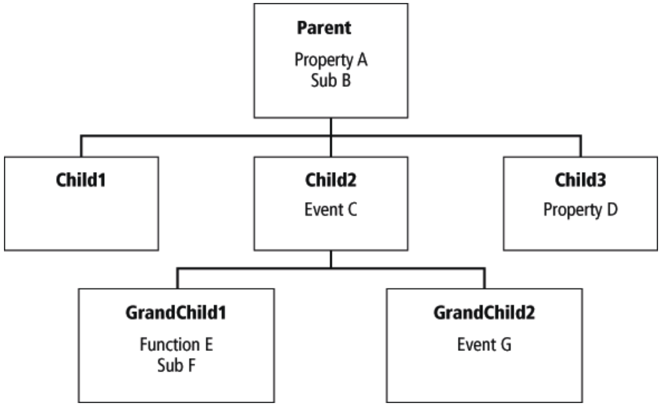
```
Class Parent
   Property A
      'Property Get and Set blocks
   End Property
   Sub B()
      'Code for Sub procedure B
   End Sub
End Class


Class Child2
   Inherits Parent
   Event C()
End Class
```



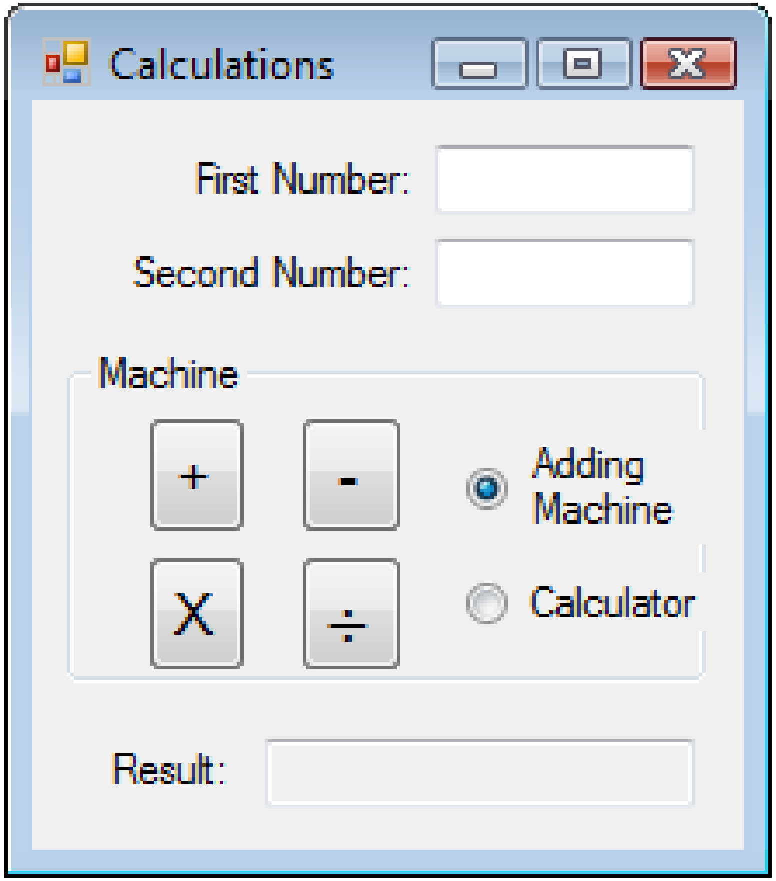
Indentifies the Parent Class: Child2 inherits From Parent

Child class as parent

```
Class GrandChild1
  Inherits Child2
  Function E()
    'Code for function E
  End Function
  Sub F()
    'Code for Sub procedure F
  End Sub
End Class
```



Example 1: Form



● Adding Machine

○ Calculator

Adding Machine and Calculator Classes

- Adding Machine – a machine that is capable of adding and subtracting
- Calculator – a machine that is capable of adding, subtracting, multiplying, and dividing
- A calculator is an adding machine
- Therefore, the calculator class should inherit from the adding machine class

AddingMachine Class

```
Class AddingMachine

    Public Property FirstNumber() As Double
    Public Property SecondNumber() As Double

    Function Add() As Double
        Return FirstNumber + SecondNumber
    End Function

    Function Subtract() As Double
        Return FirstNumber - SecondNumber
    End Function
End Class     'AddingMachine
```

Calculator Class

```
Class Calculator
  Inherits AddingMachine
  'Calculator inherits properties FirstNumber and
  'SecondNumber and functions add() and subtract().

  Function Multiply() As Double
    Return FirstNumber * SecondNumber
  End Function

  Function Divide() As Double
    Return FirstNumber / SecondNumber
  End Function
End Class     'Calculator
```

Polymorphism and Overriding

- The set of properties, methods, and events for a class is called the class **interface**

- The interface defines how the class will behave

- Programmers only need to know how to use the interface in order to use the class

Polymorphism

- Literally means "many forms."
- The feature that two classes can have methods that are named the same and have essentially the same purpose, but different implementations, is called **polymorphism**

Employing polymorphism

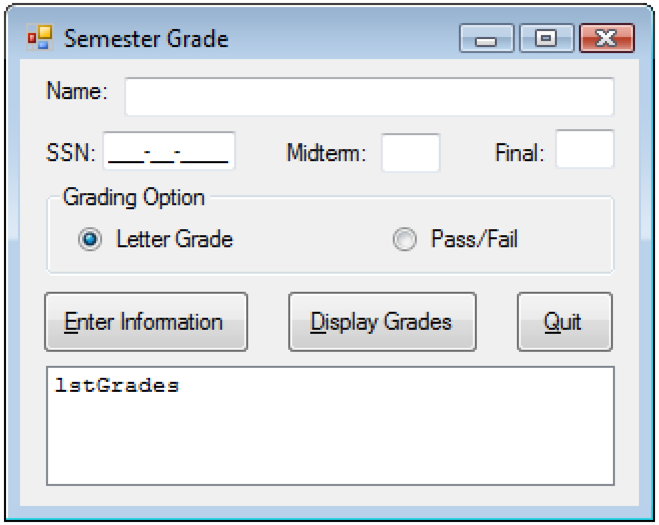
- A programmer may employ polymorphism in three easy steps
- First, the properties, methods, and events that make up an interface are defined
- Second, a parent class is created that performs the functionality dictated by the interface
- Finally, a child class inherits the parent and overrides the methods that require different implementation than the parent

Overridable

- The keyword **`Overridable`** is used to designate the parent's methods that are overridden, and the keyword **`Overrides`** is used to designate the child's methods that are doing the overriding

- There are situations where a child class's needs to access the parent class's implementation of a method that the child is overriding. Visual Basic provides the keyword **`MyBase`** to support this functionality

Example 2: Form



Grading Option

● Letter Grade ○ Pass/Fail

Example 2

- The objective of this program is similar to that of Example 1 in Section 11.2

- This program will consider two types of students
  - ordinary students who receive letter grades
  - pass/fail students

- We will have a Student class and a PFStudent class
  - PFStudent class inherits everything from the Student class
  - PFStudent class will *override* the CalcSemGrade method with its own

- In the class Student, replace

```
Function CalcSemGrade() As String
```

with

```
Overridable Function CalcSemGrade() As String
```

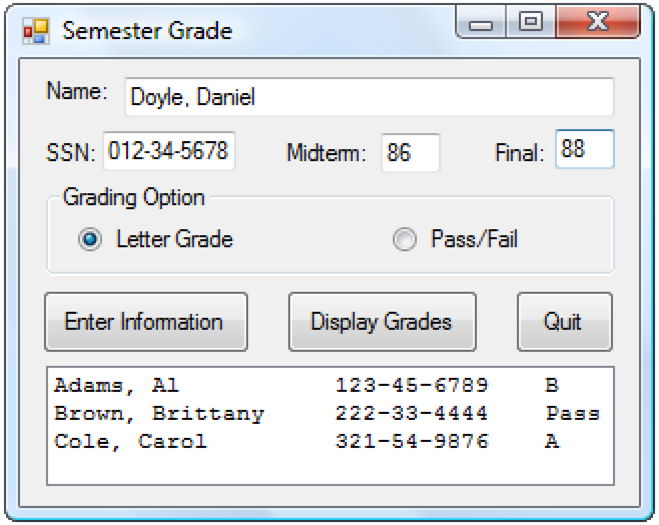
Example 2: PFStudent Class

```
Class PFStudent
  Inherits Student

  Overrides Function CalcSemGrade() As String
    'The student's grade for the semester
    If MyBase.CalcSemGrade = "F" Then
      Return "Fail"
    Else
      Return "Pass"
    End If
  End Function
End Class     'PFStudent
```

Example 2: Form's Code

```
Public Class frmGrades
  Dim students(50) As Student           'Stores the class
  Dim lastStudentAdded As Integer = -1
              'Last student added to students()

  Private Sub btnEnter_Click(ByVal sender As System.Object,
                             ByVal e As System.EventArgs)
                              Handles btnEnter.Click
    'Stores a student into the array.
    Dim pupil As Student
    'Create the appropriate object
    If radPassFail.Checked Then
      pupil = New PFStudent()
    Else
      pupil = New Student()
    End If
…
```

Example 2: Output



Name: Doyle, Daniel

SSN: 012-34-5678   Midterm: 86   Final: 88

Grading Option

● Letter Grade   ○ Pass/Fail

```
Adams, Al           123-45-6789    B
Brown, Brittany     222-33-4444    Pass
Cole, Carol         321-54-9876    A
```

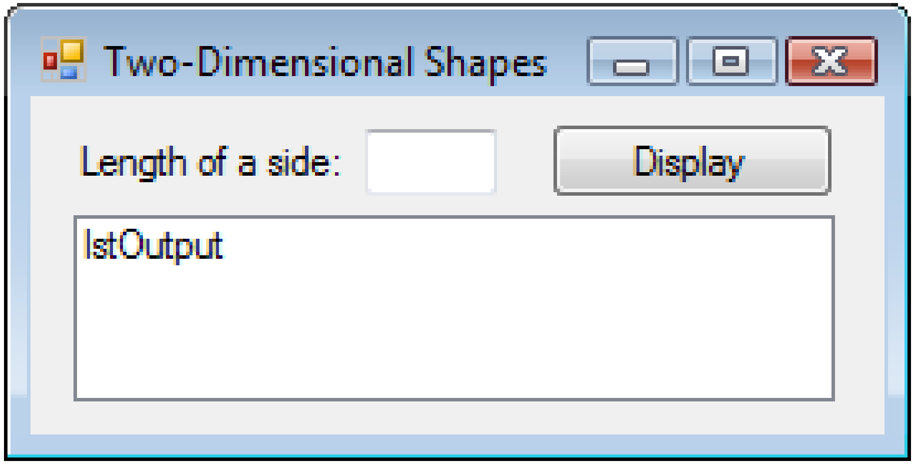
Abstract Properties, Methods and Classes

- Sometimes you want to insist that each child of a class have a certain property or method that it must implement for its own use

- Such a property or method is said to be **abstract** and is declared with the keyword **MustOverride**

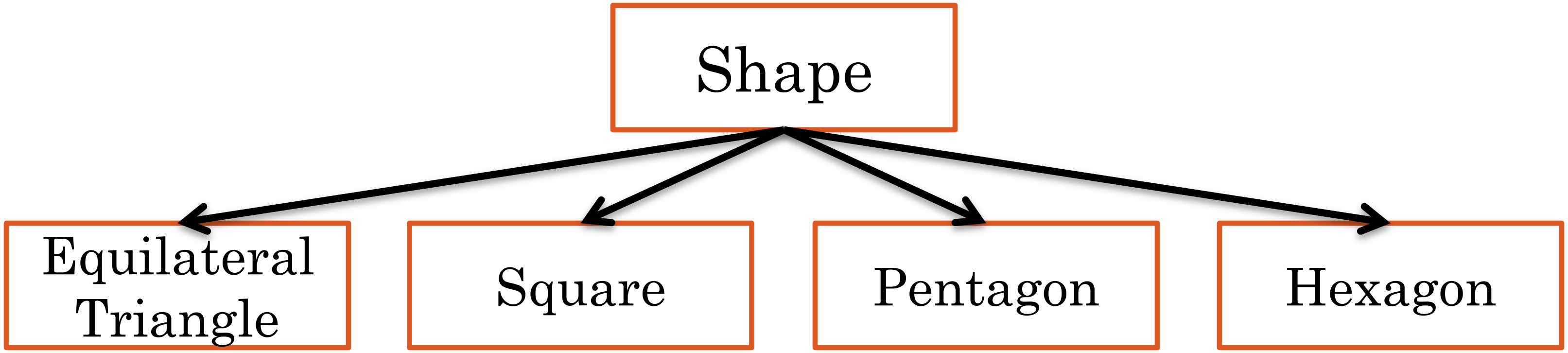
- An abstract property or method consists of just a declaration statement with no code following it

Example 3: Form



The program will display the names and areas of several different regular polygons given the length of one side.

EXAMPLE 3: CODE FOR PARENT CLASS - SHAPE



```
MustInherit Class Shape

  Public Property Length() As Double

  MustOverride Function Name() As String
  'Returns the name of the shape

  MustOverride Function Area() As Double
  'Returns the area of the shape
End Class      'Shape
```

Example 3: Code for Child Class – Equilateral Triangle

```
Class EquilateralTriangle
  Inherits Shape

  Overrides Function Name() As String
    'The name of this shape
    Return "Equilateral Triangle"
  End Function

  Overrides Function Area() As Double
    'Formula for the area of an equilateral triangle
    Return Length * Length * Math.Sqrt(3) / 4
  End Function
End Class     'EquilateralTriangle
```

Example 3: Code for Child Class - Square

```
Class Square
  Inherits Shape

  Overrides Function Name() As String
    'The name of this shape
    Return "Square"
  End Function

  Overrides Function Area() As Double
    'Formula for the area of a square
    Return Length * Length
  End Function
End Class     'Square
```

Example 3: Code for Child Class - Pentagon

```
Class Pentagon
  Inherits Shape

  Overrides Function Name() As String
    'The name of this shape
    Return "Pentagon"
  End Function

  Overrides Function Area() As Double
    'Formula for the area of a pentagon
    Return Length * Length * Math.Sqrt(25 + (10 *
Math.Sqrt(5))) / 4
  End Function
End Class     'Pentagon
```

Example 3: Code for Child Class - Hexagon

```
Class Hexagon
  Inherits Shape

  Overrides Function Name() As String
    'The name of this shape
    Return "Hexagon"
  End Function

  Overrides Function Area() As Double
    'Formula for the area of a hexagon
    Return Length * Length * 3 * Math.Sqrt(3) / 2
  End Function

End Class     'Hexagon
```

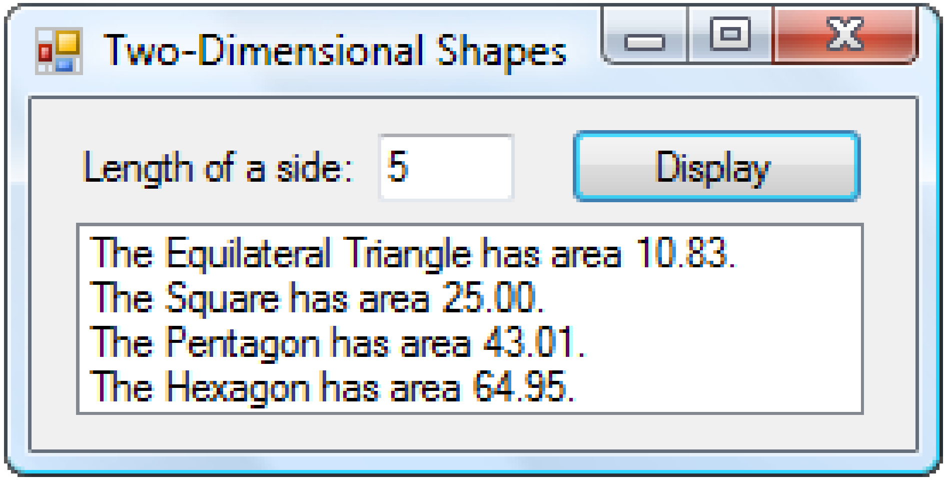
Example 3: Form's Code

```
 Private Sub frmShapes_Load(ByVal sender As System.Object,
                             ByVal e As System.EventArgs)
Handles MyBase.Load
    'Populate the array with shapes
    shape(0) = New EquilateralTriangle()
    shape(1) = New Square()
    shape(2) = New Pentagon()
    shape(3) = New Hexagon()
  End Sub
```

Example 3: Form's Code continued

```
 Private Sub btnDisplay_Click(ByVal sender As System.Object,
                               ByVal e As System.EventArgs)
                               Handles btnDisplay.Click
    Dim length As Double
    'Set lengths of all shapes
    length = CDbl(txtLength.Text)
    For i As Integer = 0 To 3
      shape(i).Length = length
    Next
    'Display results
    lstOutput.Items.Clear()
    For i As Integer = 0 To 3
      lstOutput.Items.Add("The " & shape(i).Name & " has area " &
                          FormatNumber(shape(i).Area) & ".")
    Next
  End Sub

End Class     'frmShapes
```

Example 3: Output



The Equilateral Triangle has area 10.83.
The Square has area 25.00.
The Pentagon has area 43.01.
The Hexagon has area 64.95.