**OOP and Inheritance worked Example (Visual Basic 2015)**

This worked example will take students through the basics of:

* Creating a base class with custom properties and methods
* Deriving a new class from the base class by the use of *Inherits* statements

This activity does assume the student has a basic understanding of the principles of OOP, including the concepts of sub classes, super classes, encapsulation and inheritance.

# Introduction: Adding a New Class to Your Project

A *class* in Visual Basic is a blueprint that sets out the structure of one or more objects. Creating a class allows you to define your own objects in a program. As you already know objects have properties, methods and attributes. To add a new class to your project, you click the Add Class command on the Project menu, and then you define the class by using program code and a few Visual Basic keywords.

In the following worked example, you’ll create a program that prompts an owner for their new pet’s type, name and date of birth. You’ll store this information in the properties of a new class named *Animal*, and you’ll create a method in the class to compute the current age of the owner’s new pet. This worked example will teach you how to create your own classes and also how to use the classes in the event procedures of your program.

# Stage 1: Building the Animal Class project

1. Start Visual Studio, and create a new Windows Form Application project named **My Animal Class.**
2. Use the *Label* control to add a label object on the top of Form1.
3. Use the *TextBox* control to add two text box objects to the top of Form1.
4. Use the *DateTimePicker* control add a date time picker object below the text box objects.
5. Use the *Button* control to add a button object below the date time picker object.
6. Set the following properties for the objects on the form:

|  |  |  |
| --- | --- | --- |
| **Object** | **Property** | **Setting** |
| ***Label1*** | *Text* | “Enter your new pets type, name and date of birth” |
| ***TextBox1*** | *Text* | “Pet type” |
| ***TextBox2*** | *Text* | “Pet name” |
| ***Button1*** | *Text* | “Display record” |
| ***Form1*** | *Text* | “Animal Class” |

Your form looks something like this:

This is the basic user interface for a form that defines a new pet record.

Now you’ll add a class to the project to store the information in the record.

1. Choose the Add Class… option from the Project menu.

Visual studio displays the Add New Item dialog box, with the Class template selected, as show here:

1. Type **Animal.vb** in the Name box, and then click Add.

Visual Studio opens a blank class module in the Code Editor and lists a file name Animal.vb in the Solution Explorer for your project, as show here:

Now you’ll type the code which makes up the class definition. In other words you are going to set up your “template” for the Animal class.

You’ll follow three steps to do this

* Step 1: Declare class variables,
* Step 2: Create properties
* Step 3: Create a method,

Finally, in a 4th step you will instantiate (create) a new object based on your new Animal class.

## Step 1: Declare class variables

1. Below the Public Class Animal line, type the following variable declarations:

Private petType As String

Private petName As String

Here you have declared two variables that will be used exclusively within the class to store the values for two string property settings. They have been declared using the Private keyword because, by convention, it is good practice to keep internal class variables private – in other words, not available for inspection outside the class itself (this is the principle of encapsulation which you have been learning about).

## Step 2: Create properties

1. Below the variable declaration, type the following program statement:

Public Property Type() As String

1. Directly underneath this line type the following code. You will notice that much of the code is automatically added for you once you type Get and press Enter:

 Get

 Return petType

 End Get

 Set(value As String)

 petType = value

 End Set

End Property

The *Return* keyword specifies the that *Name1* string variable will be returned when the *Type* property is referenced. The *Set* block assigns a string value to the *Name1* variable when the property is set. Notice here especially the *value* variable, which is used in the property procedures to stand for the value that’s assigned to the class when a property is set.

1. Below the *End Property* statement, type a second property procedure for the *Name* property in your class. It should look like the code on the following page. Once again as you start to type and press enter Visual Studio will fill in much of the code for you:

 Public Property Name() As String

 Get

 Return petName

 End Get

 Set(value As String)

 petName = value

 End Set

End Property

This property procedure is similar to the first one, except that it uses the second string variable (*petName*) that you declared at the top of the class.

You’re finished defining the two properties in your class. Now let’s move on to a method named *Age* that will determine the age of your new pet based on its date of birth.

## Step 3: Create a method

1. Below the *Name* property procedure you just wrote, type the following function definition.

 Public Function age(ByVal Birthday As Date) As Integer

 Return Int(Now.Subtract(Birthday).Days / 365.25)

 End Function

To create a method in the class that performs a specific action, you add a function or a Sub procedure to your class. Although many methods don’t need parameters passing in, the *Age* method we are defining requires a *Birthday* parameter of type *Date* to complete its calculation. The method uses the *Subtract* method to subtract the new pets birth date from the current system time, and it returns the value expressed in days divided by 365.25. The *Int* function converts this value to an integer, and this number is returned to the calling procedure via the *Return* statement – just like a standard function.

Your class definition (template) is finished, and in the Code Editor, the *Animal* class now looks like the following:


## Step 4: Create an object based on the new class

1. Right click Form1.vb in the Solution Explorer and choose “View Designer” from the pop up menu. The Form1 user interface should appear.
2. Double-click the “Display Record” button to open the *Button1\_Click* event procedure in the Code Editor.
3. Type the following program statements:

 Dim Pet As New Animal

 Dim DOB As Date

 Pet.Type = TextBox1.Text

 Pet.Name = TextBox2.Text

 DOB = DateTimePicker1.Value.Date

MsgBox("Your pet " & Pet.Type & “ Named: “ & Pet.Name & “ is “ & Pet.age(DOB) & “ years old.”)

This routine stores the values entered by the user in an object named *Pet* that’s declared as type *Animal*. The *New* keyword indicates that you want to immediately create a new instance of the *Animal* object. You’ve declared variables often, now you get to declare one based on a class you created yourself! The routine then declares a *Date* variable named *DOB* to store the date entered by the user, and the *Type* and *Name* properties of the *Pet* object are set to the Type and Name returned by the two text box objects on the form. The value returned by the date and time picker object is stored in the *DOB* variable, and the final program statement displays a message box which joins all of this information together into a single sentence and displays it to you.

1. Click the Save All button to save your changes.
2. Click the Start button to run the program.

The user interface appears ready for your input.

1. Type a you Pet type into the first box (Dog, Cat etc) and its Name into the second box.
2. Click the date time picker object’s arrow, and choose your Pet’s Date of Birth.

Your form looks similar to this:

1. Click the Display Record button.

Your program stores the pet type and its name values in the property settings and uses the *Age* method to calculate the new pet’s current age. A message box displays the result, as shown here:

1. Click OK to close the message box, and then experiment with a few different date values, clicking Display Record each time you change the birth date field.

# Stage 2: Inheriting a Base Class

Now we have a working class let’s see how in Visual Basic we can inherit from that class to create a new sub-class. Again we will be using the Add Class command and a class module. The mechanism for inheriting a base (parent / super) class is to use the *Inherits* statement to include the previously defined class in a new class. You can then add additional properties or methods to the derived (child / sub) class to distinguish it from the base class.

In the following exercise, you will modify the “My Animal Class” project so that it stores information about new offspring and how much you want to sell each one for (let’s face it, one of the best thing about your new Cat having eight kittens is being able to sell them to your friends)! First you’ll add a second user-defined class, named *newBorn*, to the *Animal* class module. This new class will inherit the *Type* property, the *Name* property, and the *Age* method from the *Animal* class will add an additional property named *Cost* to store the price you wish to sell each one for.

## Use the ‘Inherits’ keyword

1. Right click the Animal.vb in Solution Explorer, and then choose “View Code” from the menu.
2. Scroll to the bottom of the Code Editor so that the insertion point is below the final *End Class* statement.

You can include more than one class in a class module, as long as each class is split up inside its own *Public Class* and *End Class* statements. You’ll create a class named *newBorn* in this class module, and you’ll use the *Inherits* keyword to incorporate the method and properties you defined in the *Animal* class.

1. Type the following class definition in the Code Editor. As with before Visual Studio will start to fill in much of this class template for you as you start to type and press Enter:

Public Class newBorn

 Inherits Animal

 Private Price As Integer

 Public Property Cost() As Integer

 Get

 Return Price

 End Get

 Set(value As Integer)

 Price = value

 End Set

 End Property

End Class

The *Inherits* statement links the *Animal* class to this new class, incorporating all of its a, properties, and methods. Here we have defined the *newBorn* class as a special type of *Animal* class – in addition to the *Type* and *Name* properties, the *newBorn* class has a *Price* property that records the cost you are prepared to sell each new born pet for.

Now you will use the new classes *Button1\_Click* event procedure.

1. Bring up the code window which has the code for the *Button1\_Click* event procedure in Form1.

Rather than create a new variable to hold the *newBorn* class, we will just use the *pet* variable as it is – the only difference will be that we can now set a *Cost* property for the new newBorn pet.

1. Modify the *Button1\_Click* event procedure as follows. (The shaded lines are the ones that you need to change).

 Dim Pet As New newBorn

 Dim DOB As Date

 Pet.Type = TextBox1.Text

 Pet.Name = TextBox2.Text

 DOB = DateTimePicker1.Value.Date

 Pet.Cost = InputBox("How much do you want to see your new born pets for?")

MsgBox("Your new born pet of Type: " & Pet.Type & “ Named: “ & Pet.Name & “ is selling for: £" & Pet.Cost)

You will notice we have removed the current age calculation – the *Age* method isn’t used – but I did this only to keep information to a minimum in the message box. When you define properties and methods in a class, you aren’t required to use them in the program code.

Now we will run the program.

1. Click Start to run the program.
2. Type a you Pet type into the first box (Dog, Cat etc) and its Name into the second box.
3. Click the date time picker object, and scroll choose the date of birth of your pet.
4. Click the Display Record button.

Your program stores the Type of pet and its Name values in property settings and then displays the following input box, which prompts you for the price you would like to sell each new born pet for:

1. Type 30, and then click OK to close the input box.

The application stores the number 30 in the new *Grade* property and uses all the data captured to display the new employee information in a confirming message box. You see this message:

1. Experiment with a few more values if you like, and then click the Close button on the form.

*Note: This is a heavily adapted and updated version of a worked example which originally appeared in “Microsoft® Visual Basic® 2008 Step-by-Step” by Michael Halvorson.*