CS608 Lecture Notes

Visual Basic.NET Programming

Introduction to Visual Basic.NET

VB.NET Programming Environment (Review)

(Part I of IV)

(Lecture Notes 1A)

Prof. Abel Angel Rodriguez

Chapter 1   Introduction to Object-Oriented Programming

1.1 Understanding Object-Oriented Programming

1.1.1 The Procedural Programming Approach to Programming

❑ We will begin this course with a brief discussion of the programming methodologies that you are most likely accustom to in your previous Visual Basics.Net introductory courses.

❑ Programming as it was done in the past and still being done today in many cases is based on the *Event-Driven* and *Procedural* Programming approach.

❑ These methods of programming are based on what's known as **Structured Programming**.   Structure programming has been the traditional way of programming.

Procedural Programming

❑ If you have taken a course in C, Visual Basic, Pascal, FORTRAN, Cobol etc. the programs you wrote were Procedural.

❑ In procedural programming, the **focus** of the programs was to **solve** a problem.

❑ For example, supposed you were asked to write a program to solve the following problem:

- ▪ Write a Video Management System that will process the rental/return of videos tapes for a retail store such as a program used for Blockbuster Video.

❑ Using a language like C or can be done even with VB.NET, this is usually done as follows:

1. Analyze the problem required to be solved: Design flow chart, algorithm etc.
2. Break the problem into smaller manageable pieces, such as the Rental Module, Return Module, Customer Information Module etc.
3. Design the UI for each of the pieces using Forms , controls or any structure supplied by the language to implement the UI
4. Write code to implement each piece using variables, functions & procedures to implement each of the modular pieces.

❖ **Note that the focus is on solving the problem via the programming code and breaking the problem into smaller manageable pieces**.

❑ Dividing a program into Procedures/functions and modules is one of the cornerstones of structured or procedural programming. But here as some of the drawbacks:

- ▪ As programs grow larger and more complex, even the procedural programming approach begins to show signs of strain. Projects can become too complex, schedules slip, more programmers are added, cost skyrockets etc.
- ▪ In **Procedural programming** data or the variables & data structures that hold or store the data, are usually **unprotected** and may be accessible to functions & procedures that have no business changing them, therefore they can be easily corrupted.
- ▪ Procedural programs are difficult to design because their chief components, procedures, functions and data structures don't model the real world very well.

Event Driven Programming

❑ If you wrote the Video Management Program using Visual Basics 6 or in some cases VB.NET, as it's taught in courses such as CS101 & CS508, then you would normally tend to write this program as an *Event-Driven* Application.

❑ *Event-Driven* applications react to user events or actions such as clicking buttons, check boxes or navigating through forms or graphical front-ends. These programs are still based on the procedural programming philosophy, but are base on code reacting to user actions on the GUI or front-end.

❑ The steps to write an Event-Driven program are as follows:

1. Analyze the problem required to be solved and derive the algorithm:
   - ▪ Design flow chart, algorithm to solve this problem etc.
2. Use Forms & Controls to designed the User Interface (UI)
   - ▪ Drop some controls to implement the GUI, such as labels, text boxes. Command buttons etc.
   - ▪ Use the controls to implement features such as Rental, Return, Customer Information, Video Tape Information etc.

3.   Placed programming code inside the *Event-Handlers of the* controls on the Form, to respond to actions taken by the users on the controls. Such as the *button_Click()* event of a Command Button etc.

❖   **Note that with this approach, the focus again is on breaking the program into sections and solving the problem via the Form, controls & code in the Event-Handlers**

1.1.2 The Object-Oriented Programming (OOP) Approach

Thinking Objects
-         The newer programming languages use a different approach. With OOP, programs are based on <u>**real world objects**</u>.
-         This method of programming is based on creating programming code that emulates real world entities, thus the word Object.
-         In OOP, instead of focusing on solving the problem, **you focus and think in terms of the Objects that will play an important role in the program**.
-         That is you first create the *Objects* that are the important characters in the program, such as Employees, Departments, Customers, Products etc.
-         Using *Objects*, allow programs to be based on <u>real world entities</u>, such as a car, person, employee, customer, inventory part etc
-         Examples of pure OOP languages are **C++, Visual Basics.NET & Java**.

-         In OOP, each object has its own *Properties* or *Attributes* (access to Data), the *Methods* (Functions/Procedures) that operate on the data & the *Events* that are automatically triggered when the object is used or manipulated.
-         The fundamental idea behind object-oriented languages is to combine into a single package both the *data*, *Methods* (functions/procedures) & *Events* (Event-Procedures) that operate on that data. <u>**Such unit is called an** <mark>*object*</mark></u>.
-         Combining the *Data*, *Methods* & *Events* that operate on that data into a single package means that the *Objects* handle themselves and have a life of their own, and most important, they can be <u>*re-used*</u> in other applications
-         The mechanism to implementing Object-Oriented Programming is the <u>**Class**</u> & the <u>**Object**</u>.

-         Object-Oriented approach to solving the *Video Management* problem:
  1.   Analyze the problem required to be solved and derive the algorithm:
     - Design the Objects that are the key protagonists of the program.
     - For example, a Video Object, Customer Object, Employee Object etc.

  2.   Implement or create the **template or Classes** for each of the required Objects with the properties, methods (actions) and events required to perform the functionality of each object. For example implement a video object that behaves as a video, a customer object that behaves as a customer & an employee object that behaves as an employee.
  3.   Use Forms & Controls to designed the User Interface (UI) for implementing the Video Rental/Return processing

  4.   Create the <u>**Objects**</u> and use programming code to manipulate the Objects as necessary via the User Interface Forms to <u>**solve**</u> the problem at hand.

❖   **Note that with this approach, the focus is on the Objects not the problem. The object was the first thing that was created, then the problem was applied to the objects**

Data Encapsulation
-         A very important feature of OOP is Data Encapsulation or Data Hiding.
-         In OOP, the object's data is *Private* thus hidden and is only accessible by the *Public Methods* (Functions/Procedures) and *Public Properties*.
-         *Private* data means that there is no way for the outside world to access the data directly. Thus the data is protected and invisible or hidden from the outside world.
-         *Public* Methods & Properties are the interface or vehicle for the outside world to be able to access or manipulate the data.
-         This means that you can only do to an object what the *Public Methods* and *Properties* allow you to do. If there is no Public Methods or Properties for a particular task, then it can not be done.
-         An Object behaves exactly as they were specified by the Public Class Methods and Properties. No more, no less
-         A benefit of Data Encapsulation is Robustness or a solid, reliable error-free Object.

Reusability

❑   This method of writing program is very powerful and the **objects** written can be easily *re-used* in other applications.

❑   This concept of re-using objects is very powerful and known as *reusability*. This concept has revolutionized the field of programming since reusing objects yields faster and more robust applications. Applications which took longer to developed are now being created at a much faster rate since objects from other applications are being reused, thus saving time on programming and testing.

    ▪   For example if we create a *Customer* **Object** in a Banking Program, we can reuse this Object in a financial program etc. since Customer Objects have similar functionalities.

❑   This concept of *reusability* spawned a new software industry where companies were established whose sole business is to create ready tested Objects to sell to other software development houses.

1.2 Components of an Object-Oriented Program

1.2.1 Understanding Classes & Objects

❑ Real world objects have attributes or properties that define the objects.
❑ Also, real world objects are based on some mold or template.
❑ In Object-Oriented programming, the objects are based on a **class** or template. In this section we take a look at the components that make up an Object-Oriented Program

The Class

❑ The mechanism VB.NET provides to implement *Objects* is the *Class*.
❑ A *Class* is a template or blueprint that defines what *Object* of the class look like.
❑ A *Class* is a plan or template that specifies what *Data* , *Methods* & *Events* will reside in *objects*
❑ The objects of the class contain *data* and the *Methods* (member functions & procedures) that operate on such data
❑ When creating a Class Module, the *Data* is made *Private*, & the interface or method to access the data (Procedures & Functions) are *Public*.
❑ For example:

- We can have a Class called *Automobile*, and from this class, we can define the Properties, Methods and Events of this class.
- From this Automobile class we can create Objects of this class such as a Car Object, Truck Object, SUV Object etc.
- The objects created have all the properties, methods and events dictated by the Class from which they were created from.

Objects

❑ Think of Objects as a <u>thing</u> or a <u>noun</u>. Objects are the items that represent real-world entities, such as a person, place or thing in a program.
❑ In a program an *Object* is a software representation of a real-world entity.

Objects - vs - Class

- The concept of a Class an Object can be very confusing. A Class is NOT an Object. An Object is not a Class
- DO NOT confuse a *Class* with the *Objects*, they are two different things.
- *Objects* are the manifestation or *instance* of a *Class* specification.
- A class **IS NOT** the object, but the template in which *Objects* will be created from!
- **Think of the <u>class</u> as the architectural floor plan of a house, and the <u>objects</u> as the houses that are built from that plan**. You create ONE floor plan or blue print, but you can create as many houses as you like from the blue print.
- Objects behave exactly as they were specified in the Class. No more, no less

Private Data

❑ Data is the storage mechanism inside the object to store information.
❑ Data is what we want to manipulate and protect.
❑ For example, a person has a name, an ID, birth date etc. These entities are stored and preserved INSIDE THE OBJECT.
❑ In a class Object, **Data** is Private and cannot bee seen by the outside world..

Public Properties (Attributes)

❑ An Object has characteristics. Such characteristics or attributes are called properties of an Object. For example a Person Object has a name property, a social security property, a birth date property etc.
❑ Properties represent the *Data* of the Object. **DO NOT CONFUSE THE PROPERTY WITH THE DATA**. They are two different things.
❑ In reality, the Property is the way the outside world access the actual data directly.
❑ This is confusing; the property is not the data, but a vehicle to access the data. The actual data is private and cannot be seen by the outside world, only the properties are seen by the outside world because they are **PUBLIC**.

❑ For example from an *Automobile* Class, you may create an Object named **objCar**. The Automobile Class may have a *color* property, as well as a *Make* & *Model* property. But inside the Data is what stores this information. This is done using private variables inside the class. For example these variables can be named *m_color*, *m_make* & *rm_Model* etc. but the outside world cannot see these variables, when they want to use the data they see the property *Color*, *Make* & *Model* and through these properties the data are accessed.

❑ **Properties** are the vehicle in which you can **SET** (write) or **GET** (access) the **DATA**!!!!!!!

❑   Syntax for using an object **Properties** is based on the DOT OPERATOR:

    Object.*Property*

❑   Example, assuming you create an object named **objCar** from the *Automobile* class, writing and accessing data is done as follows:

Purpose	Syntax	Example
SET or write data	**Object**.*Property = value*	**objCar**.*Make = "Acura"*
GET or access data	*value* **= Object**.*Property*	*aStringVariable = **objCar**.Color*

Methods (Behavior)
- ❑ Objects have **behavior** or take **action**.
- ❑ **Methods** are <u>actions</u> that the Objects can take. Where Objects are the Nouns, Method are the verbs or actions of an Object.
- ❑ For example a Car Class Object can take the following actions: *Start*, *Stop*, *Speed Up*, *Slow Down*, *turn left*, *turn right* etc.
- ❑ Methods are implemented in a class by creating ***Functions*** and ***Sub Procedures*** that you write to make the object do things or take some kind of action
- ❑ Syntax for using an object **Methods** uses the DOT OPERATOR as well:

    Object.*Method()*

- ❑ Example, using the **objCar** from the *Automobile* class:

Purpose	Syntax	Example
Executing or telling the Car object to take an action such as stopping	**Object**.*Method()*	***objCar**.Stop()*
Executing or telling the Car object to take an action such as starting the car	**Object**.*Method()*	***objCar**.Start()*

Events
- ❑ This is a tough one to explain and understand!
- ❑ **Events** are *actions* taken UPON the object by an outside force (User, Program code etc).
- ❑ These actions or **Events** upon the **object** will automatically **trigger** specialized Methods *<u>automatically created outside of the object</u>* known as *Event-Handlers*. In other words, Objects respond to *<u>events</u>* by executing this special method or procedure know as an *Event-Handler*
- ❑ Do not confuse **Events** and *Event-Handlers* with regular Methods. Events are the action taken by an <u>outside</u> source upon the object, while Methods are action taken by the Object itself when told.
- ❑ **Events** & *methods* may work hand in hand, but they are two different things.

  - ▪ This can be confusing. For example an Object such as the **objCar** Object can have a method called *Stop()*. You can explicitly call the *objCar.Stop()* method to so the car will stop itself.
  - ▪ On the other hand, The Car Object can also have an **Event** programmed into it called *OnCrash* which will *create outside the object* and associated *Event-handler* named ***objCar_OnCrash()***.
  - ▪ In the **event** that the car is hit by another car or crashes, the *OnCrash* event will automatically trigger or EXECUTE the Event-handler ***objCar_OnCrash()***. Inside the ***objCar_OnCrash()*)** *Event-Handler* you can code in what ever you like. For example you may want to put in a statement that calls or execute the ***objCar.Stop()*** method to stop the car,

8

or call 911 or what ever action you seem fit for this event. Makes sense right? Here an Event occurs, triggers the Event-handler, in the Event-Handler we call a Method to stop the car. Confused yet? ☺

1.2.2 Creating Object-Oriented Programs (IMPORTANT!)

❑ Object-Oriented Programs (OOP) are written based on the Class Objects and not on the functionality of the program
❑ The following steps is what you need to do every time you create an Object-Oriented-Program
❑ The three steps required to creating an Object-Oriented Programs are shown below:

I.   **Create and Define the class specification or Class**
  ▪ Define Private Data, Properties, Methods & Events

II. **Create Object of the Class**

III. **Use the Object of the Class**
  ▪ Write the program to manipulate, access or modify the objects as follows:
    ▪ **Get and Set Properties (Manipulate the data)**
    ▪ **Call Methods**
    ▪ **Trigger Events**
    ▪ **Program Event-Handlers**
    ▪ **Interact with other objects**

1.2.4 Object-Oriented Programming and Graphical Elements (Forms & Controls)

❑ At this point you have a basic understanding of a **Class**, **Object**, **Properties**, **Methods** and **Events**
❑ Also, you have taken courses (CS101 & CS508) in which you crated VB.NET programs using Forms and graphical controls such as Text Boxes, Buttons, List Boxes, Labels, etc.
❑ You wrote graphical programs using the following steps:

  1. Create a Form
  2. Dropped Graphical Controls onto the Form (Labels, Text Boxes, Buttons, List Box etc.) to create your GUI
  3. Added programming code to the Event-Handlers of the controls. For example if you have an OK button, you added code to do something on the Event-Handler: *btnOK_Click()*
  4. You build & compiled the program
  5. Executed the program

❑ This process just described is known as *Event-Driven Programming*, as stated in section 1. Why? Because you are simply programming the *Event-Handlers* of the Graphical Controls, which execute as a reactions to Events on the Controls.
❑ What is the point?

  ❖ It turns out that Every Graphical Element in VB.NET, such as Forms, Controls etc. Are all **OBJECTS**!
  ❖ And if they are **Objects**, a **Class** exits for them.
  ❖ The **CLASS** was **CREATED** by **Microsoft**. That's right!
  ❖ And every time you drag and place a control to a Form object **YOU** are actually **CREATING** an **OBJECT** of that Control.
  ❖ The Form is an Object and the Controls are now child Objects of the Form (Object Interaction).
  ❖ Every Time you set or get a property using the *Property Window*, you are setting or getting a property of the Control Object, so you are **USING** the object.
  ❖ Every time you placed code such as: txtBox.Clear() you were executing a Method.
  ❖ And finally, when you are placing code inside the Event-Handler, you were telling the Event-Handler what to do when the Event takes place or Event-Driven Programming.

❑ So as you can see, all along you have been applying the rules to create classes, create objects and use them as follows:
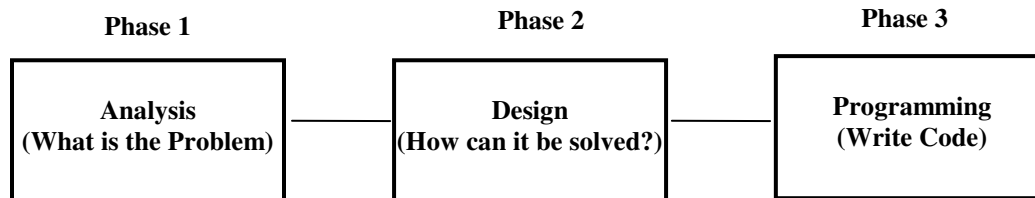

  I.   **Create** and **Define** the class (Done by Microsoft)

  II.   **Create** Object of the Class (Doe by YOU when you dropped a control on a Form)

  III. **Use** the Object of the Class (Done by YOU)
      ▪ Get and Set Properties (Using Property Window)
      ▪ Call Methods (Inside the Event-handler code)
      ▪ Trigger Events (Every time you clicked on an OK button, lost focus, etc)
      ▪ Interact with other objects (Drag-Drop controls onto Forms etc)

❑ So all along you were programming using OOP techniques. But Microsoft made it easier for you by creating all the classes and providing an interface or IDE to make it visual and simply the process.

❑ In this course (CS608), you will perform all the steps. You will creating custom classes, creating the objects and using them.

1.3 Object-Oriented Analysis, Design & Programming

1.3.1 Analysis and Design

❑ If you recall in our introduction to Object-Oriented Programming, I used the *Video Management* Program as an example to demonstrate the difference between *Procedural*, *Event-driven* and *Object-Oriented Programming*.

❑ Note that in the beginning of each of the example, the first step was as follows:

    1. Analyze the problem required to be solved: Design flow chart, algorithm etc.

❑ In OOP, there are three phases or steps required for developing an Object-Oriented System:

Phase 1	**Phase 2**	**Phase 3**
Analysis **(What is the Problem)**	**Design** **(How can it be solved?)**	**Programming** **(Write Code)**

❑ Analysis and Design are very important steps. This is where all the "brain work" is done to developing the system and implementing the algorithm.

1.3.2 Program Development Cycle

❑ The program development cycle refers to the steps or process required to create an application from start to finish.
❑ The process involves first understanding what the problem is that you are required to solve, come up with a design or idea on how to solve the problem and finally choose a programming language to implement the project.
❑ The programming language is simply the tool to create the actual program code. You can choose any language you wish to achieve the final results. The difficult part is in the thought process or design to solve the problem.
❑ Remember the following:

** IS NOT THE LANGUAGE THAT MATTERS BUT THE <u>SKILL</u> OF PROGRAMMING**

❑ Knowing how to program is what's important, not what language you know. Programming languages evolve and change, and new and more powerful languages are being developed. Once you have the SKILL of programming, you can tackle any language, simply learn the new syntax or rules to a new languages and you should be able complete your project.

Visual Basic Solution & Project
❑ When you write a program is basically done to solve a problem.
❑ Solving the problem results in a solution to the problem. A VB application is called a **Solutions** & a solution is composed of one or more **Projects**.
❑ In this **CS608** and **CS708**, you will be creating *Solutions* that can contain one or multiple *projects*.

Creating an Applications to solve a problem

❑ Creating an Application involves a Two Part process, I) **Planning** II) *Design* III) **Programming**.
❑ Each Part is broken into three phases or steps:

I. **PHASE 1 – System Analysis (Understanding the Problem):**

1. Analysis means to study, understand and define the ***requirements*** of the system.
2. Read and understand the problem description.
3. Write down the requirements and fully understand what is being asked for you to do.
4. Identify what are the **INPUT** or data to be entered or manipulated by the program
5. Identify what are the **OUTPUT** or results that are required.

II. **PHASE 2 – Design (Solving the Problem):**

1. Design means developing or creating the solution to the problem. This is a thinking process and the solution is derived from the **analysis** of the requirements.
2. From the analysis of the INPUT & OUPUT we can derive the required processing.
3. You can use tools such as:

    o *Design the solution –* Use diagram, flow charts etc., to analyze and design the solution.

4. *Plan the code (Problem Solving)* – This step you will write down the action required to solve the problem, this is known as the *Algorithm*. The *Algorithm* is obtained using the following tools:

    ▪ **Flow Charts** – Graphical representation of the program logic
    ▪ **Pseudo-code** – Abbreviated short-hand English-like statements representation of the flow chart

Wow!! this is Too Much THINKING!

5. In this phase we design on paper (Not in the computer) the Forms and User Interface required.

    o *Design the User-Interface (GUI) –* You draw a sketch of what the Front-End or GUI will look like and the Control Objects required.

    o *Plan the Properties or attributes to the objects in the GUI* – In a table write down the properties to the Control Objects in the GUI.

III. **PHASE 3 – Programming (Writing the Code):**

1. Here is where we use VB.NET to create our application.
2. *Create* the User-Interface (GUI)

3. *Set* the Properties or attributes to the objects in the GUI

4. *Write the code* – Use the syntax or programming code of the programming language to create the program.

1.3.3 Designing the Code – Creating an Algorithm

Problem Solving via an Algorithm

❑ We now focus on PHASE 2 of our program development cycle, which is planning the code or how to solve the problem. **This is the most difficult part of creating an application since it requires thinking logically**.

❑ We will start with the definition of an *Algorithm:*

> *Algorithm*: **a logical <u>sequence</u> of steps or <u>actions</u> executed in a particular <u>order</u>**

❑ The *Algorithm* is the collection of logical steps require for the solution of the program. The algorithm is what we will derive during the planning phase.
❑ We will use the development tools describe in previous lectures to derive the algorithm:

- ▪ **UML**– A graphical diagrams of the CLASSES which will be used to create the OBJECTS of the program
- ▪ **Flow Charts** – A graphical representation of the algorithm
- ▪ **Pseudo-code** – Short-hand English-like statements. Pseudo code programs are not executable code, but they help programmers **"THINK OUT"** before attempting to write it in the actual programming language

❑ *Plan the code* – In this step, you plan (THINK) the code or steps required for the program to run. You will write down the action required to solve the problem. You will use **programming tools** like *UML* to design your classes and work flow, **pseudo-code** and **flow charts** to plan the necessary <u>logic</u> to solve the problem. This is really the tough part of programming since it requires thinking logically

❑ How the algorithm is written can affect the results or solution to a problem.
❑ For example, supposed you were asked to develop an algorithm name "*rise and shine*", which lists the steps for a manager to get out of bed and get to work. The *pseudo-code* for the algorithm may be as follows:

1. Get out of bed
2. Take off pajamas
3. Take a shower
4. Get dressed
5. Eat breakfast
6. Drive to work

❖ This algorithm gets the manager to work and prepared to make critical decisions.

❑ Now lets change the **program control** or **order** in which the pseudo codes are written:

1. Get out of bed
2. Take off pajamas
3. Get dressed
4. Take a shower
5. Eat breakfast
6. Drive to work

❖ This Algorithm gets the manager to work wet and probably not in a condition to make critical decisions.

Recommendation on how to Create Algorithms

❑ Thinking like a programmer is not easy, but with a problem solving strategy in place and practice you will get better at it.
❑ One method I can recommend to get you started when writing programs is as follows:

- Read the requirements several times until you understand what is required of the program.
- Now put yourself in the role of the computer itself and what is require of you to do, and then switch to the role of the user is standing in front of the computer to use it and what is required of the user to do.
- As yourself questions as to what is required for each of these characters during their interaction.
- Now you write down the steps of the interaction which takes place between the program and the user. Now extract the performed by the computer program, put these steps in their proper order and you have an algorithm and an idea of what needs to be done
- For example, supposed you were asked to create the following program:

Problem Statement:
- Create a login program to authenticate users (similar as to when you log in to your computer). The program should have a Login Form with controls to allow the user to enter the username and password. In addition should have a button to execute the request or cancel.
- The program should search a database of username and passwords to verify if this user/password combination is allowed access to the system if found in the database.
- From result of the database search allow access to the system or display a message stating that access is denied.

- The interactions or exchange between the computer and the user is as follows:

What is the First thing the User does?
    a) Sits in front of the computer

What should the Computer do?
    a) Display the Login Screen

What should the User do?
    a) Enter username
    b) Enter password
    c) Click Ok button or Cancel button

What should the Computer do?

If you user clicks the **OK** button:
    a) Extract the username
    b) Extract the password
    c) Search the database for the username and password
    d) If the username and password is found in the database:
        1. Allow access to the system
        2. Display a welcome screen

    e) If the username and password is NOT found in the database
        1. Display a message stating that access is denied
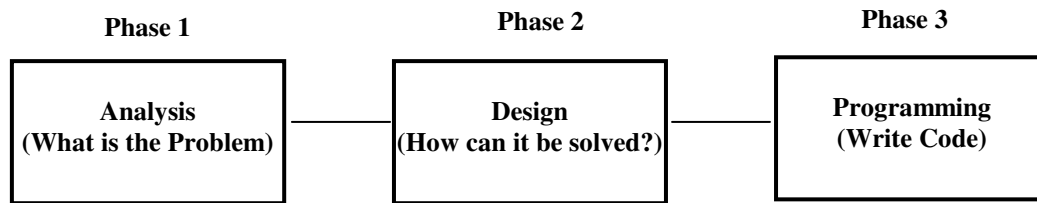   2. Go back and display the login screen again.

If you user clicks the Cancel button:
    a) Display a message stating that the user cancelled the operation
    b) Clear the text boxes

- The *algorithm* is derived by extracting ONLY the steps performed by the *Computer Program*:

    1. Display the Login Screen
    2. If you user clicks the OK button:
        a) Extract the username
        b) Extract the password
        c) Search the database for the username and password
        d) If the username and password is found in the database:
            1. Allow access to the system
            2. Display a welcome screen

        e) If the username and password is NOT found in the database
            1. Display a message stating that access is denied
            2. Go back and display the login screen again.

    3. If you user clicks the Cancel button:
        a) Display a message stating that the user cancelled the operation
        b) Clear the text boxes

1.3.4 Summary of Strategy for Developing the Algorithm

❑ Ok, lets come up with a strategy to help us learn to think and develop the algorithm.
❑ The strategy to solve the problem will require seven steps as follows:

Phase 1   **Phase 2**   **Phase 3**

Analysis **(What is the Problem)**	**Design** **(How can it be solved?)**	**Programming** **(Write Code)**

I.   **PHASE 1 – System Analysis (Understanding the Problem):**

     1.   *Problem:* Write down the problem statement
     2.   Understand the problem. Make sure you understand what are the inputs and outputs expected.

II.   **PHASE 2 – Design (Solve the Problem):**

    1. *Discussion:* Think! If it helps, write down your thoughts on what the problem is and what it takes to solve it

       How am I going to do this? LET'S THINK

    2. Ask yourself, what is the *input*?
    3. What is the *output* desired?
    4. Try the technique I recommended of listing the expected interactions between the user and the computer
    5. List what *processing* is required to obtain this output, such as classes or objects, user-interface etc.
    6. Create Algorithm using, **UML** for illustrating the object model , *Flow Charts* or/and write down the *pseudo-code*
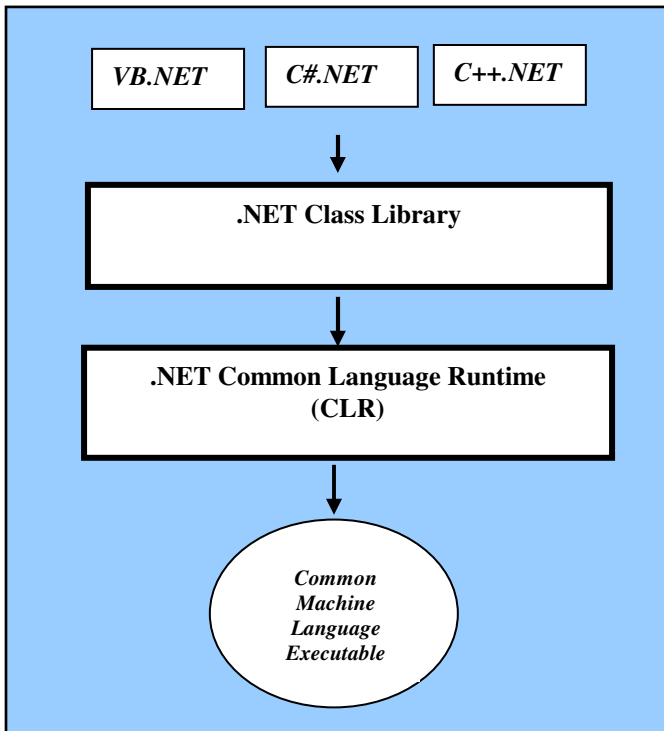
III.   **PHASE 3 – Program (Write the Code):**

    1. *Create* the Classes
    2. *Create* the User-Interface (GUI) using a Graphical programming language such as Visual Basic.NET
    3. *Write the code* – Use the syntax or programming code of the programming language to create the program based on the *UML diagram*, *Algorithm's pseudo-code*. In other words begin to create the **OBJECTS AND USE THEM!!!**

Chapter 2 Visual Studio.NET Development Environment

2.1 Microsoft .NET Framework and Visual Studio.NET

2.1.1 Microsoft .NET Framework

❑ The *Microsoft.NET Framework* is the new computing platform from Microsoft designed to simplify the development for distributed environment such as the Internet.
❑ The framework is designed from the grounds up with the Internet in mind.
❑ It is not that this Framework was designed for Internet programming only, but simply that if an application you create needs Internet capabilities, access to those capabilities are available and almost transparent.
❑ The .NET Framework is composed of two main components shown in the figured below:

```
┌─────────────────────────────────────────────────┐
│                                                 │
│  ┌──────────┐   ┌──────────┐   ┌──────────┐     │
│  │ VB.NET   │   │ C#.NET   │   │ C++.NET  │     │
│  └──────────┘   └──────────┘   └──────────┘     │
│                      │                          │
│                      ▼                          │
│  ┌───────────────────────────────────────┐     │
│  │         .NET Class Library            │     │
│  └───────────────────────────────────────┘     │
│                      │                          │
│                      ▼                          │
│  ┌───────────────────────────────────────┐     │
│  │   .NET Common Language Runtime        │     │
│  │             (CLR)                     │     │
│  └───────────────────────────────────────┘     │
│                      │                          │
│                      ▼                          │
│                  ╱───────╲                      │
│                 │ Common  │                     │
│                 │ Machine │                     │
│                 │Language │                     │
│                 │Executable│                    │
│                  ╲───────╱                      │
└─────────────────────────────────────────────────┘
```

1. **.NET Common Language Runtime or CLR**
    - The CLR is the key component of the .NET Framework. It is a common compiler for all Microsoft programming language. It compiles all Microsoft languages to one *Machine Language*.
    - The CLR Allows programmers to write code in different Microsoft languages of their choice, and ensure that the parts can work together
    - In addition it provides programmers with *services* for memory management, data types, security etc.

2. **.NET Framework Class Library.**
    - A collection of ready-made *reusable* classes that work with the CLR.
    - Programmers using any of the .NET programming languages can use these classes to include in their applications.
    - The CLR Allows programmers to write code in different Microsoft languages of their choice, and ensure that the parts can work together
    - Examples of the classes provided by the library are:

        o Database access and manipulation
        o User Interface – Windows Forms, Web Forms, Web Services
        o Security
        o Encryption and decryption

2.2 The Visual Studio.NET Environment & Visual Basics.NET

2.2.1 Introduction

- ❑ We will use Visual Basic.NET *Integrated Development Environment* or *IDE* to program our applications.
- ❑ We could choose any of the languages to create our application such as C#, VB.NET etc., but this course requires that we use **Visual Basic.NET**.
- ❑ Before we begin using the IDE, lets point out some of the types of applications we will be creating in CS608 & CS708:

  - ▪ **Console Application** – Usually Text only application. Runs from a DOS or COMMAND PROMPT screen.
  - ▪ **Windows Application** – Graphical Interface user application. Typical Windows application that you normally use, such as MS WORD etc.
  - ▪ **Web Based Windows Application** – Web application that uses a Browser such as Internet Explorer.

- ❑ Let's look at these type of applications in more details

  1. **Console Application**
     - ▪ A *Console Application* is a program whose output is usually text based (No Graphics).
     - ▪ Console Applications usually do not contain Forms or any graphics, but they can. You can if you like from a console application call Windows Forms etc.
     - ▪ Console Applications are created when an application performs processing that requires very or no user interaction.
     - ▪ They are lighter and have less overhead than standard windows applications since they contain no graphical libraries etc.
     - ▪ Console applications are a good choice when creating programs such as login scripts, device drivers, backend processes, test programs, programs that control hardware devices etc.
     - ▪ The results of a Console Application is placed or controlled from a **Command Prompt Window**:

  2. **Windows Application**
     - ▪ Windows Applications are your standard graphical applications we are used to using.
     - ▪ Windows Apps use graphical entities such as Forms, Web Forms etc.

3. Web-Based Application
- These are applications created for the World-Wide-Web or Internet.
- These applications run from a Browser, but the actual program code for these applications (HTML) reside on a Web Server and are distributed to any client or browser which makes the request.

Browser

Web Page or Web Application

2.2.2 Creating Project Using the Integrated Development Environment (IDE)

❑ The steps to using the *IDE* to program our applications as follows:

Step 1: Open the *Visual Studio IDE* and invoke the *Start Page*:

Step 2a: In the *Start Page*, select *New Project*

Step 2b: In the New Project Dialog select Application Type

a). In the *Project Types* box select: *"Visual Basic Projects"*
b). In the *Template* box select: *"Windows Applications"* or "*Console Application*"
c). Enter the project name into the *"Name:"* text box
d). Set the project path or location in the *"Location:"* text box. You can also browse for the path using the *"Browse"* button.
e). Click *OK*

Step b – Windows Application

Step a – Visual Basics Project type

Step c – Name the Project

Step d – Project Path or Location
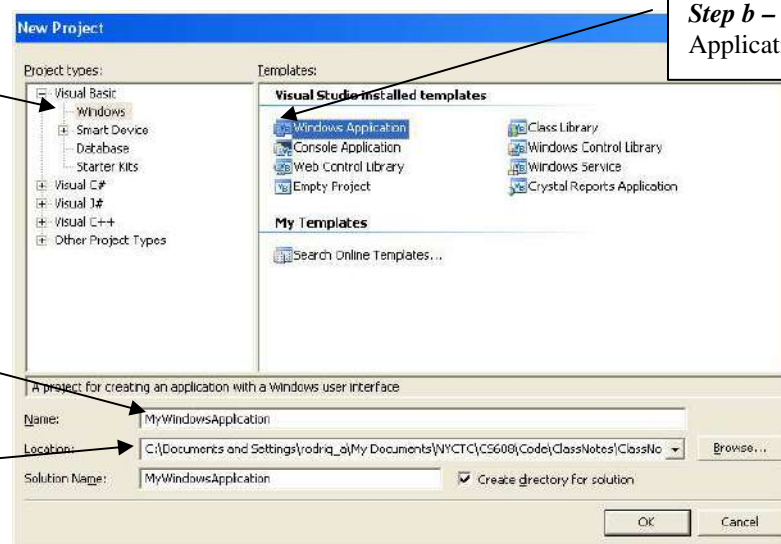
❑ Example of selecting Console Application:

Console Application
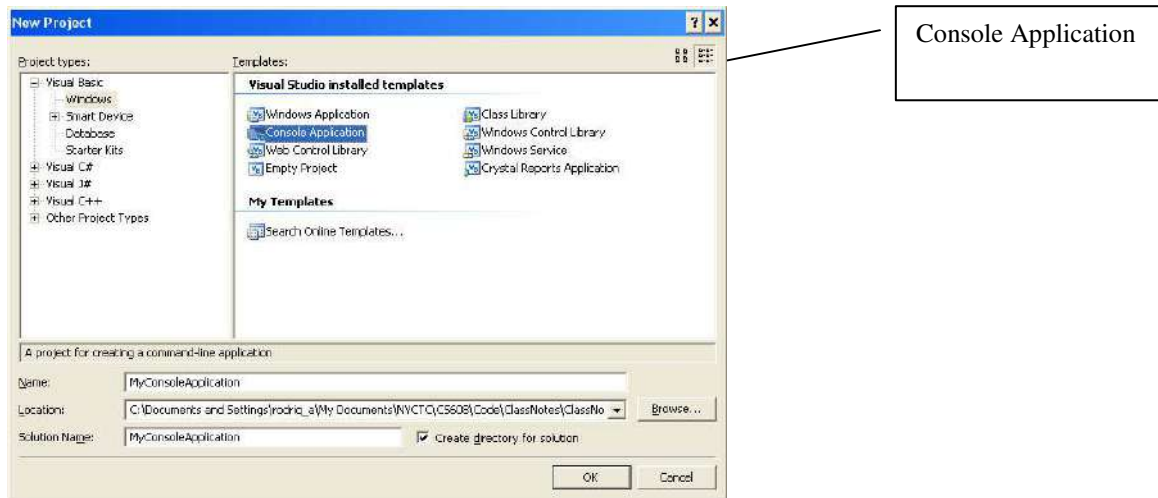
Step 3: The IDE Main Screen.

- This screen is where you will create all your programs.
- This window will vary depending on which type of application you have selected, *Console Application*, *Window Application* or *other*.

Part I - Main Screen for a Console Application:

- This screen is composed some basic Window items such as *Title bar*, *Menu bar*, *Menu Toolbars* and a *Status Bar*.
- In addition some new components that will be important for creating applications, such as *Document Window*, *Solution Explorer Window*, and *Properties Window*.
- A Console Application will automatically create a **Module** document in the *Document Window*.

Module

Solution Explorer

Property Window
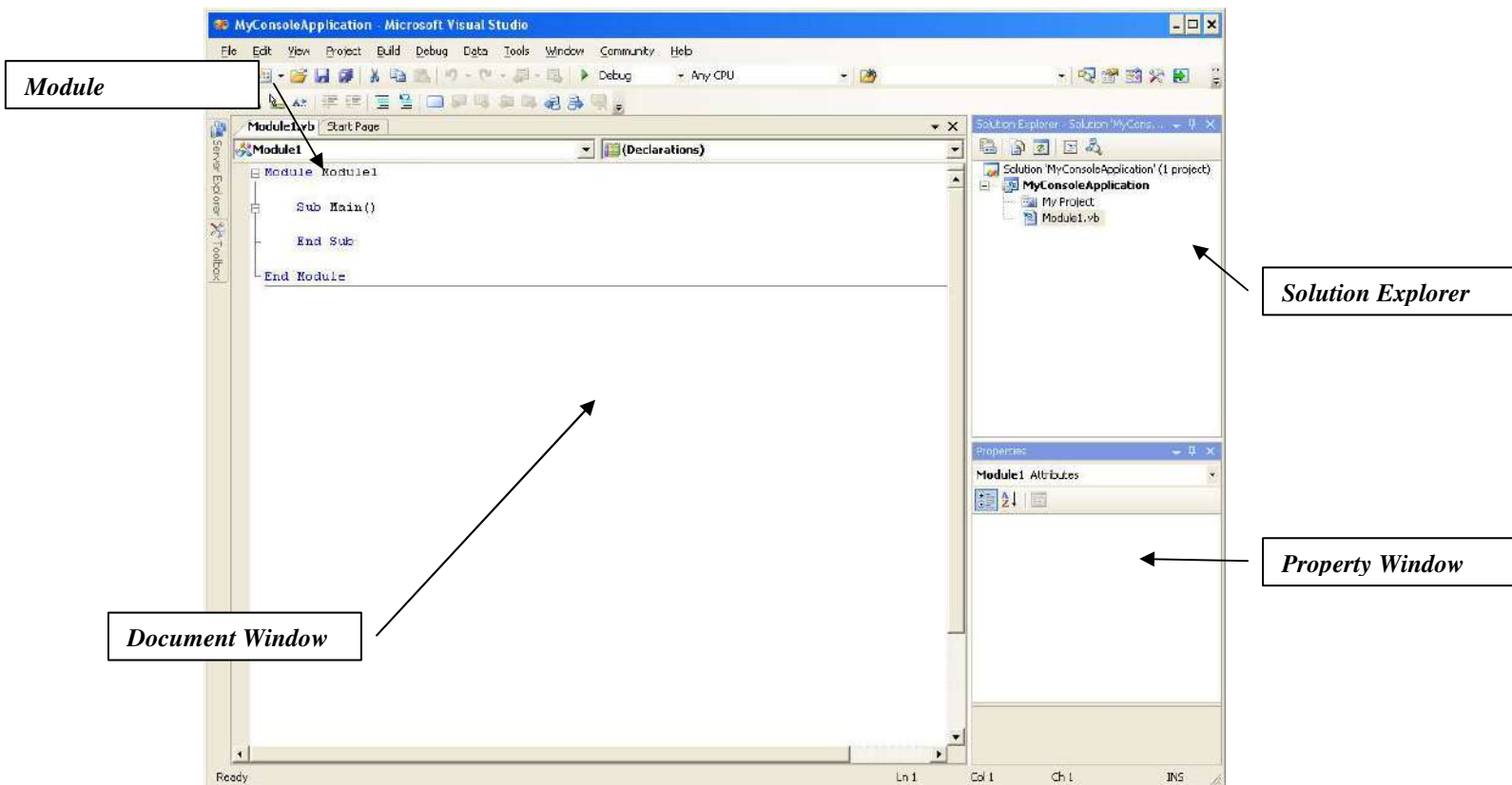
Document Window

Part I - Main Screen for a Windows Application:

- ▪ This screen is composed some basic Window items such as *Title bar*, *Menu bar*, *Menu Toolbars* and a *Status Bar*, *Document Window, Solution Explorer Window*, and *Properties Window*
- ▪ In addition some new components that will be important for creating Windows Applications, such as a *Form Designer* to create the Forms and User Interface. Also the *Toolbox* which contains the controls to create the UI.

Form Designer

Toolbox

Solution Explorer

Property Window

Document Window
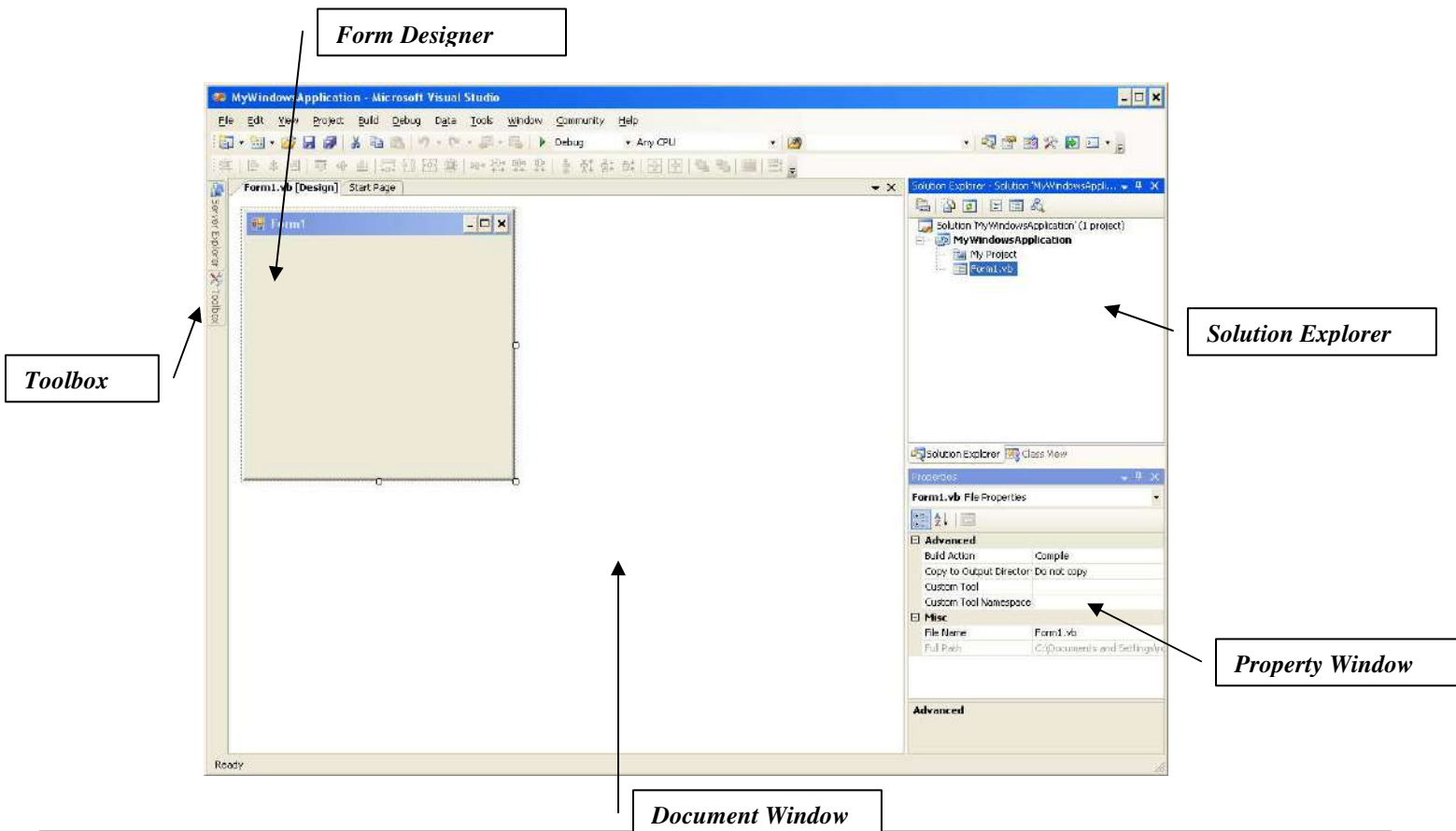
Form Designer

- ❑ This is the where you will design you Forms as a basis to your *User Interface (UI)* or *Graphical User Interface (GUI).*
- ❑ When you begin a new Visual Basic project, a new form is automatically added to the project with the default name **Form1**
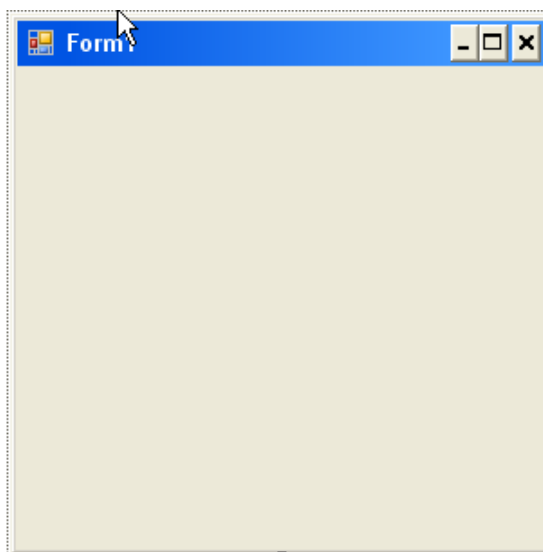
Toolbox

❑ The Toolbox is a palette that holds the control objects you will place on the Forms.

Control Objects
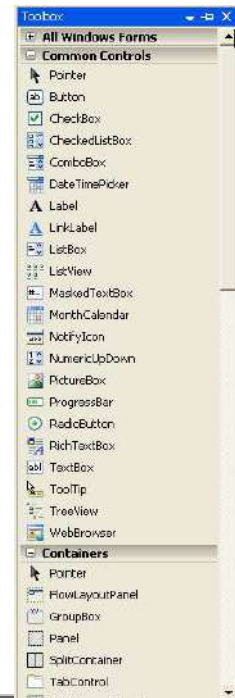
Solution Explorer

❑ Displays the *Files*, *Forms*, *Modules* and *Class* Objects included in the project.
❑ The Solution Explorer has two views:

1) *Solution Explorer View -* Files are displayed
2) *Class View –* Class Objects and their methods & events are shown
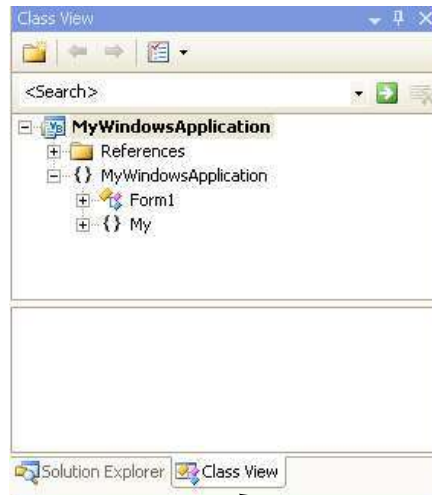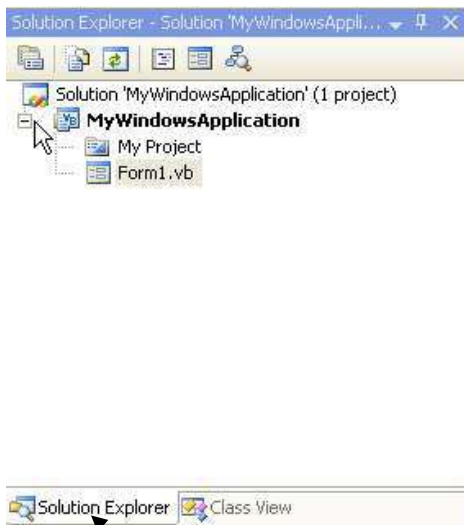
Class View

Solution Explorer View

Property Window

❑ This window is used to SET the Properties of the objects in your project.

❑ You can view the Name of the property in the *Name column*, and set the property value in the *Value Column*.
❑ Note that these objects include the *controls*, *Forms*, *Class Objects* and other objects used throughout your project.
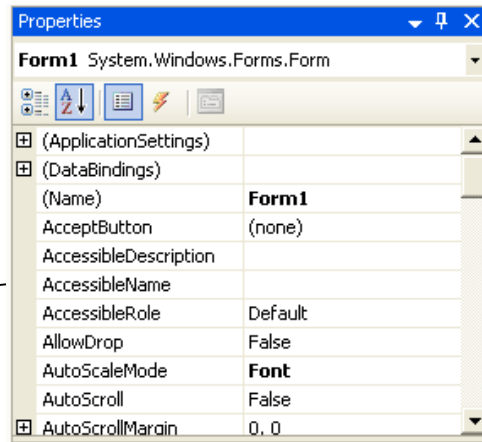
Property Value

Property Name

Document Window

❑ Largest window in the center of the screen. Allows you to switch between the open documents.
❑ For *Console Applications*, this window contains a *Module* & *Code Editor*.
❑ For *Windows Applications*, this window contains the *Form Designer* & *Code Editor*.
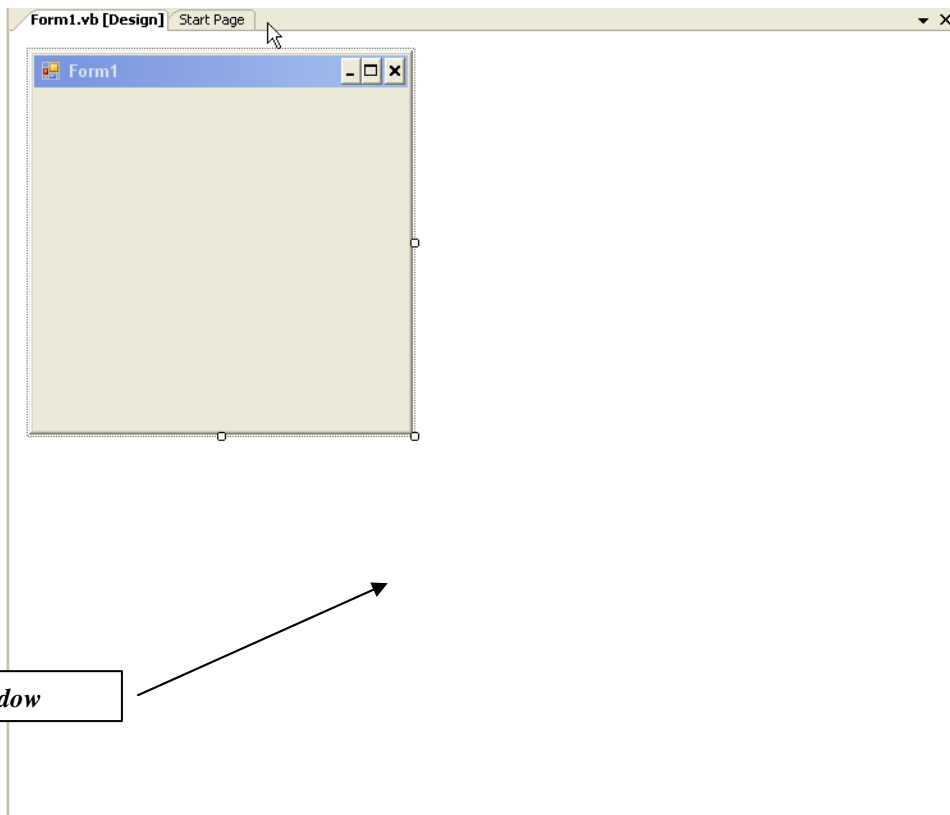
Document Window

Step 4: Set the Project Properties (StartUp Object)

❑ So far we have talked about the Properties of Forms and Control Object. Now we focus on the properties of the **Project** itself.
❑ In Visual Basic you can set properties for the entire project. Some of these properties you have already populated in the *New Project Window* in Step 2.

❑ The *Project Properties Window* is invoked as follows:

- In the Menu Bar select **Project**|*NameOfProject* **Properties** to invoke the *Project Property Page*
- Another Method uses the **Solution Explorer**. Simply *Right-Click* on the *project name* and select *Properties* from the context menu to invoke the *Project Property Page*

Startup Form/Startup Object
❑ There is one very important Project Property that must understand prior to writing code. That is the **Startup Form or Startup Object**.
❑ The **StartUp Form** or **Startup Object** is the EXECUTION starting or entry point of a program.
❑ A Visual Basic Application can be started using the following two options:

I. **FORM** – (Default) this is the default method you are accustomed to, program starts via a FORM:

- What this means is that when the project executes the Startup Form will display and control the flow of the program
- Control or flow of program is done via a Form. In a standard *Windows Application*, this is by default automatically assigned to **FORM1**
- I call this method of starting a Windows Application as a *Form-Driven Application*
- For this method of executing the project, the property is named **Startup Form**
- NOTE! There is a checkbox for a property named *Enable application framework* in the property page. This property when checked, displays the startup option as *Startup Form*.   This property is checked by default, so you don't need to do anything, but if you uncheck this box, it will show as Startup Object (More on this below).
- The figure below shows the default startup option as *Startup Form*:
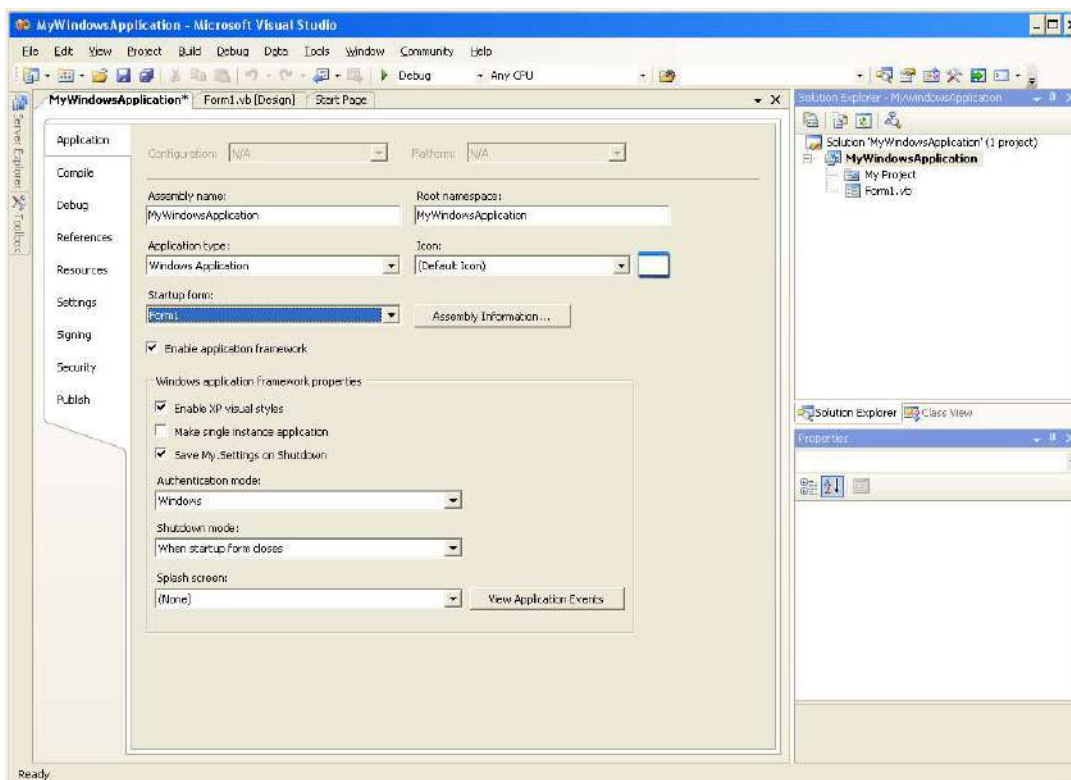
II.   SUB MAIN – You can also have your program start execution via a special method named **Sub Main()**.

- In this case, the method **Sub Main()** is created in a MODULE object (More on this later). Program execution is done from within the module's Sub Main() method.
- I will not go into details about the module now, but there is a Visual Basic Object named a **Module**.
- A Module is simply a special Code Editor Screen that allows you to enter code only. It has no graphical portion.
- The code inside a module can be seen by all objects of a program. We will go into this in future lectures
- A module can contain a special Method named **Sub Main()**. This **Sub Main()** Method will control the flow of the program.
- You must display your *form* via code within this **Sub Main()** method.
- I call this type of application a ***Module-Driven Windows Application***. This method has the advantage that you can execute other processes in your application before any forms are displayed.
- For this method of executing the project, the property is named **Startup Object**.
- NOTE! The checkbox for a property named ***Enable application framework*** in the property page MUST BE UNCHECKED, for the startup option to be named **Startup Object**, and you select **Sub Main()** in list box as shown in figure below:
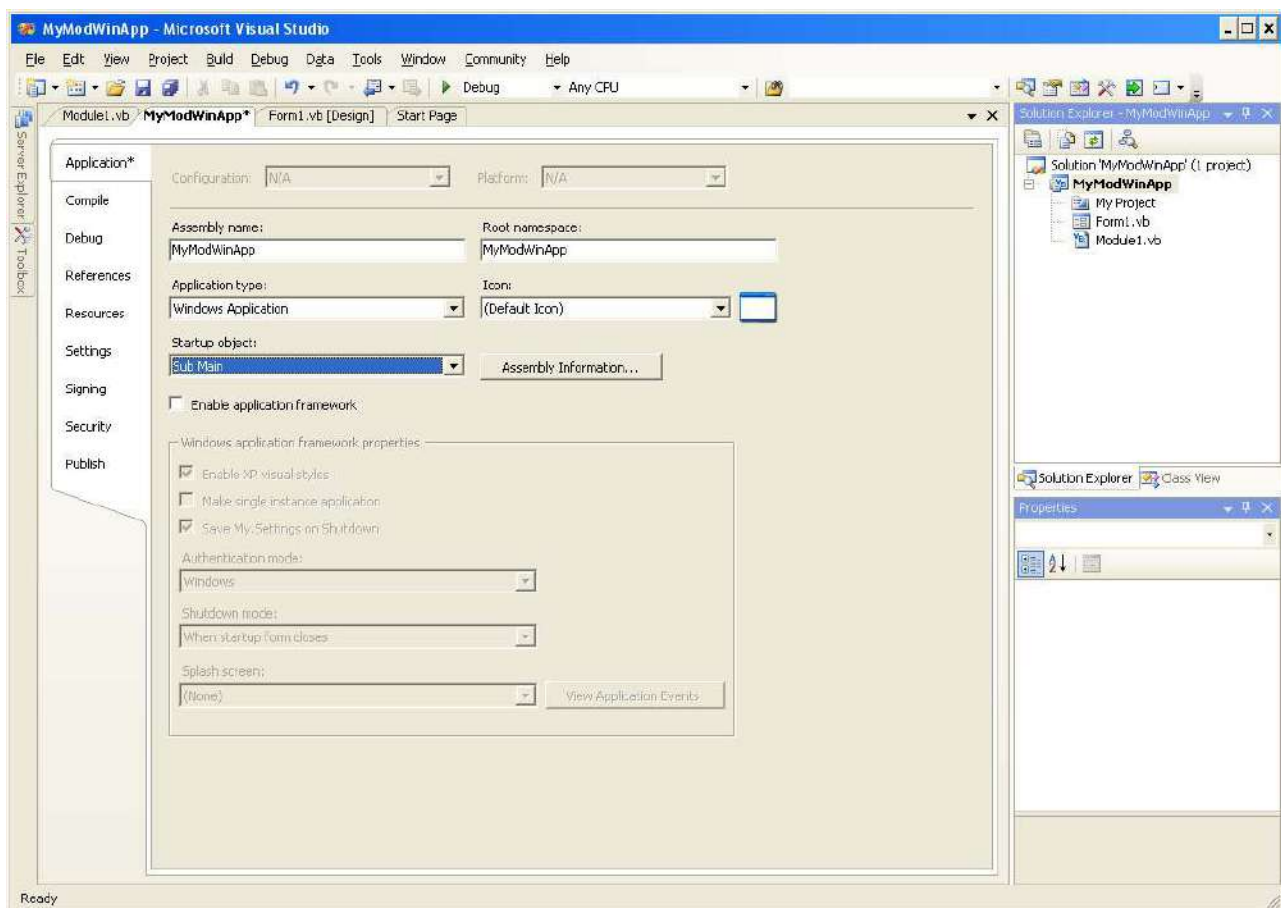
## Step 5: Create your program using the components of the *IDE Main Screen*.

❑ Now you can begin to write your code and create the program using the IDE:

1. *Create* the User-Interface (UI) using the *Toolbox* and the *Form Designer* to drop controls onto the Form.

2. *Set* the Properties to the Control Objects in the Form using the *Property Windows*.

3. *Write the code* – Write the code to manipulate the Controls via the *Event-Procedures* and *Method*s using the *Code Editor Screen*.

Visual Basic Standard Modules

❑ A module is a global or public object where you can place program code such as variables, functions & procedures that can be seen by all code in the entire project.
❑ The key here is Global code!
❑ Up to now, all code written was contained inside the Event-Handlers of various controls on a form.
❑ As controls & objects are placed on a form, all code written is usually seen & relates specifically to that form.
❑ Suppose you wanted to write code that can manipulate or operate on various forms & controls. How can you do this? Since every code you write for a control or form is only visible within the form? Where would you place the code? The answer is the **Module**!
❑ The Module has the following characteristics:
   ▪ **Modules** contain a method named Sub Main() that can be used as the starting point of the application. Note this applies to both *Console Applications* and *Windows Applications*.
   ▪ **Modules** contain variable declarations, functions & procedures or source code ONLY! No forms!
   ▪ **Modules** are a good place to put program code that may be common to several forms or other modules.
   ▪ **Modules** contain variables & Methods.
   ▪ **Modules** are like forms but without the visual!

Module in Console Applications

❑ When you create a *Console Application*, Visual Basic automatically creates a Module where you can begin entering your code:

Module File

28
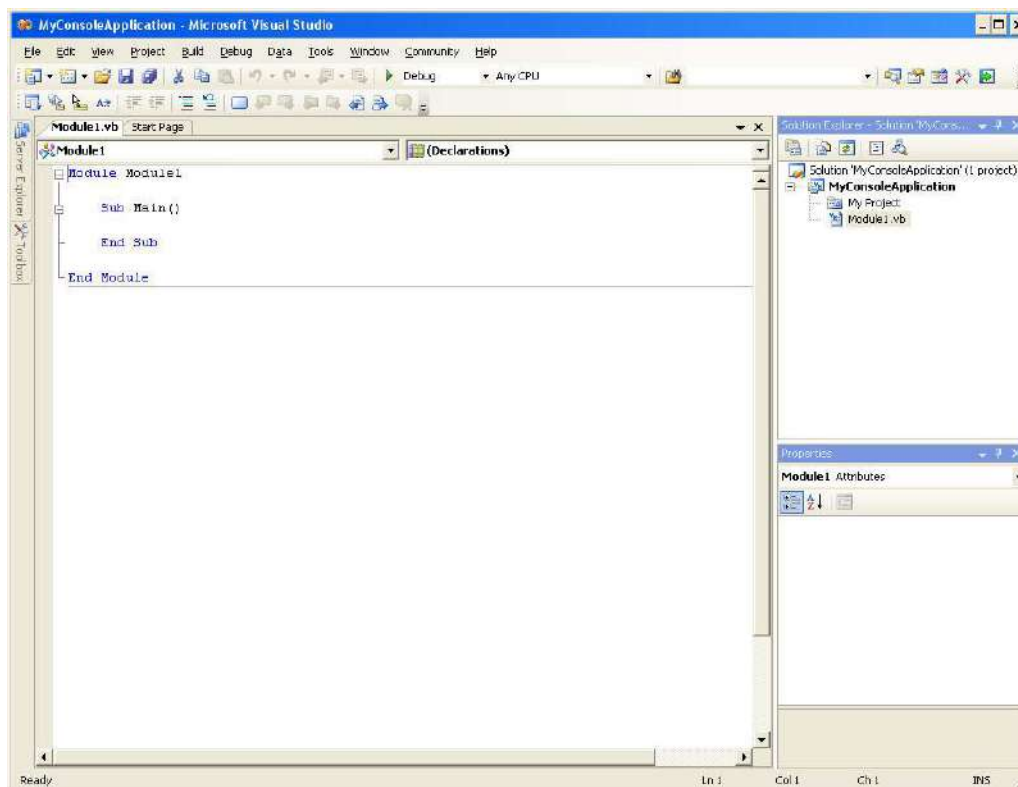
Module in Windows Applications

❑ In a Windows Application you can add modules to place global code that can be access by all the Windows Forms.
❑ In addition, the **StartUp** Object can be the *Module* or *Sub Main* when control of the program is initiated.

Code Editor Screen

❑ This screen is where Visual Basic code is written.

Code Editor for Console Application

❑ For a Console Application, the code editor in invoked immediately to allow you to enter code in the Module.
❑ Simply begin entering code in the Module Document.
❑ The Code Editor screen contains two drop-down list boxes, one for the Object you are coding and the other for the Methods & Event-Procedures associated with the object:

Objects

Methods & Event-Procedures

```
Start Page  Module1.vb                                        ◁ ▷ ✕

Module1                        ▼    Main                            ▼

 Module Module1

     Sub Main()

         Console.WriteLine("Hello World")

     End Sub

 End Module
```

Enter Code Here
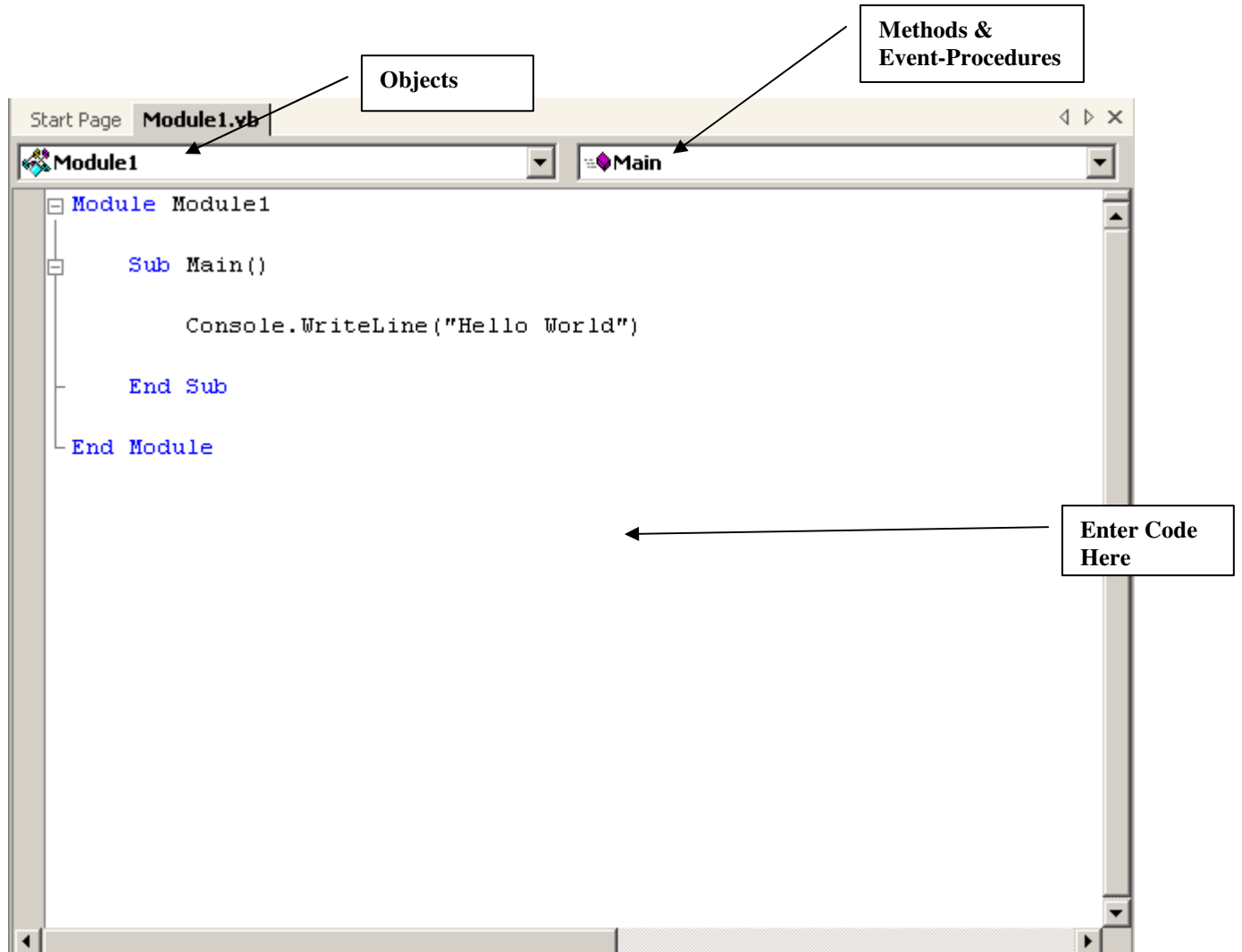
Code Editor for Windows Application

❑ This screen is invoked by *Right-Clicking* on the *Form* object, and selecting *View Code* from the context menu that appears. Another method is by simply *Double-Clicking on the object*:
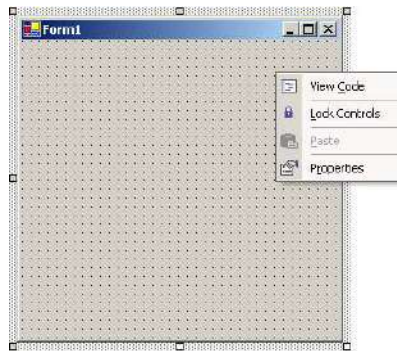
❑ The Code Editor screen contains two drop-down list boxes, one for the Object you are coding and the other for the Methods & Event-Procedures associated with the object:
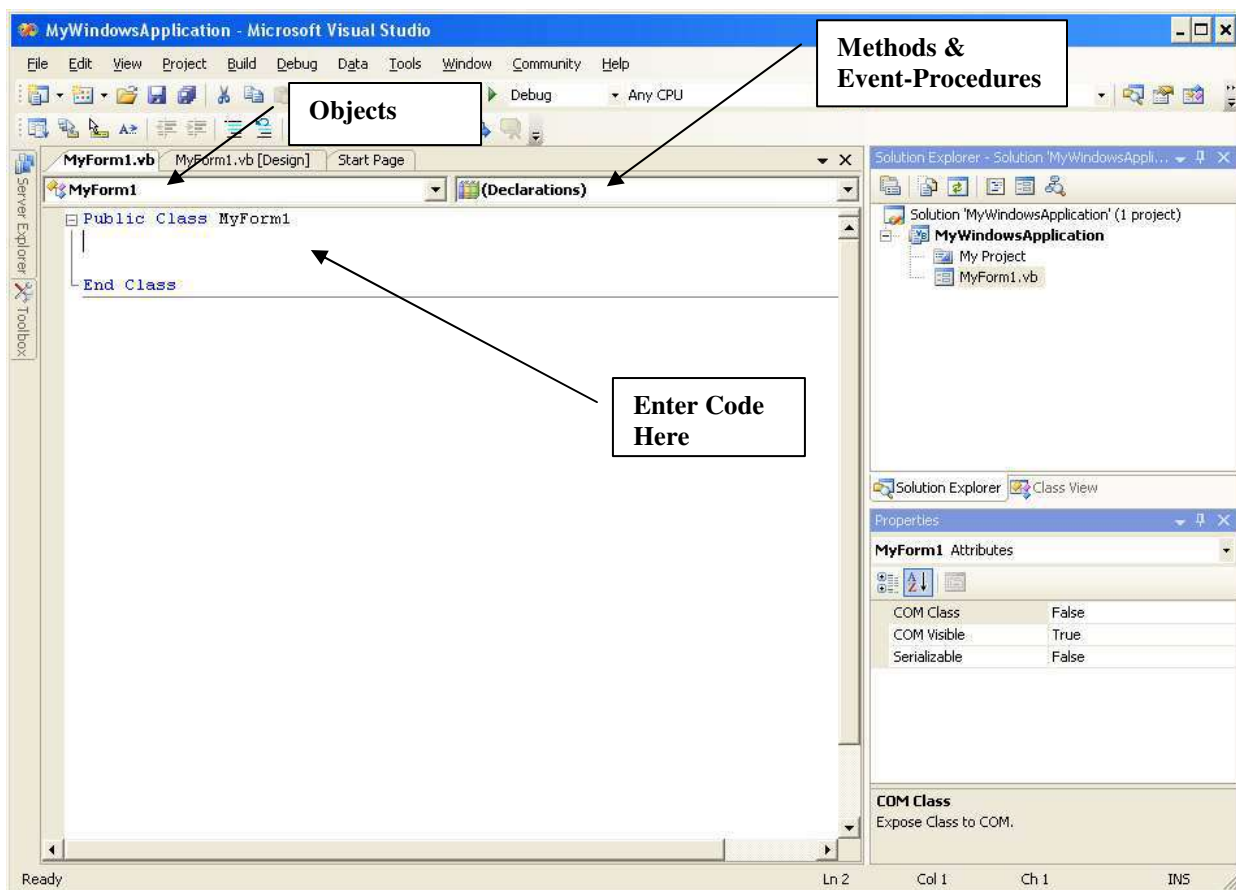
Help

❑ Visual Studio has an extensive help feature.
❑ The Help system contains an entire reference manual, as well as coding examples.
❑ Using this help system you can define the properties to the controls that you will place on the forms.

Step 6: Compile or Build the Program. (Console & Windows Applications)

- In this process you translate from a *High-level Language* to a *Machine Language* the Computer understands.
- This process is know as Building or Compiling the program.
- Note that this process will generate and *executable file* that will be located in the *project folder\bin* directory.
- The steps to build are as follows:
    1) In the Menu Bar select **Build|Build Solution**, this will invoke the *Output Window*
    2) The Output Window displays the results of the Compiler process. The program is fully compiled when all compiler errors are solved.

Output Window - *Compiler Results*
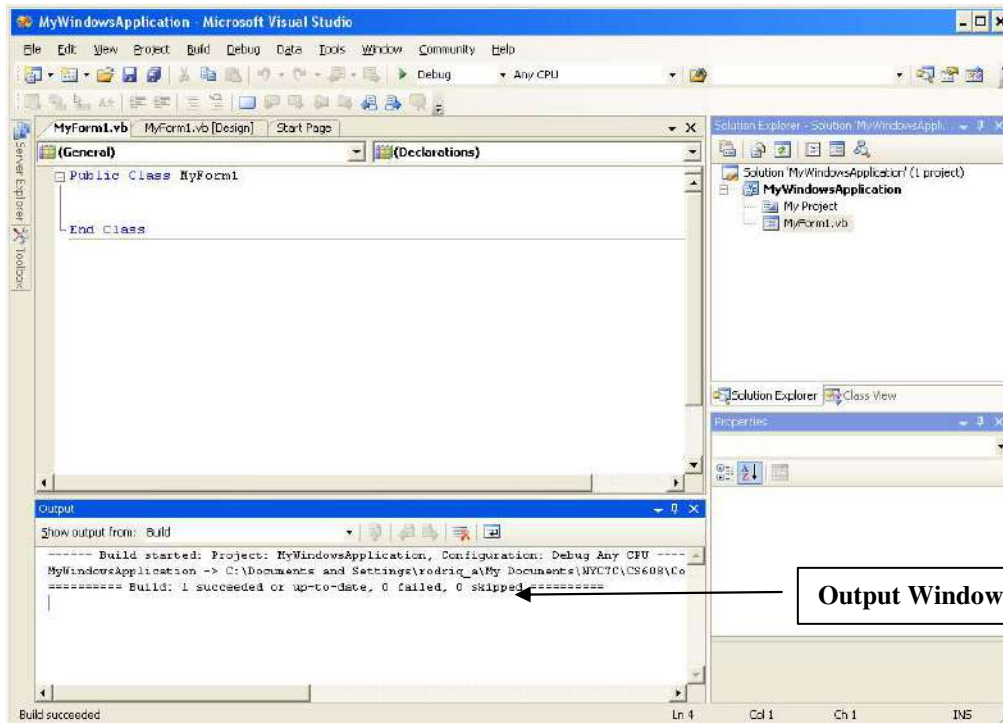
- If the compilation is successful, an *executable file* will generated in the *project folder\bin* directory of your file system.
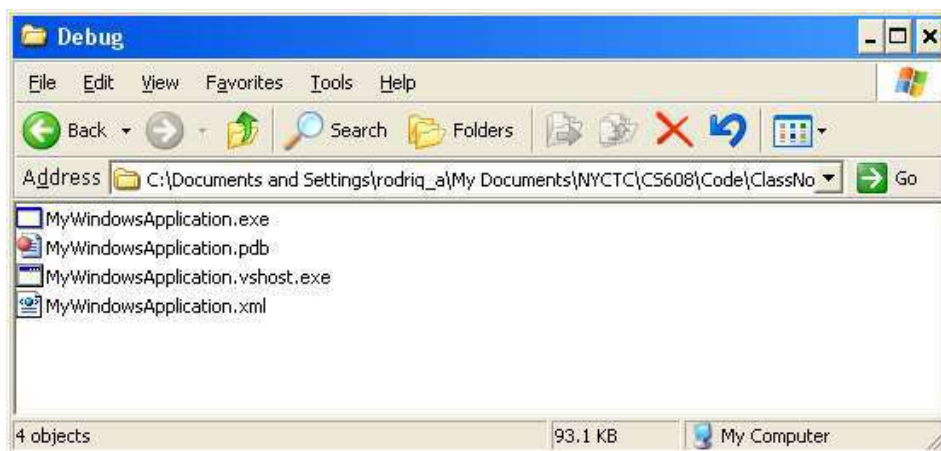
Step 7: Run the Program

- Now you are ready to execute the program.
- Note that you can only execute the program if all Compiler Errors are resolved.
- There are many ways to execute your program primarily using the debugger.

Console Application where Output Window stays displayed (IMPORTANT!):
- Console applications will execute and automatically close the output window. The two methods to executes are:
  1) In the Menu Bar **Debug|Start Without Debugging**.
  2) Or using the keyboard use the Crtl-F5.

Windows Applications & Console Application where Output window automatically closes:
- The three methods to executes are:
  1) In the Menu Bar select **Debug|Start**.
  2) Or using the keyboard use the F5 key.
  3) Or Click on the Start Icon in the Toolbar:

  4) Or navigate to the program folder\bin directory and double-click on the executable file.

2.3 Visual Basics Modes, Error Types & Other Concepts

2.3.1 How Visual Basic Organizes Your Program or Application Files

❑ When you create a VB program a Folder is automatically created in you computer which contains the files of the program.
❑ A Visual Basics application is called a *Solution*. The Solution is composed of one or more *Project*.
❑ The table below lists several important files and their file extensions:

File Extension	Explanation	Example
.sln	**Solution –** File that holds information about all the projects in the application	**VideoManagement.sln**
.vb	**Object File –** Contain definition and code for Forms, Modules, Class Modules etc.	**frmLoginScreen.vb** **MainModule.vb**
.resx	**Resource File –** This is a resource file for the Forms in the project. It contains information about all resources used by the form.	**frmLoginScreen.resx**
.vbproj	**Project File – This file describes the project and lists the files included in the project.**	**VideoManagement.vbproj**

2.3.2 Visual Basics Modes

❑ Visual Basics has three distinct modes:
- **Design Time –** When you design the User Interface (GUI) and writing the code.
- **Run Time –** When you execute and test the program
- **Break Time –** When you have Run time errors and you stop execution

2.3.3 Programming Errors

❑ There are three varieties of programming errors:
- **Syntax Errors –** Errors generated due to the code not following the rules of the language
  o These errors usually involve improper punctuation, spelling or format
  o These problems are easier to solve since the compiler identifies the erroneous statement and you just have to review the rules of the language to make sure you are writing the statement properly

- **Run Time Errors –** Program halts during execution. The program passed the syntax test but fails during execution. .
  o These problems are usually caused by improper arithmetic execution, attempting to access recourses that are not available etc
  o These problems are difficult to solve since they only show up when the program runs.

- **Logical Error –** The program is not doing what is supposed to do. .
  o The algorithm fails to solve the problem the program was written to resolve
  o These problems are even more difficult to solve since you need to re-think and go back to the planning phase and review the Algorithm.

2.3.4 Two Aspects of Visual Basic Object Programming

❑ When programming using Visual Basic Objects & environment there are two aspects to keep in mind :

▪ **Visible Graphical Aspect –** Most Visual Basic Objects & Controls have a Graphical representation. In other word the Object has a visible part. For example in the Toolbox there is a graphical element for a Command Button, Label, Text Box etc.
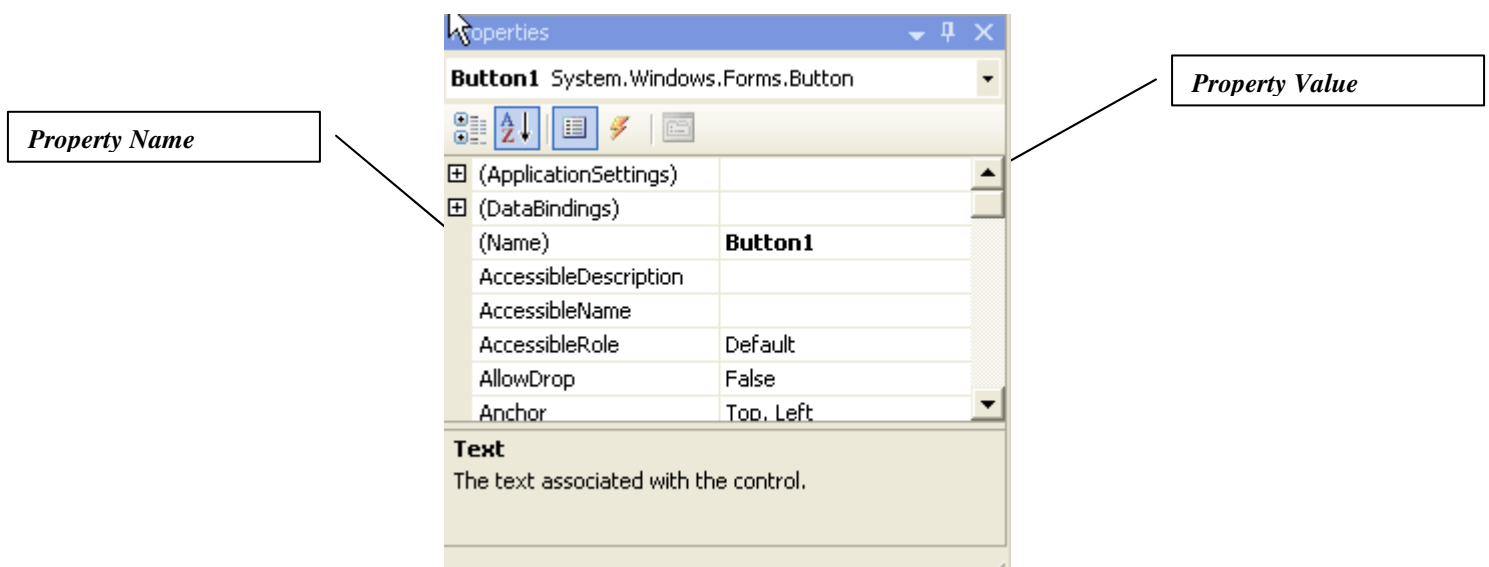

▪ **Invisible Aspect –** The invisible aspect is the part of the Object where you write the code in. This involves double-clikcin on the object and invoking the *Code Editor Screen*.
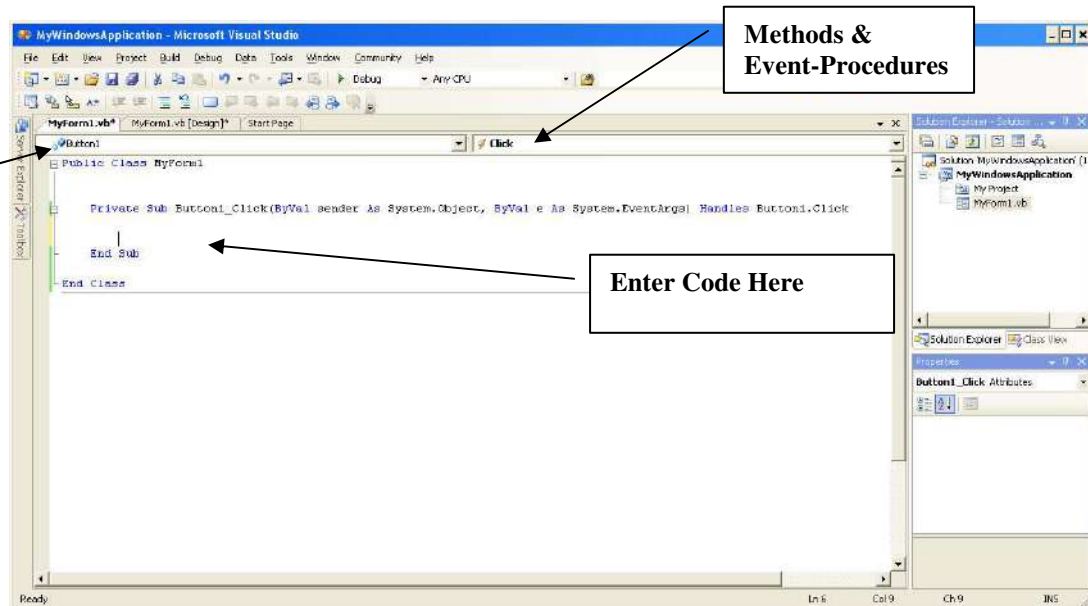
2.3.5 Properties Revisited

❑ Properties are attributes or the data of an Object.
❑ The Property Window is used to set the Properties of Objects

Setting Properties at Design Time
❑ When you set the properties to an Object during Design time, or when you are creating your program, these properties are present when the program executes.

35

❑ These Properties will not change unless otherwise modified by program code during run time.

Setting Properties at Run Time
❑ During run time or when program is executing, you can write code in your program to set the properties of an Object.
❑ For example:

Object.*Property = value*

Example:

btnOK.*Text* = "Exit"

❑ Note that the Properties set and modified during runtime are only valid during Run Time or while the program is executing. After execution the properties set during design time are replaced by those set during design time.

Common Properties
❑ In order to make VB easier to learn, Microsoft gave most of the controls and objects similar properties
❑ For examples, most controls have the following properties:

- **Name Property –** Name used in the Code to identify the Control
- **Text Property –** Text displayed on the Control
- **Enabled Property –** Indicates if Control is enabled (Can be used)
- **Visible Property –** Indicates if the Control can be seen

2.3.6 Windows Applications Control Flow

❏ Before we continue I want to go over the three types of flow of execution. What I mean by this is what object controls the flow of a Window Application.

❏ From previous courses, you were accustomed to controlling the flow of an application via Forms. In other words, the Form was the Startup Object of the application.

❏ But using a Module you can also control the flow of the application via the Sub Main(), which means that that program flow of execution is being controlled by the module.

❏ Now with the introduction of Modules, we can write 2 types of Window Applications:

  1) **Form-Driven Windows Applications:** Application whose flow is controlled by a Form. *Startup Object* is a ***Form***.

    ▪ **Advantage:** Straight forward to program. Simply load and displays the form immediately upon startup.
    ▪ **Disadvantage:** Cannot perform initialization tasks prior that don't involve the Form prior to displaying the form. It is true you can use Form.Load() to perform initialization, but you need to do it from the form, without the form it cannot be done.

  2) **Module-Driven Windows Applications:** Application whose flow is controlled by a Module via Sub Main(). Startup Object is the *Module Sub Main() procedure*. The user interfaces or GUI are initially displayed from the Sub Main procedure.

    ▪ **Advantage:** You can do many task and execute functionalities prior to displaying any forms.
    ▪ **Disadvantage:** Requires more logic to program.

2. 4. Visual Basic Debugging Tool

2.4.1 Understanding the Debugger

❑ The VB debugger is a powerful and useful tool available for you to debug or isolate errors.
❑ Some errors are sometimes difficult to isolate when the program is executing. The debugger can assist you in isolating some of these errors.
❑ The debugger is not intended to help you identify syntax errors, that is the job of the Compiler or Build engine.

❑ When you execute a program, you can execute it in **Debug Mode**.
❑ Once you execute a program in Debug Mode, you can use the tools and features of the debugger to isolate problems.

2.4.2 Setting Breakpoints

❑ The debugger contains many powerful features that I will not cover here, but one feature that we will be using quite often as we test our program is to set **breakpoints**.
❑ Breakpoints are a flag or insertion point that tells the debugger to temporarily suspend the execution of the program at that point.
❑ Breakpoints work as follows:

1. At the desired point, in the code window, *right-click* on a code statement and a menu will appear.
2. Select Insert *Brakepoint*
3. The code statement will be highlighted indicating that the breakpoint has been set
4. Run the program, the execution will pause at the breakpoint
5. Once execution is suspended, you can view information of the code at that point to determine if the code is working correctly up to that point. View the various variables and program code to determine the status of the program at that point.
6.   Once you are finished your observation and reach a conclusion, then you can continue with the flow of the program. In the menu bar click **Debug|Continue**

2. 4. Putting it All Together

2.4.1 OOP Programming using Visual Basics In a Nutshell

❑ OK lets put it all together

Part I – Planning & Design of GUI, Properties & Algorithm (Think)

- Analyze the requirements
- Use UML to design class diagrams, work flow etc.

Part II – Implement the Program

1. **Enter Design Mode –** Use the Visual Basics.NET IDE screen to create the program.
2. **Set the Project Properties or Startup Object**
3. **Design the GUI or Visible Aspect of the program**
   - Using the IDE components, create the GUI by placing controls onto the Form
   - Set the properties to the Controls

4. **Invoke the Invisible Aspect of the Objects or Control to Write the Code**
   - Invoke the Code *Editor Screen* to write code to manipulate the control objects on the Form
   - This involves coding in the Methods & Event-Procedures in order to implement the Algorithm

5. **Compile the Program –** Resolve all *Compiler Syntax Errors*

6. **Enter Run Time & Break Time –** Execute the program
   - Execute the program and resolve all *Run Time Errors*
   - Resolve all *Logical Errors*

2.5 Sample Programs

### 2.5.1 Sample Program 1: Console Application – Login Request Application

❑ The following example illustrates a Console Application.   Recall that a Console Application is a text based application.
❑ Problem Statement:

> ▪ Create a Console Application that prompts the user for the username & password.
> ▪ The values entered are displayed via a message box

❑

Step 1: Open the Visual Studio IDE and invoke the Start Page & Select New Project

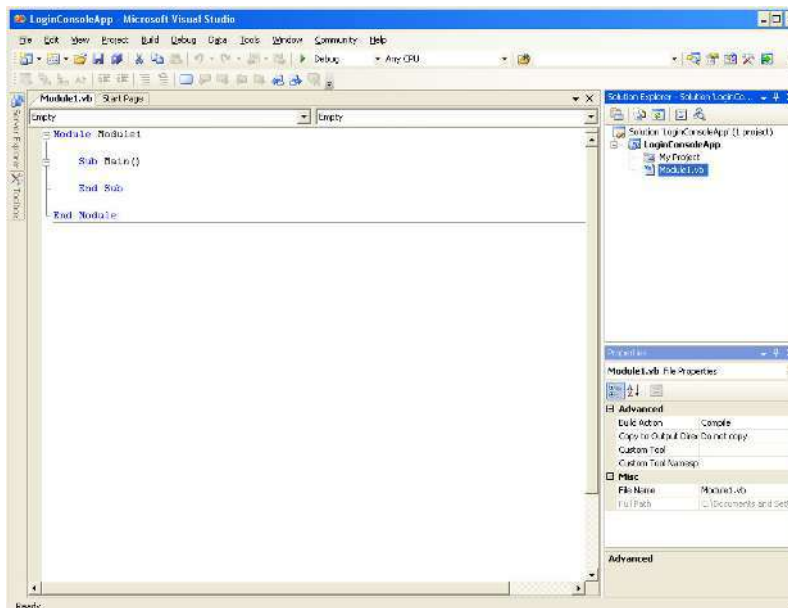## Step 2: In the New Project Dialog select Console Application

Step a – Project type
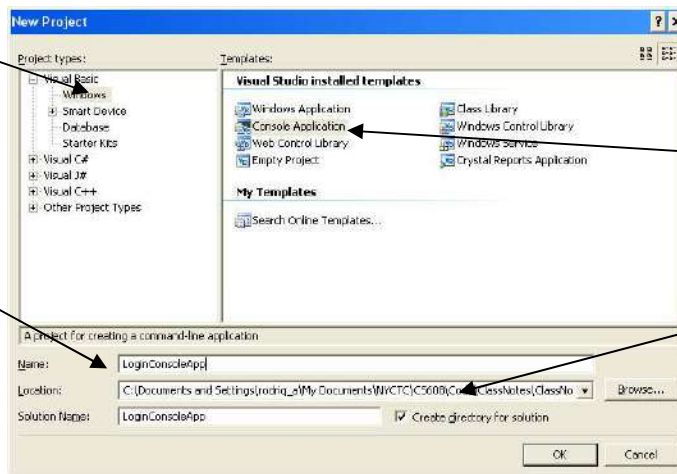
Step b – Console Application
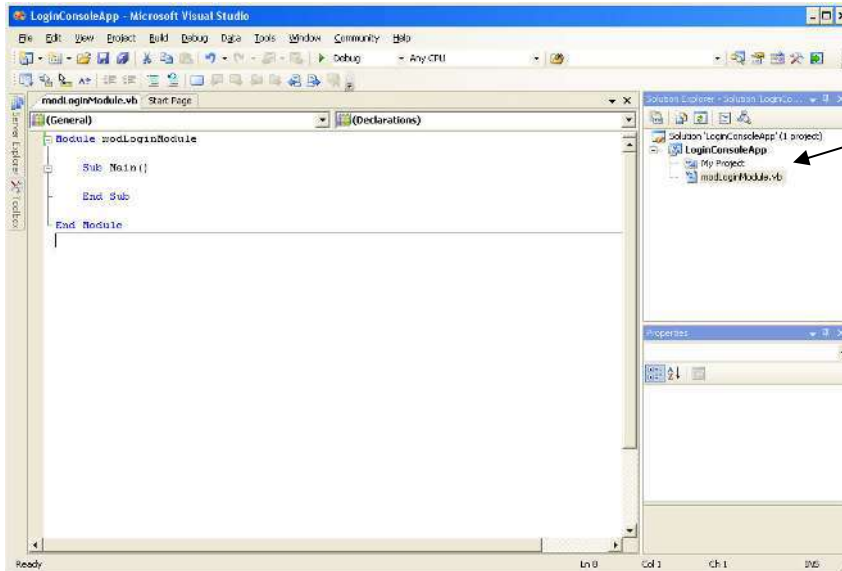
Step c – Project Name

Step d – Project Path

Step 3: IDE is invoked

## Step 4: Rename Module & Project Files

❑ Now we want to rename some of our objects in the Solution Explorer Window & Document Window
❑ The Objects we wish to rename are the *Module File*:

- Take the following steps:
  - o *Step a - Module Object***:** In the Document screen, highlight the name of the Module declaration "Module1". Using the keyboard type in the name of the Module Class
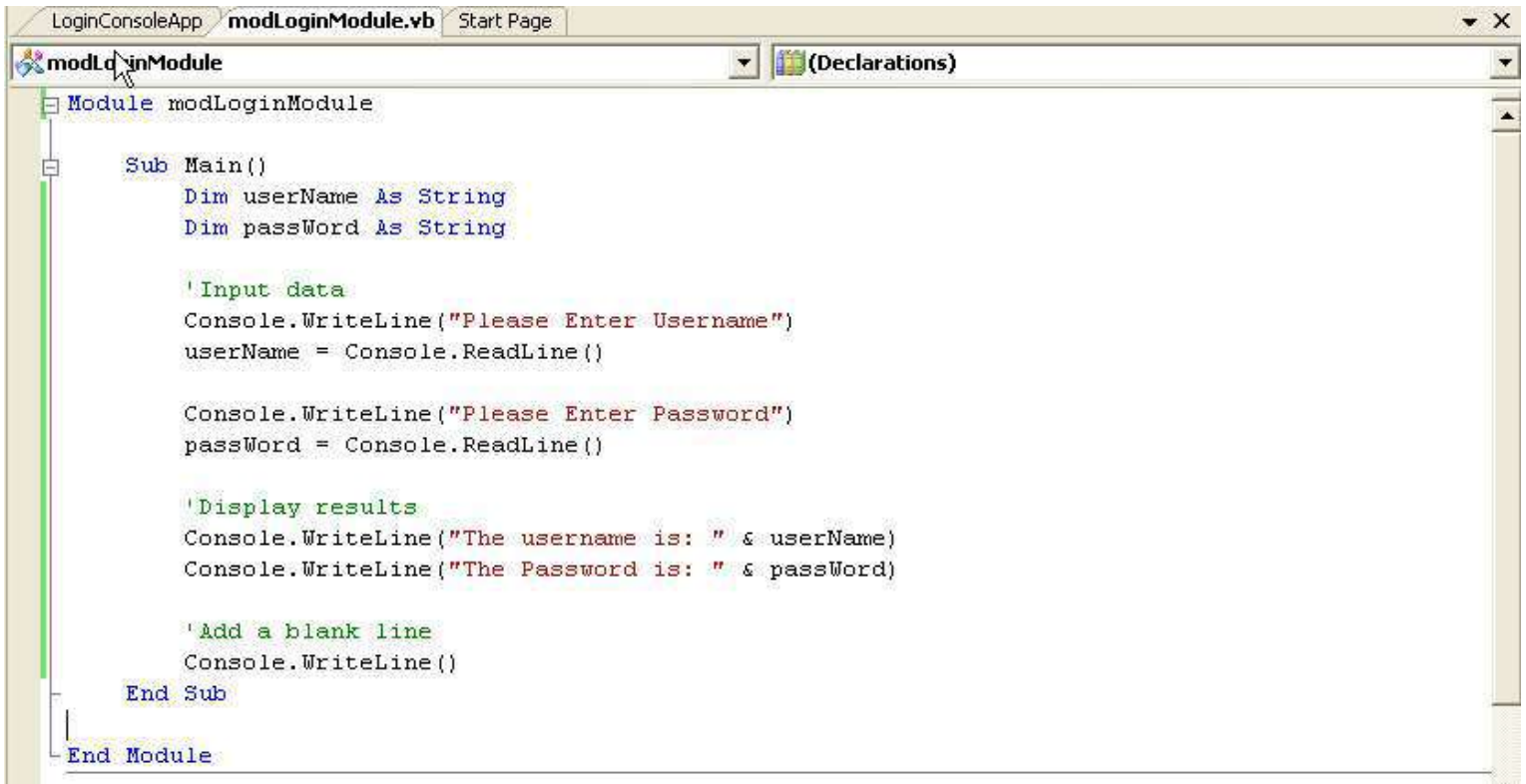

Step a – Rename
Module File

Step 5: Set Project Properties Startup Object

❑ The *Project Properties Window* is invoked as follows:

- In the **Solution Explorer**, simply *Right-Click* on the *project name LoginScreenProject* and select Properties from the context menu to invoke the *Project Property Page*

- Since this is a Console Application, the startup Object will need to be set the Module *modMain*.

Startup Object is
automatically the module
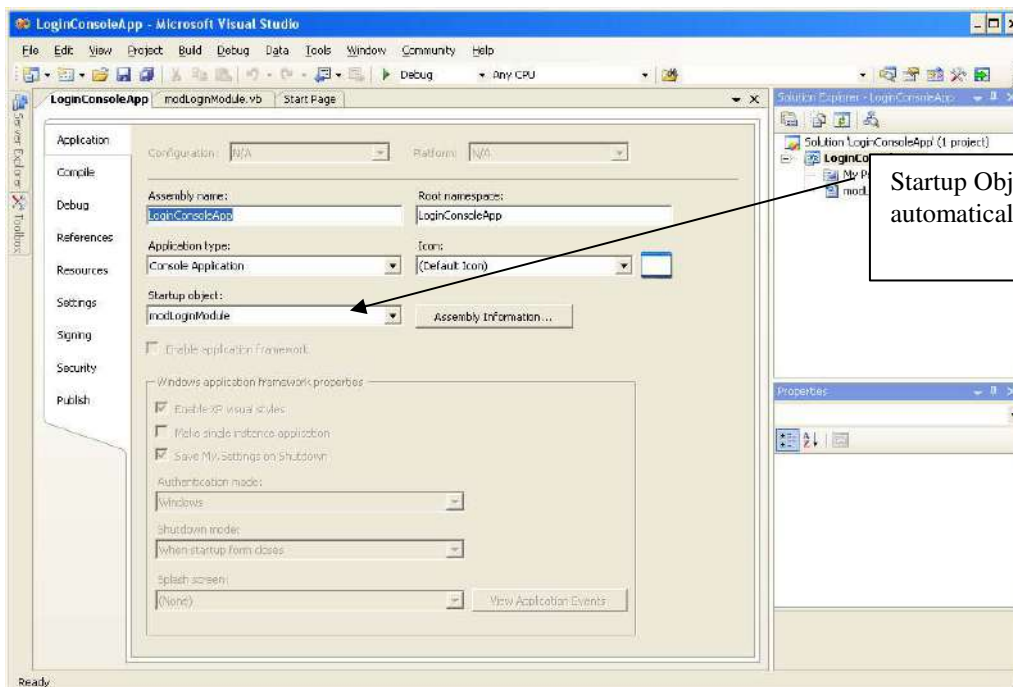
Step 6: Add Code in Module Object

❑ In a Module the starting point is Sub Main. Enter desire code in the Sub Main() method.

▪ In this example, we will add a Console Application Output statement to input data and display text onto the screen:

▪ Enter the following code:

```
LoginConsoleApp  modLoginModule.vb  Start Page

modLoginModule                                    (Declarations)

Module modLoginModule

    Sub Main()
        Dim userName As String
        Dim passWord As String

        'Input data
        Console.WriteLine("Please Enter Username")
        userName = Console.ReadLine()

        Console.WriteLine("Please Enter Password")
        passWord = Console.ReadLine()

        'Display results
        Console.WriteLine("The username is: " & userName)
        Console.WriteLine("The Password is: " & passWord)

        'Add a blank line
        Console.WriteLine()
    End Sub

End Module
```
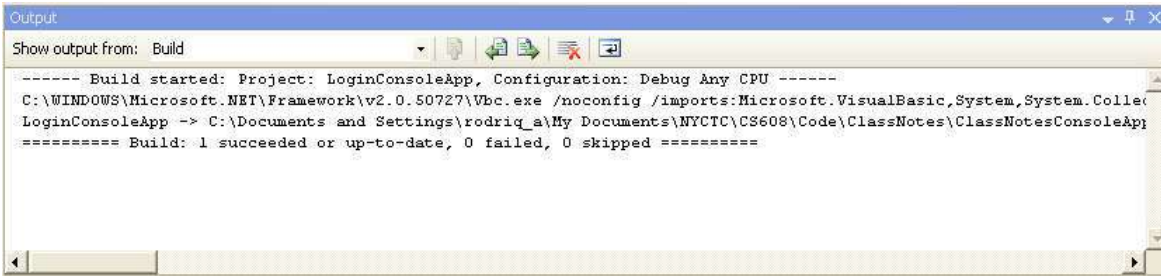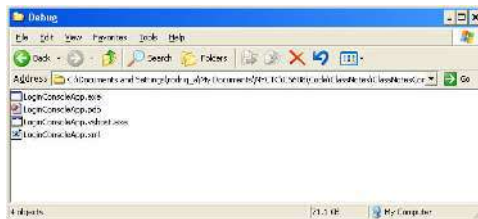
Step 7: Build or Compile the Program

❑ Now Compile or Build the program to generate the executable .EXE file.

  ▪ The steps to build are as follows:
    1) In the Menu Bar select **Build|Build Solution**, this will invoke the *Output Window*
    2) The Output Window displays the results of the Compiler process. It shows if the results of the compiler were successful or failed. For a fully compiled program, all compiler errors must equal to 0

❑ If you browse to the project folder and you navigate to the *bin* folder you will see the executable file:

Step 8: Execute or Run the Program

❑ To run or execute the program do the following:

Console Application where Output Window stays displayed:
  ▪ The three methods to executes are:
    1) In the Menu Bar **Debug|Start Without Debugging**.
    2) Or using the keyboard use the Crtl-F5.
    3) Or navigate to the project folder\bin directory and double-click on the executable file *LoginScreenProject.exe*.

❑ Note that the output windows stays displayed until the user presses any key

2.5.2 Sample Program 2: Form-Driven Windows Application – Login Request Application

❑ The following example illustrates a Windows Application. Recall that a Windows Application is a Form based application.
❑ **IN A *FORM-DRIVEN APPLICATION*, CONTROL OF THE PROGRAM EXECUTION IS DONE BY THE STARTUP FORM! WHEN YOU CLOSE THAT FORM THE PROGRAM ENDS!**
❑ Problem Statement:

> ▪ Create a Windows Application that displays a Login Form on the screen requesting a Username & Password
> ▪ The values entered are displayed via a message box when the user clicks an OK. When the user clicks the Cancel button, the text boxes are cleared.

❑ The steps and code are as follows:

Step 1: Open the Visual Studio IDE and invoke the Start Page & Select New Project

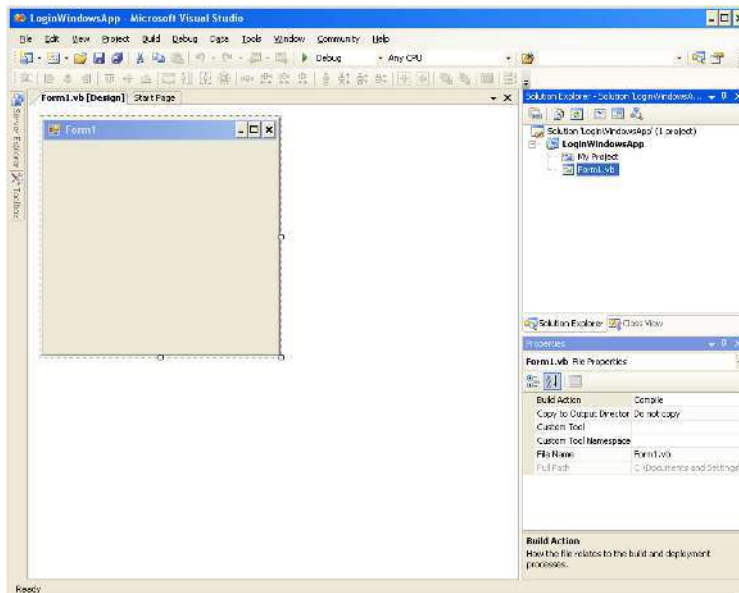## Step 2: In the New Project Dialog select Windows Application

Step a – Project type

Step b – Window Application

Step c – Project Name

Step d – Project Path

Step 3: IDE is invoked

Step 4: Rename Form & Project Files

❑ Now we want to rename some of our objects in the Solution Explorer Window
❑ The Objects we wish to rename is the *Form File*:

- Take the following steps:
  - o **Step a - *Form File*:** Select the *Form1.vb* item in *the Solution Explorer Window*, and in the *Property Window* select the *File Name* Property. Enter the desire name.

Step a – Rename Form File
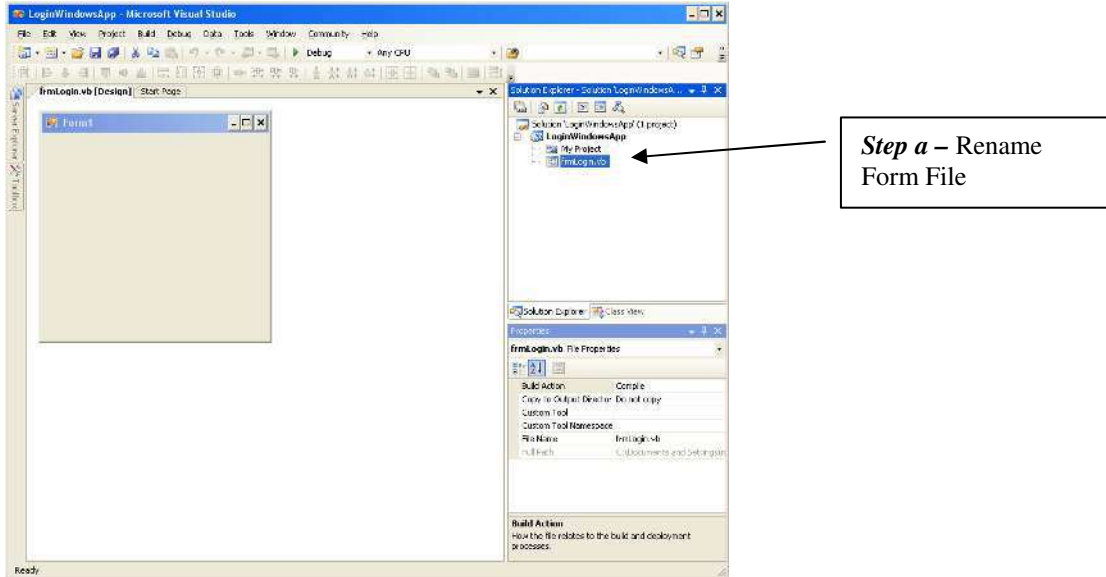
Step 5: Set Project Properties Startup Object

❑ For Windows Applications, the startup object will automatically be the FORM.
❑ If we open the *Project Properties Window* we see the following:

- In the **Solution Explorer**, simply *Right-Click* on the *project name* and select Properties from the context menu to invoke the *Project Property Page*

- The startup form is frmLogin.

Select *frmLogin* as Startup Object
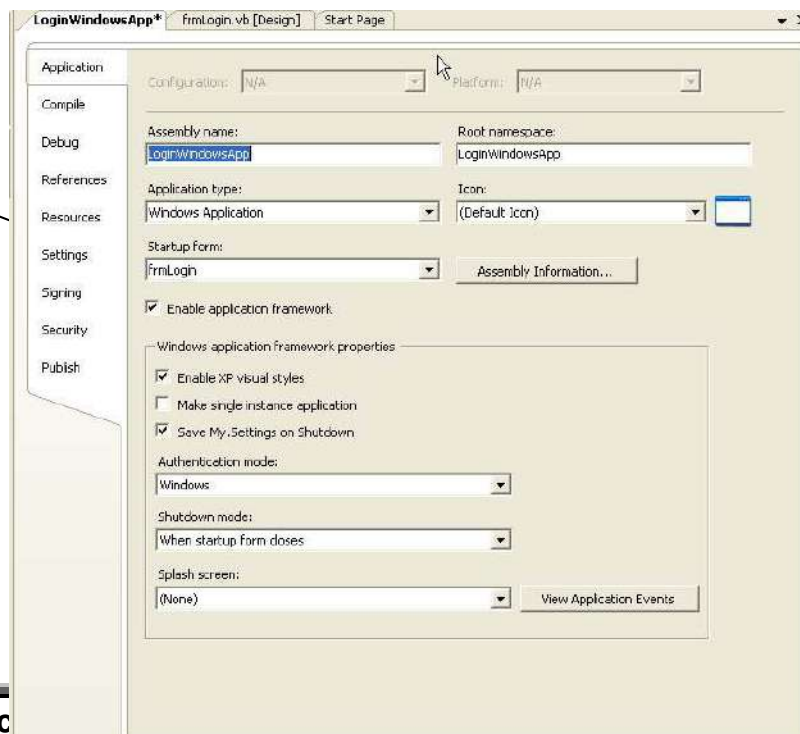
Step 6: Design the User Interface

- This is where the real work takes place. You will now apply the process of thinking, planning , designing & implementing your program.
- You will use the IDE *Form Designer* & *Toolbox* to create your User Interface followed by the *Code Editor* to enter you code.

Three Step Process
- Creating an Application involves a Two Phase process, I) **Planning** II) **Programming**.
- Each Part is broken into three steps:

   I.   **Planning & Design Phase:**

       2. *Design* the User-Interface (GUI) – You draw a sketch of what the Front-End or GUI will look like and the Control Objects required.
- In this case we wish to create a login screen with labels describing what it is and text box controls to accept the input from the user. In addition push buttons to proceed or cancel the operation. For example we could draw the following diagram:
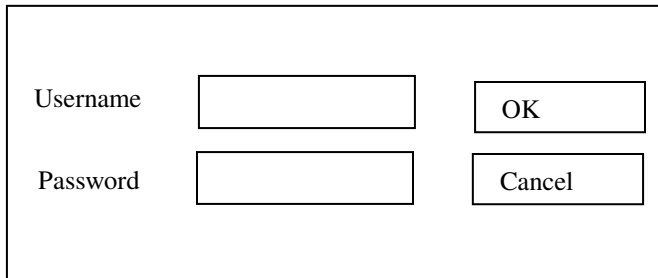
       3. *Plan* the Properties or attributes to the objects in the GUI – You need to write down the properties to the Control Objects that will make up the GUI. Usually you create a table listing the control and its properties. From the diagram above we see that we need one Form two labels, two textboxes and two push buttons.

Object	Property	Value
Form 1	Name Text	**frmLogin** **Login Form**
Label1	Name Text	**lblUsername** **Username**
Label2	Name Text	**lblPassword** **Password**
Textbox1	Name	**txtUsername**
Textbox1	Name	**txtPassword**
Command button 1	Name Text	**btnOK** **OK**
Command button 2	Name Text	**btnCancel** **Cancel**

4. *Plan the code* – In this step, you plan (THINK) the code or steps required for the program to run. You will write down the action required to solve the problem. You will use **programming tools** like *UML* to design your classes and work flow, **pseudo-code** language similar to a short-hand English expressions and **flow charts** to plan the necessary <u>logic</u> to solve the problem. This is really the tough part of programming since it requires thinking logically.

❑ One method I can recommend is as follows:

- Read the requirements several times until you understand what is required of the program.
- Now put yourself in the role of the computer program and what is require of you to do, and then switch to the role of the user is standing in front of the computer to use it and what is required of the user to do.
- As yourself questions as to what is required for each of these characters during their interaction.
- Now you write down the steps of the interaction which takes place between the program and the user. Now extract the performed by the computer program, put these steps in their proper order and you have an algorithm and an idea of what needs to be done
- For example, in this login program here are the interactions between the computer program and the user:

    What is the First thing the User does?
    a) Sits in front of the computer

    What should the Computer do?
    b) Display the Login Screen

    What should the User do?
    d) Enter username
    e) Enter password
    f) Click Ok button or Cancel button

    What should the Computer do?

    If you user clicks the OK button:
    f) Extract the username
    g) Extract the password
    h) Display a message stating that the user entered a username & password

    If you user clicks the Cancel button:
    c) Display a message stating that the user cancelled the operation
    d) Clear the text boxes

- Now write down all the steps performed by the Computer Program to derive your algorithm:

    1. Display the Login Screen
    2. If you user clicks the OK button:
        a) Extract the username
        b) Extract the password
        c) Display a message stating that the user entered a username & password

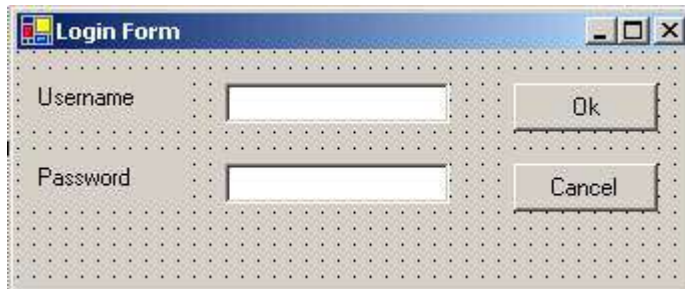   3. If you user clicks the Cancel button:
        a) Clear the text boxes

❑   Add controls to the Form and create the program code.
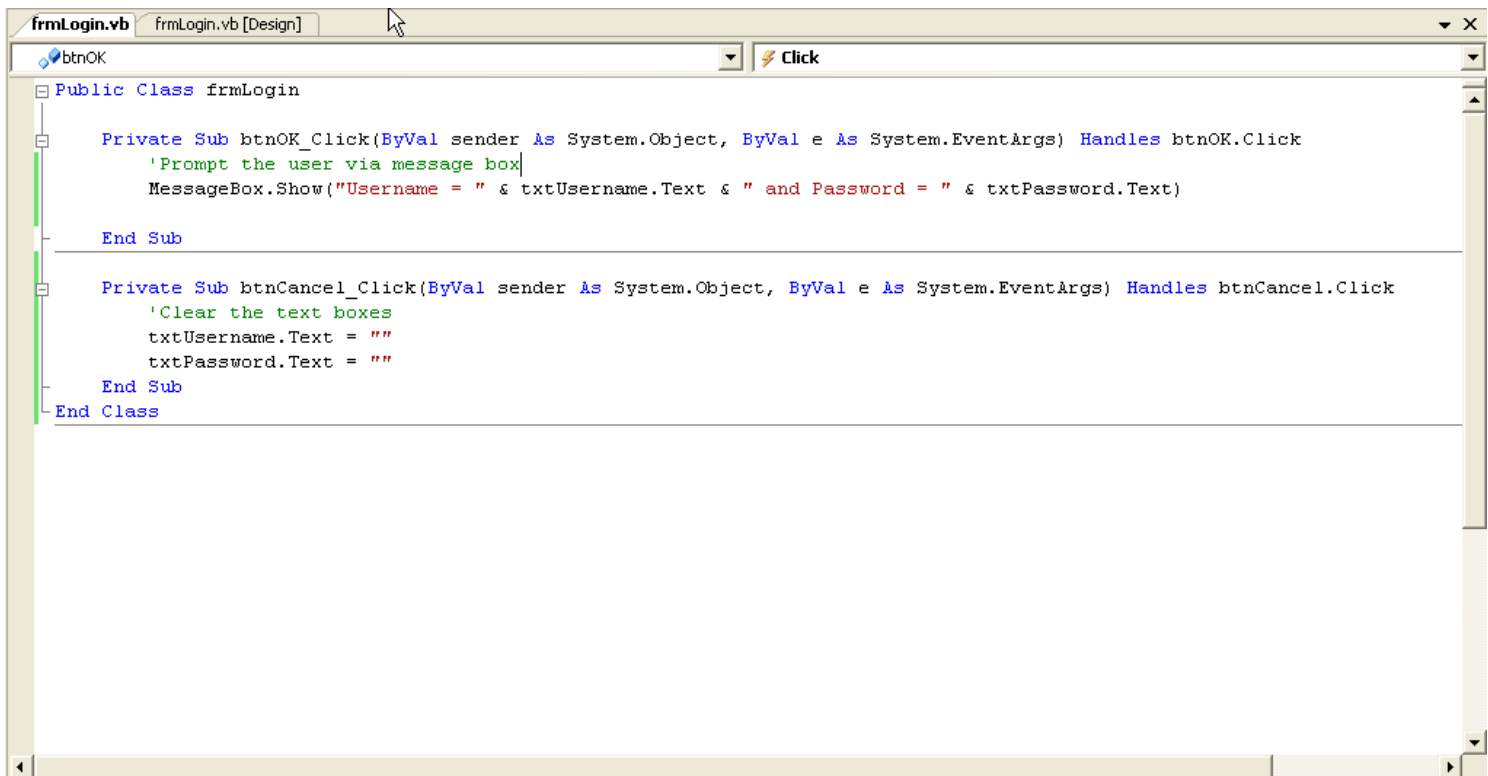
II.   Programming:

5.   *Create* the User-Interface (GUI) as follows:

1.   Add controls to your form
2.   Set their properties based on the property table you derived during design. The results should be as
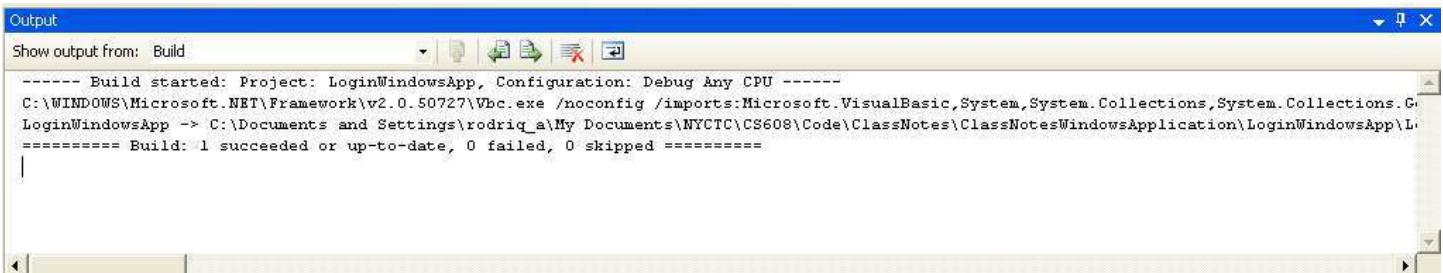

6.   *Write the code* – Use the syntax or programming code to create the program base on the algorithm previously derived:

■   The algorithm derived was as follows:

4.   Display the Login Screen
5.   If you user clicks the OK button:
   a)   Extract the username
   b)   Extract the password
   c)   Display a message stating that the user entered a username & password

6.   If you user clicks the Cancel button:
   a)   Display a message stating that the user cancelled the operation
   b)   Clear the text boxes

■   Since this is a graphical application and the Startup Form is the login Form it will display automatically, therefore we don't need to manually specify step 1 or display the login form. It will be done automatically when the program starts. So now the algorithm looks like this:

1.   If you user clicks the OK button:
   a)   Extract the username
   b)   Extract the password
   c)   Display a message stating that the user entered a username & password

2.   If you user clicks the Cancel button:
   a)   Clear the text boxes

■   From the algorithm it shows that you need to program the click event of the OK button and the click event of the cancel button and enter the code for each step:
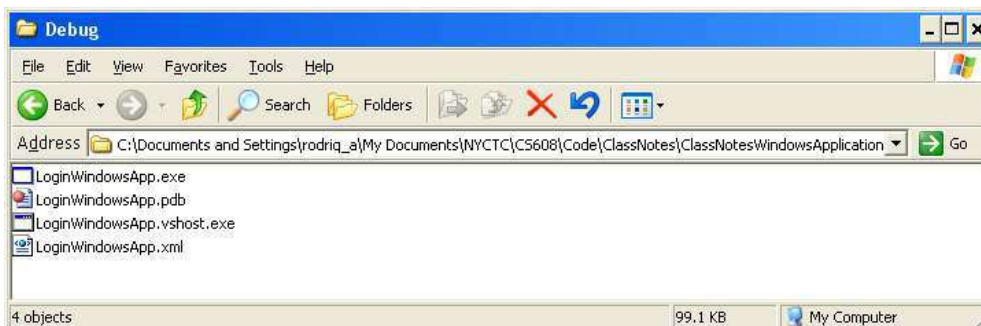
48

```
frmLogin.vb    frmLogin.vb [Design]                                                        ▼ ✕

  btnOK                                              ▼  ⚡ Click                               ▼

□ Public Class frmLogin

        Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.Click
             'Prompt the user via message box
             MessageBox.Show("Username = " & txtUsername.Text & " and Password = " & txtPassword.Text)

        End Sub

        Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCancel.Click
             'Clear the text boxes
             txtUsername.Text = ""
             txtPassword.Text = ""
        End Sub
  End Class
```

Step 8: Build or Compile the Program

❑ Now Compile or Build the program to generate the executable .EXE file.

- The steps to build are as follows:
  1) In the Menu Bar select **Build|Build Solution**, this will invoke the *Output Window*
  2) The Output Window displays the results of the Compiler process. It shows if the results of the compiler were successful or failed. For a fully compiled program, all compiler errors must equal to 0

```
Output                                                                                   ▾ 垔 ✕
Show output from: Build             ▾  |  📄  | 📲 📲 | ➡ | 🔁
------ Build started: Project: LoginWindowsApp, Configuration: Debug Any CPU ------
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Vbc.exe /noconfig /imports:Microsoft.VisualBasic,System,System.Collections,System.Collections.G
LoginWindowsApp -> C:\Documents and Settings\rodriq_a\My Documents\NYCTC\CS608\Code\ClassNotes\ClassNotesWindowsApplication\LoginWindowsApp\L
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

❑ If you browse to the project folder and you navigate to the *bin/Debug* folder you will see the executable file:

```
📁 Debug                                                                          _ □ ✕
File   Edit   View   Favorites   Tools   Help                                              ⨁

⬅ Back ▾  ➡  ▾  🔼  |  🔍 Search  📂 Folders  | 📋 📋 ✕ 🔄 | ▦ ▾

Address 📁 C:\Documents and Settings\rodriq_a\My Documents\NYCTC\CS608\Code\ClassNotes\ClassNotesWindowsApplication ▾  ➡ Go

  📄 LoginWindowsApp.exe
  📄 LoginWindowsApp.pdb
  📄 LoginWindowsApp.vshost.exe
  📄 LoginWindowsApp.xml

4 objects                                                    99.1 KB      💻 My Computer      //
```

❑ To run or execute the program do the following:

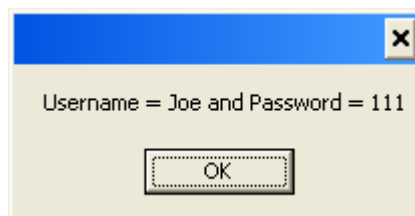- ▪ The three methods to executes are:
  1) In the Menu Bar select **Debug|Start**.
  2) Or using the keyboard use the F5 key.
  3) Or Click on the Start Icon in the Toolbar: ▶
  4) Or Navigate to the program folder\bin directory and double-click on the executable file *LoginFormWinApp.exe*

HOW IT WORKS
- ❑ The program execution is controlled by the Login Form
- ❑ When the login Form starts you can access/use the controls, to execute their event handlers and do things for you.
- ❑ In this case when you click OK, you will get display the message box of the user/pass you entered. When you click cancel, the text boxes are cleared.
- ❑ Once the Form is closed the program ends.

2.5.3 Sample Program 3: Module-Driven Windows Application– Login Request Application Version 1 (Processing Code Inside Form)

- ❑ The following example illustrates a Windows Application that is Module-Driven or uses a Standard Module Object to control the operation of the program.
- ❑ The program will be controlled from the *Sub Main()* procedure of the Module.
- ❑ **IN A *MODULE-DRIVEN APPLICATION*, CONTROL OF THE PROGRAM EXECUTION IS DONE BY THE SUB MAIN(), WHEN SUB MAIN() ENDS, THE PROGRAM ENDS AS WELL!**
- ❑ We will use the same problem Statement as the previous example 2 to illustrate this concept.
- ❑ We will provide two different versions of this application. In version 1, we will display the message box within the OK Click Event as we did in the previous Sample Program 2.
- ❑ The program statement is as follows:

> - Create a Windows Application that displays a Login Form on the screen requesting a Username & Password
> - The values entered are displayed via a message box when the user clicks an OK. When the user clicks the Cancel button, the text boxes are cleared.

- ❑ The steps and code are as follows:

Step 1: Open the Visual Studio IDE and invoke the Start Page & Select New Project

Step 2: In the New Project Dialog select Windows Application

Step a – Project type

Step b – Window Application

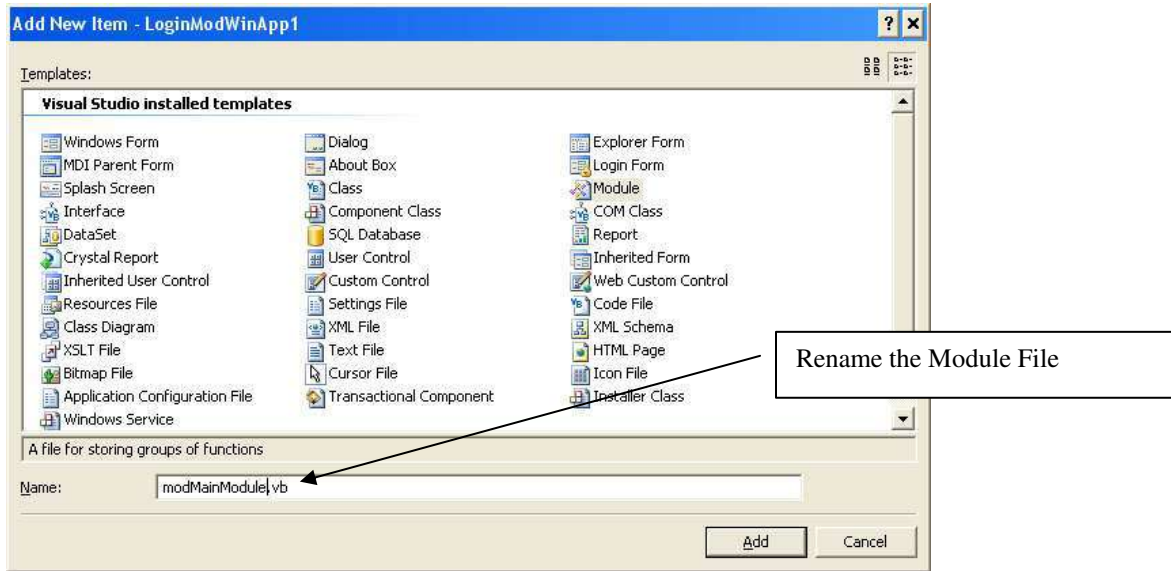*Step d –* Project Path

Step c – Project Name

Step 3: IDE is invoked

Step 4a: Add a
Module Object to the Project

- ❑ Since this is a *Module-Driven Windows* *Application* a Standard Module Object is required in your application.
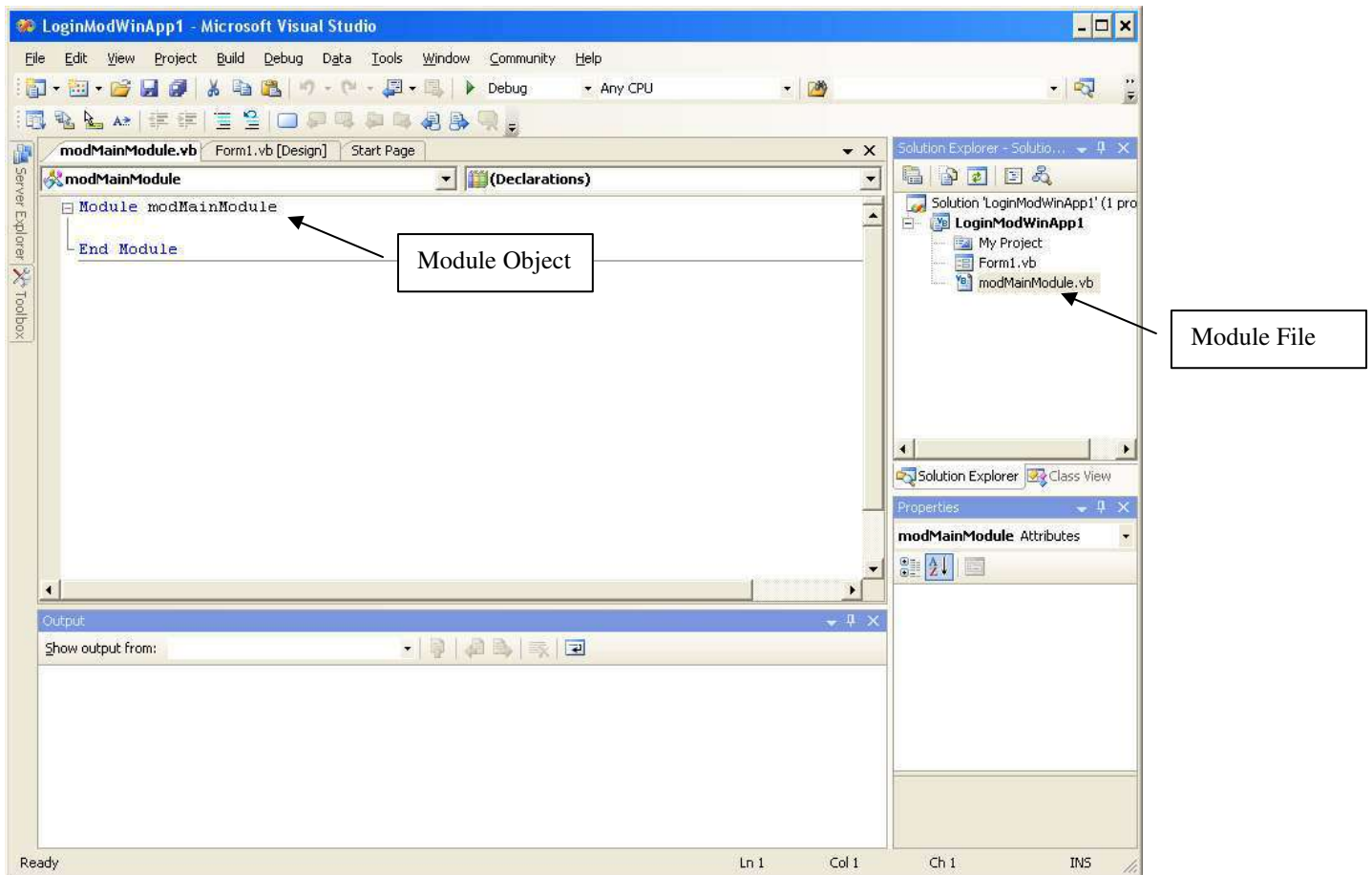- ❑ Add a Standard Module to the project. This module will contain code that will be public to the entire project.

❑ The steps are as follows:
1. In the Menu bar click **Project|Add Module…** to invoke the Add New Item screen
2. In the *Template* section select *Module*, click Open.

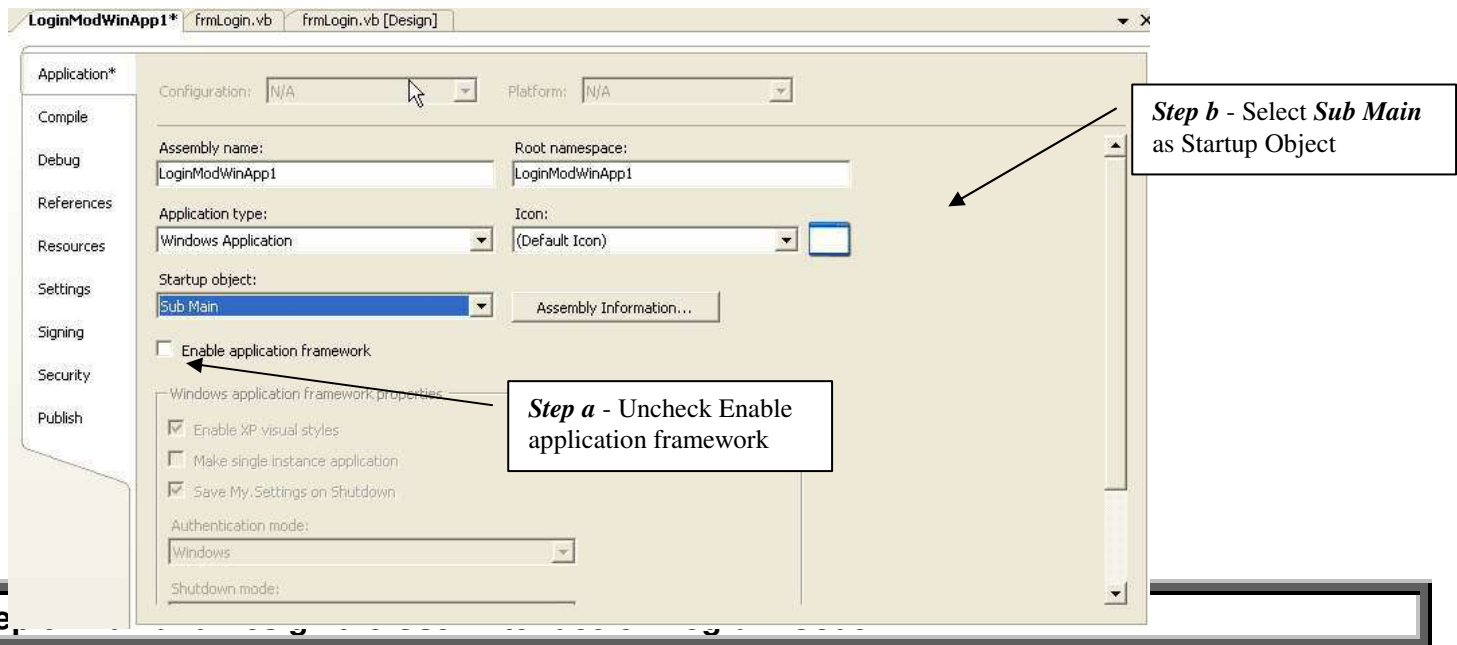3. The module is now part of the project in the IDE.

Step 4b: Rename Form

❑ Now we want to rename some of our objects in the Solution Explorer Window & Document Window
❑ The Objects we wish to rename are the *Form File*:

- Take the following steps:
  - o **Step a - Form File:** Select the *Form1.vb* item in *the Solution Explorer Window*, and in the *Property Window* select the *File Name* Property. Enter the desire name.

Step a – Rename Form File
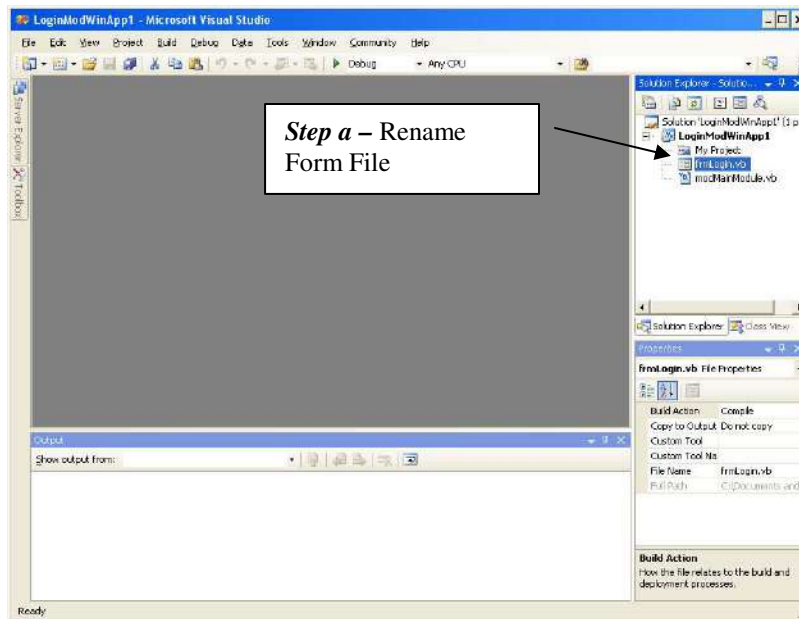
Step 5: Set Project Properties Startup Object

❑ Since this Windows Application is driven by the Standard Module **Sub Main()**, the startup Object will need to be the **Sub Main()** Procedure.

❑ The *Project Properties Window* is invoked as follows:
  - In the **Solution Explorer**, simply *Right-Click* on the **project name *LoginModWinApp1*** and select Properties
  - Uncheck the "***Enable application framework***" check box.
  - In the Startup Object section, click the drop-down arrow and select **Sub Main()**.

Step b - Select **Sub Main** as Startup Object

Step a - Uncheck Enable application framework

Ste

❑ As the previous example, here is where all the thinking, planning , designing & implementing your program.

Three Step Process

❑ Creating an Application involves a Two Phase process, I) **Planning** II) **Programming**.

53

❑ Each Part is broken into three steps:

I. **Planning**
II. **Design Phase:**

1. *Design* the User-Interface (GUI) – You draw a sketch of what the Front-End or GUI will look like and the Control Objects required.
   - We are use the same design as the previous example:

Username	[ ]	OK
Password	[ ]	Cancel

2. *Plan* the Properties or attributes to the objects in the GUI – We will use the same controls and properties as the previous example:

Object	Property	Value
Form 1	Name Text	**frmLogin** **Login Form**
Label1	Name Text	**lblUsername** **Username**
Label2	Name Text	**lblPassword** **Password**
Textbox1	Name	**txtUsername**
Textbox1	Name	**txtPassword**
Command button 1	Name Text	**btnOK** **OK**
Command button 2	Name Text	**btnCancel** **Cancel**

3. *Plan the code* – In this step, you plan (THINK) the code or steps required for the program to run. The thinking and analysis are the same as previous example.

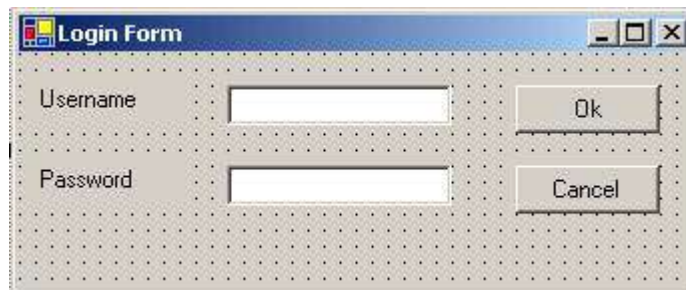❑ Derive the algorithm the same as previous example:

    1. Display the Login Screen
    2. If you user clicks the OK button:
        a) Extract the username
        b) Extract the password
        c) Display a message stating that the user entered a username & password

    3. If you user clicks the Cancel button:
        a) Display a message stating that the user cancelled the operation
        b) Clear the text boxes

Step 7: Implement the User Interface Form & Add Program Code

❑ As the previous example, we add controls to the Form and create the program code.

III. Programming:

1. *Create* the User-Interface (GUI) following the same guidelines as the previous example:

2. *Derive the Algorithm* – Create the algorithm using Pseudo-code:

▪ The algorithm derived was as follows:

    1. Display the Login Screen
    2. If you user clicks the OK button:
        a) Extract the username
        b) Extract the password
        c) Display a message stating that the user entered a username & password

    3. If you user clicks the Cancel button:
        a) Clear the text boxes

▪ Similar to the previous example, this is a graphical application but since it will be driven by the *Module* the *Startup* Object is Sub Main and NOT the login Form, we need to manually display the Form via code from the Sub Main() procedure. The login form will NOT automatically display, the program needs to do it via code. With this in mind then the algorithm stays as is, where all steps are being done by the program code:

3. *Write the code* – Use the syntax or programming code to create the program base on the algorithm previously derived:

- So to begin with we need to control the program from Sub Main. This means we need to create the Sub Main procedure in the Module as follows:
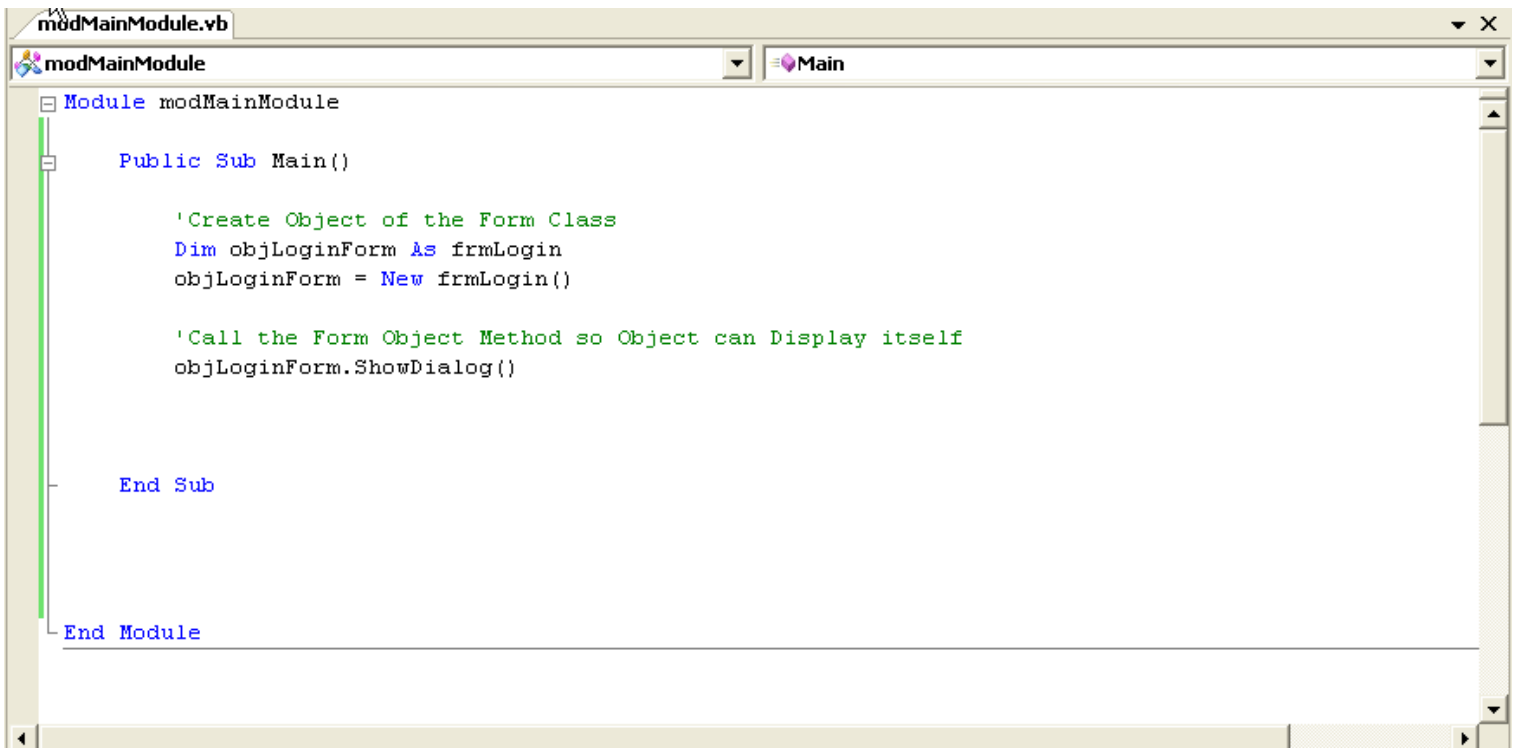
```
modMainModule.vb                                              ▾ ×
modMainModule                    ▾   (Declarations)           ▾
  Module modMainModule

      Public Sub Main()



          End Sub


  End Module
```

- Now we need to implement the algorithm each step at a time. The program starts by implementing step 1 of the algorithm:

  1. Display the Login Screen

- Note that when the startup object is a form as in the previous example, the IDE created an object of the Form Class internally and automatically displayed the form object. But in this case since you need to display the form manually, you need to create an object of the Form Class first and then display the form. The code is as follows:

```
modMainModule.vb                                              ▾ ×
modMainModule                    ▾   ◆Main                    ▾
  Module modMainModule

      Public Sub Main()

          'Create Object of the Form Class
          Dim objLoginForm As frmLogin
          objLoginForm = New frmLogin()

          'Call the Form Object Method so Object can Display itself
          objLoginForm.ShowDialog()


          End Sub


  End Module
```

- The remaining steps of the algorithm are implemented via the Event-Handlers of the form.
- IN THIS VERSION, THE FORM EVENT-HANDLER WILL CONTAIN CODE TO DISPLAY THE FORM:

2. If you user clicks the OK button:
   - **d)** Extract the username
   - **e)** Extract the password
   - **f)** Display a message stating that the user entered a username & password

3. If you user clicks the Cancel button:
   - **b)** Display a message stating that the user cancelled the operation
   - **c)** Clear the text boxes

- As in the previous example, the algorithm is implemented in the FORM as follows:

```vb
Public Class frmLogin

    Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.Click
        'Prompt the user via message box
        MessageBox.Show("Username = " & txtUsername.Text & " and Password = " & txtPassword.Text)

    End Sub

    Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCancel.Click
        'Clear the text boxes
        txtUsername.Text = ""
        txtPassword.Text = ""
    End Sub

End Class
```

Step 8: Build or Compile the Program

❑ Now Compile or Build the program to generate the executable .EXE file.

```
------ Build started: Project: LoginModWinAppl, Configuration: Debug Any CPU ------
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Vbc.exe /noconfig /imports:Microsoft.VisualBasic,System,System.Collections
LoginModWinAppl -> C:\Documents and Settings\rodriq_a\My Documents\NYCTC\CS608\Code\ClassNotes\ClassNotesModuleDrivenWin
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Step 9: Execute or Run the Program

❑ To run or execute the program do the following:
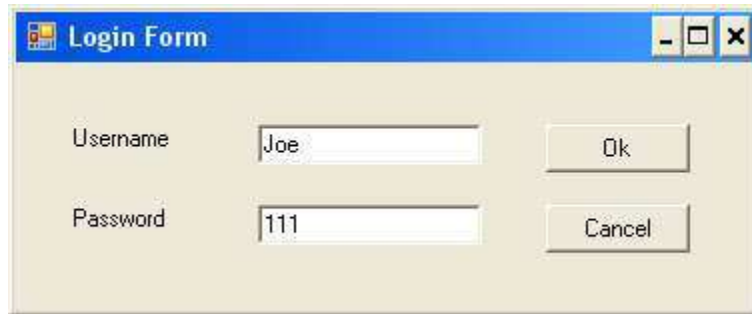
- ▪ The three methods to executes are:
  - 5) In the Menu Bar select **Debug|Start**.
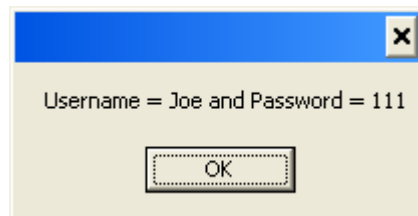 - 6) Or using the keyboard use the F5 key.
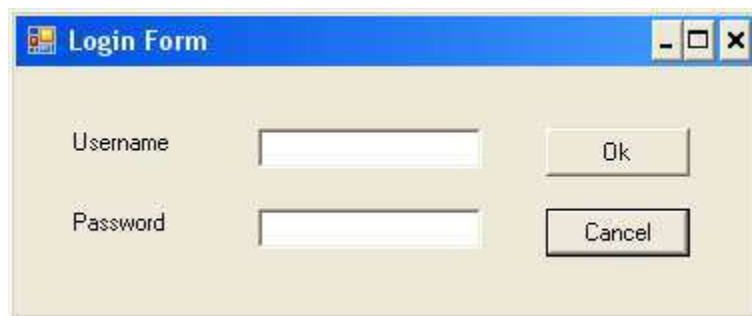 - 7) Or Click on the Start Icon in the Toolbar: ▶

  - 8) Or Navigate to the program folder\bin directory and double-click on the executable file *LoginFormWinApp.exe*

HOW IT WORKS:
- ▪ Control, is done by the **Sub Main()** method of the module.
- ▪ This method, displays the Login Form as a MODAL form using the *ShowDialog()* method.
- ▪ THERE AFTER, THE USER CAN USE THE FORM TO TRIGGER THE CONTROL EVENT-HANDLERS TO PERFORM THE PROCESSING REQUIRED.
- ▪ WHEN THE FORM _HIDES_ OR IS CLOSED DOES THE FLOW OF THE PROGRAM CONTINUES AND THE END OF THE SUB MAIN METHOD IS REACHED AND THE PROGRAM ENDS

2.5.4 Sample Program 4: Module-Driven Windows Application– Login Request Application Version 2 (Little or NO Processing Inside Form) (Best Practice!)

❑ This version of the Module-Driven Windows Application is almost the same as the previous version with ONE EXCEPTION; in the form we add very little code. ALL WORK OR PROCESSING IS DONE FROM THE MODULE.
❑ **REMEMBER, IN A *MODULE-DRIVEN APPLICATION*, CONTROL OF THE PROGRAM EXECUTION IS DONE BY THE SUB MAIN(), WHEN SUB MAIN() ENDS, THE PROGRAM ENDS AS WELL!**
❑ The program statement is the same as previous example:

- Create a Windows Application that displays a Login Form on the screen requesting a Username & Password
- The values entered are displayed via a message box when the user clicks an OK. When the user clicks the Cancel button, the text boxes are cleared.

❑ The steps and code are as follows:

Step 1: Open the Visual Studio IDE and invoke the Start Page & Select New Project

## Step 2: In the New Project Dialog select Windows Application

Step a – Project type

Step b – Window Application

Step d – Project Path

Step c – Project Name

Step 3: IDE is invoked

Step 4: Add a Module Object to the Project & Rename Form

❑ Since this is a *Module-Driven Windows Application* a Standard Module Object is required in your application.
❑ Add a Standard Module to the project. This module will contain code that will be public to the entire project.
❑ Name the Form to frmLogin Form
❑ The steps are as follows:
    1. In the Menu bar click **Project|Add Module…** to invoke the Add New Item screen
    2. In the *Template* section select *Module*, click Open.
    3. The module is now part of the project in the IDE.
    4. Rename the Form to frmLogin
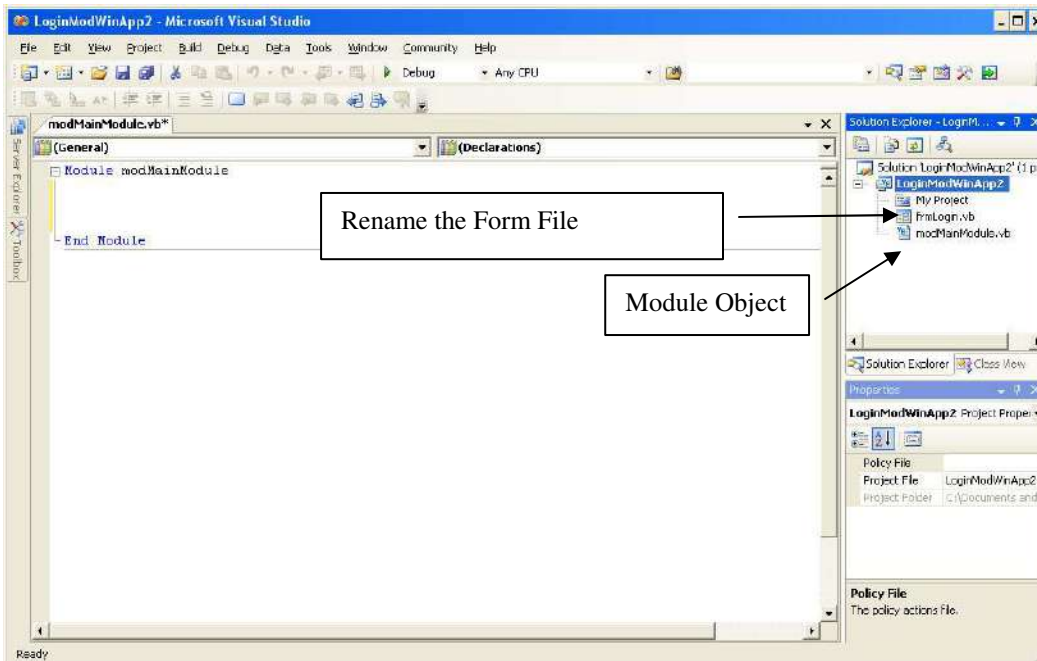
Step 5: Set Project Properties Startup Object

❑ Set the Startup Object to *Sub Main()* Procedure.
❑ The *Project Properties Window* is invoked as follows:
    ▪ In the **Solution Explorer**, simply *Right-Click* on the *project name LoginModWinApp2* and select Properties
    ▪ Uncheck the "*Enable application framework*" check box.
    ▪ In the Startup Object section, click the drop-down arrow and select *Sub Main()*.
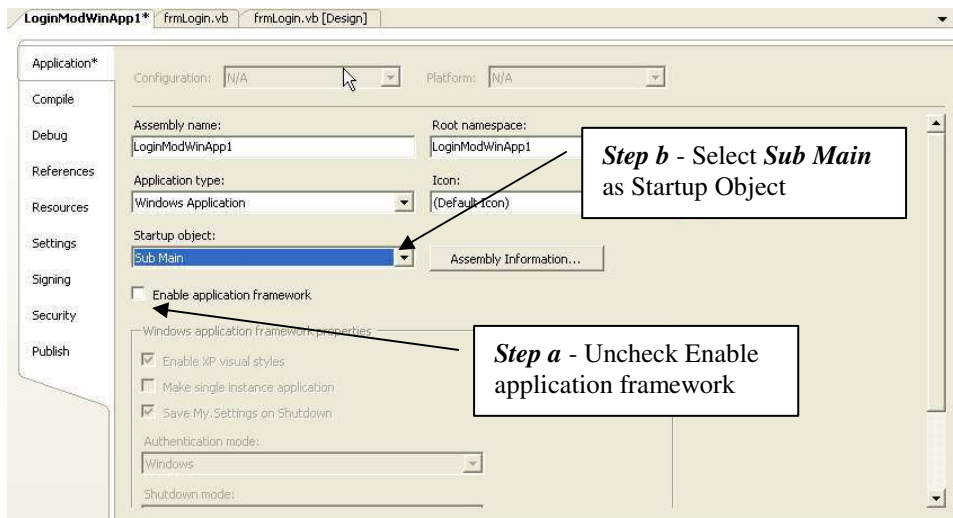
Step 6: Plan and Design the User Interface & Program Code

❑ As the previous example, here is where all the thinking, planning , designing & implementing your program.

Three Step Process
❑ Creating an Application involves a Two Phase process, I) **Planning** II) **Programming**.
❑ Each Part is broken into three steps:

    I.   **Planning**
  II.   **Design Phase:**

        1. *Design* the User-Interface (GUI) – You draw a sketch of what the Front-End or GUI will look like and the Control Objects required.
               ▪ We are use the same design as the previous example:

```
┌─────────────────────────────────────────────────┐
│                                                  │
│                                                  │
│   Username    ┌──────────────┐   ┌───────────┐   │
│               │              │   │  OK       │   │
│               └──────────────┘   └───────────┘   │
│                                                  │
│   Password    ┌──────────────┐   ┌───────────┐   │
│               │              │   │  Cancel   │   │
│               └──────────────┘   └───────────┘   │
│                                                  │
└─────────────────────────────────────────────────┘
```

        2. *Plan* the Properties or attributes to the objects in the GUI – We will use the same controls and properties as the previous example:

Object	Property	Value
Form 1	Name Text	**frmLogin** **Login Form**
Label1	Name Text	**lblUsername** **Username**
Label2	Name Text	**lblPassword** **Password**
Textbox1	Name	**txtUsername**
Textbox1	Name	**txtPassword**
Command button 1	Name Text	**btnOK** **OK**
Command button 2	Name Text	**btnCancel** **Cancel**

3. *Plan the code* – In this step, you plan (THINK) the code or steps required for the program to run. The thinking and analysis are the same as previous example.

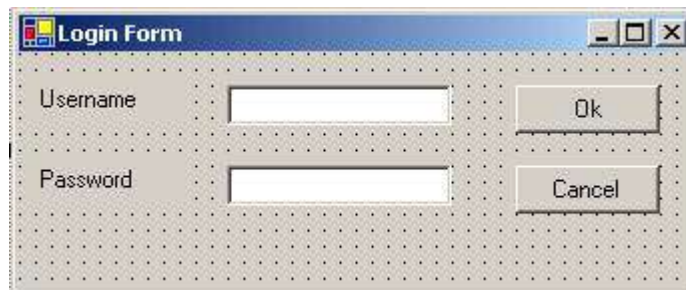❏ Derive the algorithm the same as previous example:

    4. Display the Login Screen
    5. If you user clicks the OK button:
        d) Extract the username
        e) Extract the password
        f) Display a message stating that the user entered a username & password

    6. If you user clicks the Cancel button:
        c) Display a message stating that the user cancelled the operation
        d) Clear the text boxes

Step 7: Implement the User Interface Form & Add Program Code

❏ As the previous example, we add controls to the Form and create the program code.

III. Programming:

1. *Create* the User-Interface (GUI) following the same guidelines as the previous example:
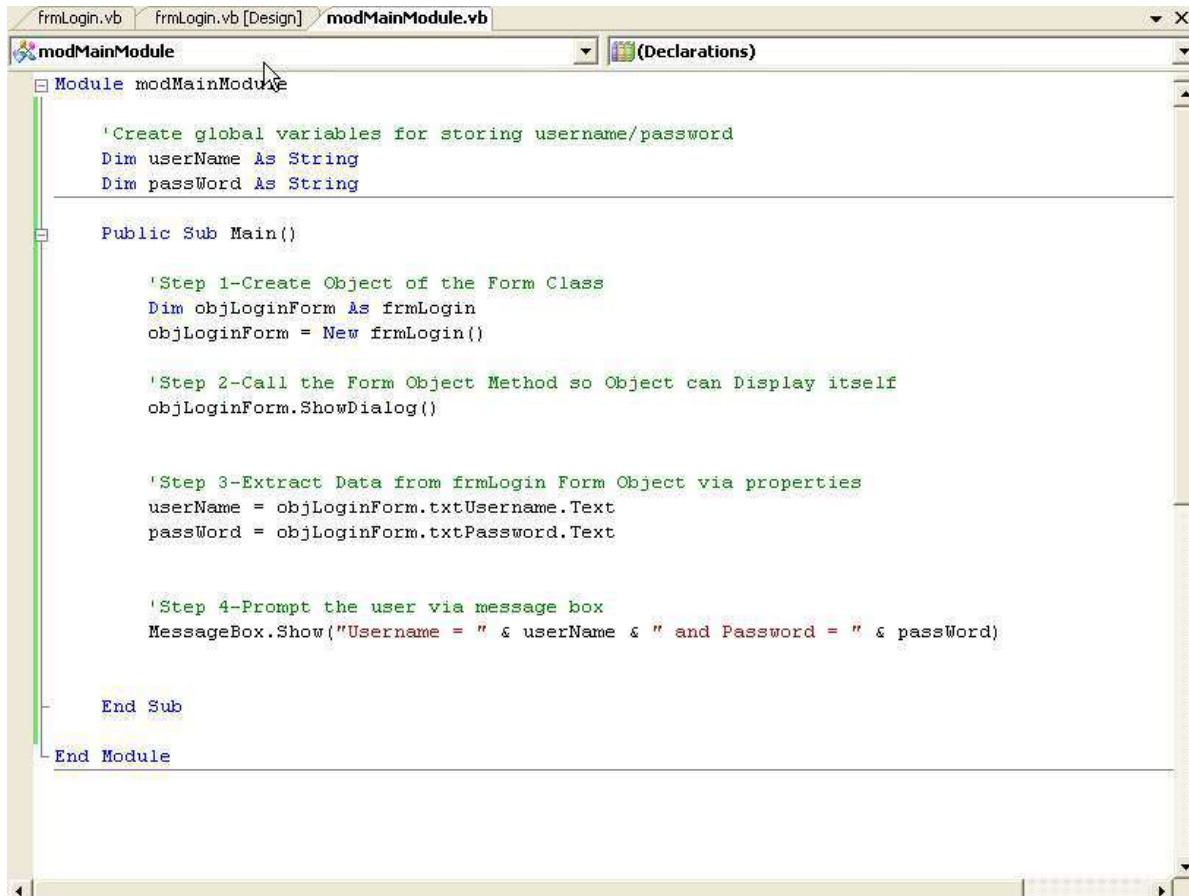

2. *Derive the Algorithm* – Create the algorithm using Pseudo-code:

▪ The algorithm derived was as follows:

    1. Display the Login Screen
    2. If you user clicks the OK button:
        g) Extract the username
        h) Extract the password
        i) Display a message stating that the user entered a username & password

    3. If you user clicks the Cancel button:
        d) Clear the text boxes

3. *Write the code* – Use the syntax or programming code to create the program base on the algorithm previously derived:

▪ So to begin with we need to control the program from Sub Main. This means we need to create the Sub Main procedure in the Module as follows:

- Now we need to implement the algorithm each step at a time.
- Note that in this example, ALL PROCESSING IS BIENG DONE IN THE MODULE NOT IN THE FORM. Thus we are separating INTERFACE from IMPLEMENTATION.

    a) In the Module, a Form object is displayed

    b) The username/password is extracted from the Form Object

    c) Finally the data is displayed to the user

```vb
Module modMainModule

    'Create global variables for storing username/password
    Dim userName As String
    Dim passWord As String

    Public Sub Main()

        'Step 1-Create Object of the Form Class
        Dim objLoginForm As frmLogin
        objLoginForm = New frmLogin()

        'Step 2-Call the Form Object Method so Object can Display itself
        objLoginForm.ShowDialog()

        'Step 3-Extract Data from frmLogin Form Object via properties
        userName = objLoginForm.txtUsername.Text
        passWord = objLoginForm.txtPassword.Text

        'Step 4-Prompt the user via message box
        MessageBox.Show("Username = " & userName & " and Password = " & passWord)

    End Sub

End Module
```

- The remaining steps of the algorithm are implemented via the Event-Handlers of the form.

- IN THIS VERSION, THE FORM EVENT-HANDLER WILL CONTAIN NO PROCESSING CODE. THE FORM ONLY HIDES ITSELF BY CALLING THE METHOD:

  Me.Hide()

- The Form CODE is implemented as follows:

```
frmLogin.vb    frmLogin.vb [Design]    modMainModule.vb                                    ▼ ✕
btnOK                                              ▼  ⚡ Click                                ▼
☐ Public Class frmLogin

    Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.Click
        'Form Hides itself
        Me.Hide()

    End Sub

    Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCancel.Click
        'Clear the text boxes
        txtUsername.Text = ""
        txtPassword.Text = ""
    End Sub


End Class
```

HOW IT WORKS:
- Since Form object is displayed from the MODULE using the *ShowDialog()* method, this means the Form was displayed as a **Modal Form**.
- Modal Forms means that control of the program execution stops as long as the Form is being displayed. You cannot access the application or navigate to any part of the program.
- ONLY WHEN THE FORM **_HIDES_** OR IS CLOSED DOES THE FLOW OF THE PROGRAM CONTINUES.
- SINCE WE ARE ONLY **HIDING** THE FORM OBJECT, THE FORM OBJECT STILL EXIST IN MEMORY AND CAN BE ACCESSED VIA its PROPERTIES. THIS IS HOW THE DATA FROM THE FORM IS EXTRATED IN THE MODULE.

Step 8: Build or Compile the Program

❑ Now Compile or Build the program to generate the executable .EXE file.

```
Output                                                                              ▼ ⏷ ✕
Show output from:  Build                    ▼  🔲  🖰 🖰  🗒 🖾
------ Build started: Project: LoginModWinAppl, Configuration: Debug Any CPU ------
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Vbc.exe /noconfig /imports:Microsoft.VisualBasic,System,System.Collections
LoginModWinAppl -> C:\Documents and Settings\rodriq_a\My Documents\NYCTC\CS608\Code\ClassNotes\ClassNotesModuleDrivenWin
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Step 9: Execute or Run the Program

❑ To run or execute the program do the following:

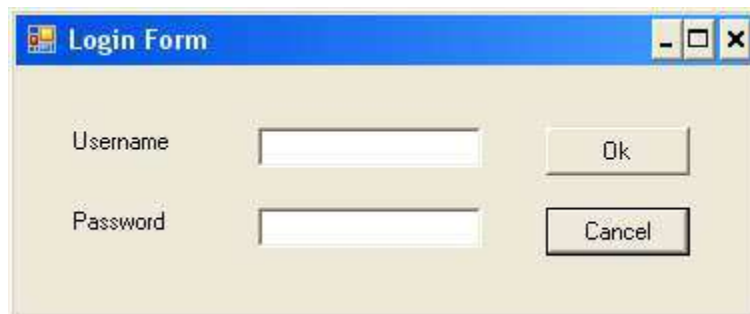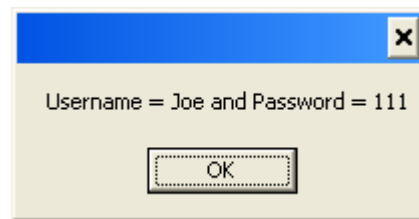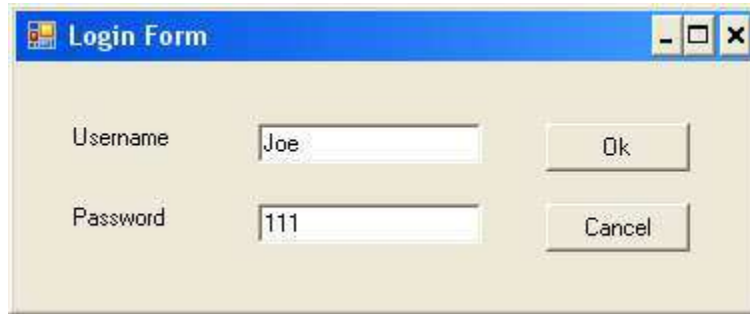- The three methods to executes are:
  9) In the Menu Bar select **Debug|Start**.
  10) Or using the keyboard use the F5 key.
  11) Or Click on the Start Icon in the Toolbar: ▶

  12) Or Navigate to the program folder\bin directory and double-click on the executable file *LoginFormWinApp.exe*

HOW IT WORKS:

- Control, is done by the **Sub Main**() method of the module.
- This method, displays the Login Form as a MODAL form using the *ShowDialog()* method.
- ONLY WHEN THE FORM **HIDES** OR IS CLOSED DOES THE FLOW OF THE PROGRAM CONTINUES.
- SINCE WE ARE ONLY **HIDING** THE FORM OBJECT, THE FORM OBJECT STILL EXIST IN MEMORY AND CAN BE ACCESSED VIA its PROPERTIES. THIS IS HOW THE DATA FROM THE FORM IS EXTRATED IN THE MODULE.
- WHEN THE END OF THE SUB MAIN METHOD IS REACHED THE PROGRAM ENDS

2.5.5 Homework

❑ The following HW is due next section.

Homework # A

❑ Problem statement.

1. **Implement Sample Program 1** – Console Login Screen Application
   - Read the section and follow steps to create the program

2. **Implement Sample Program 2** – Form-Driven Login Screen Application
   - Read the section and follow steps to create the program

3. **Implement Sample Program 3** – Module-Driven Login Screen Application (Processing on Form)
   - Read the section and follow steps to create the program

4. **Implement Sample Program 4** – Module-Driven Login Screen Application (No Processing on Form)
   - Read the section and follow steps to create the program
   - **IMPORTANT! Note that this example will be the foundation for all HW's and future projects**.