

# SIT 221: Event-Driven Programming

## Lecture 1 : Introduction to Event Driven Programming

# Software Development

- The term "software" refers to a computer program, or a collection of programs, that control the computer's hardware in order to achieve some purpose.
- Programs are written to solve a particular problem, or to perform a specific task.
- They are written by programmers, who must translate the requirements for solving the problem or carrying out the task into a language that the computer can understand.
- Computers are highly complex machines that can execute millions of instructions per second, but they have no inherent intelligence, and will only do what they are instructed to do by the programmer.
- For this reason, programs must be carefully designed, correctly coded, and thoroughly tested.

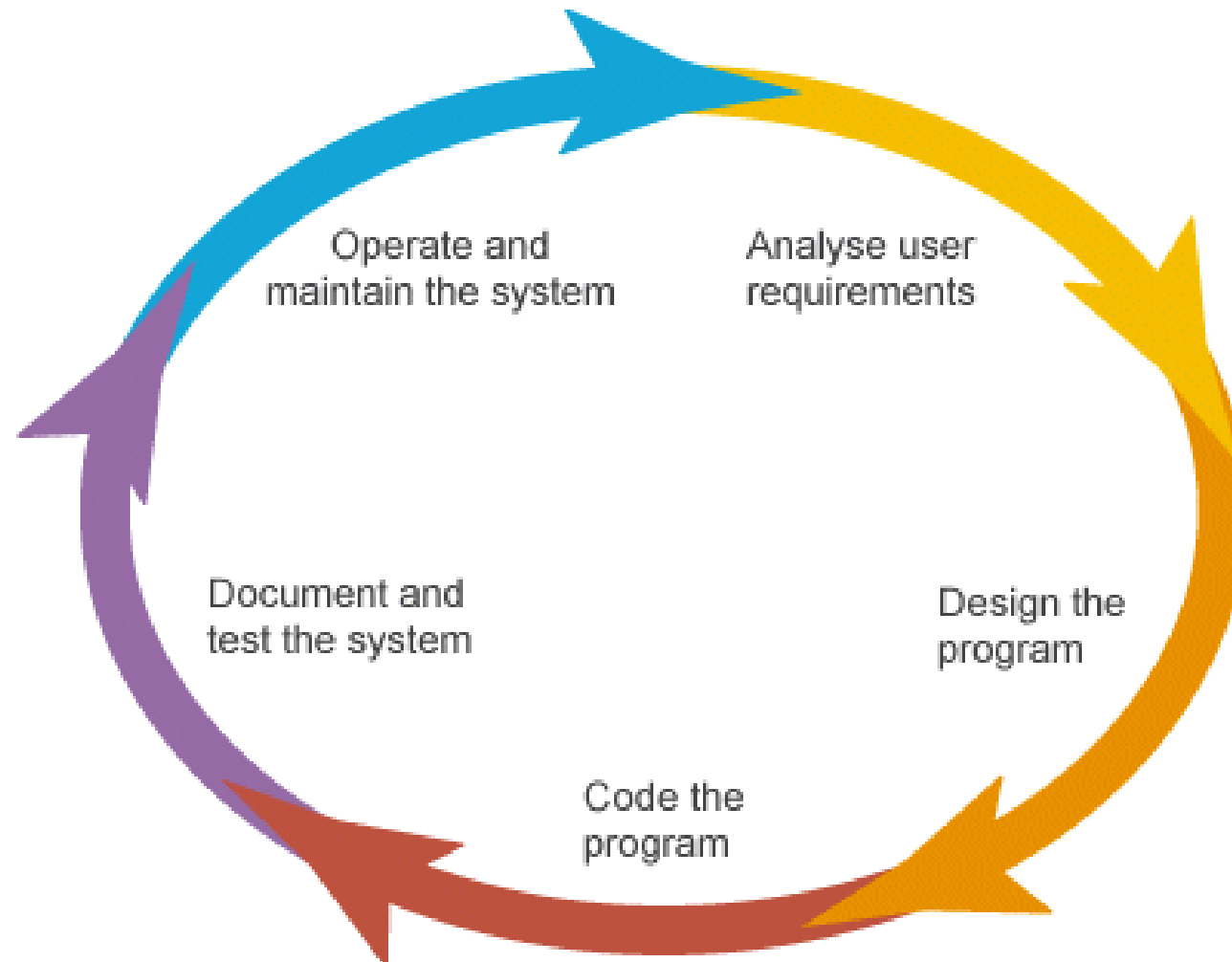
- One of the main tasks in any software development project will be to determine what the inputs and outputs of the program will be, i.e. what data will be fed *into* the program (input), and what data is expected as a result of running the program (output).
- The next step will be to determine what processing needs to be carried out on the input data in order to produce the required output.
- The complexity of the processing involved will depend on the size of the software project.
- A simple model of a software system is shown below.



A simple model of a software system

# The Software Development Life Cycle

- As with most undertakings, *planning* is an important factor in determining the success or failure of any software project. Essentially, good project planning will eliminate many of the mistakes that would otherwise be made, and reduce the overall time required to complete the project.
- As a rule of thumb, the more complex the problem is, the more thorough the planning process must be. Most professional software developers plan a software project using a series of steps generally referred to as the *software development life cycle* .
- A number of models exist that differ in the number of stages defined, and in the specific activities that take place within each stage.
- The following example is a generic model that should give you some idea of the steps involved in a typical software project.



# Analysis of user requirements

- During this stage, the problem is defined so that a clear understanding can be gained of what the system should do, i.e. what the inputs to the system are, what the output should be, and the operational parameters within which the system is expected to work. If the new system is to replace an existing system, the problem may be defined in terms of the additional or enhanced functionality that is required.

# Program design

- In this stage, a solution to the problem is designed by defining a logical sequence of steps that will achieve each of the stated system objectives. Such a sequence of steps is often referred to as an *algorithm*.
- Some of the methods used to define program algorithms are described later in this section, and include flowcharts and pseudocode.
- These tools allow the program designer to break a given problem down into a series of small tasks which the computer can perform to solve the problem.
- The user interface will also be designed during this stage, and will determine how input is obtained, how output is displayed, and what controls are available to the user.

# Program coding

- This stage, sometimes known as the *implementation* stage, is where the algorithms are translated into a programming language, and tends to be the longest phase of the development life-cycle.
- In this case, we are using Visual Basic to write the program.



# Documentation and testing

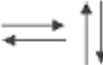
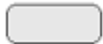
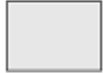






- The documentation of the program fulfils two main objectives.
- The first is to provide a technical reference to facilitate ongoing maintenance and development of the software itself.
- The second is to provide user documentation, i.e. a set of instructions that inform the user about the features of the software and how to use them.
- The aim of software testing is to find any errors ("bugs") in the program, to eliminate those errors (a process known as "debugging"), and as far as is reasonably practicable should be sufficiently rigorous to ensure that the software will function as expected under all foreseeable circumstances.

# Operating and maintaining the system

- Once the software has been "rolled out" and any necessary user training has been completed, it will be necessary to monitor the performance of the system over time to ensure that it is behaving as expected.
- The system will need to be maintained, and parts of it will need to be upgraded from time to time to handle evolving user needs or to cope with new problems.
- Eventually, as the system ages, it may no longer be able to adequately cope with the demands of a growing number of users, take advantage of advances in hardware technology, or adapt to a constantly changing environment.
- When this time comes, the system will need to be decommissioned and replaced by a new system.
- Hence, the software development life cycle will begin again.

# Design Tools: - Using Flowcharts

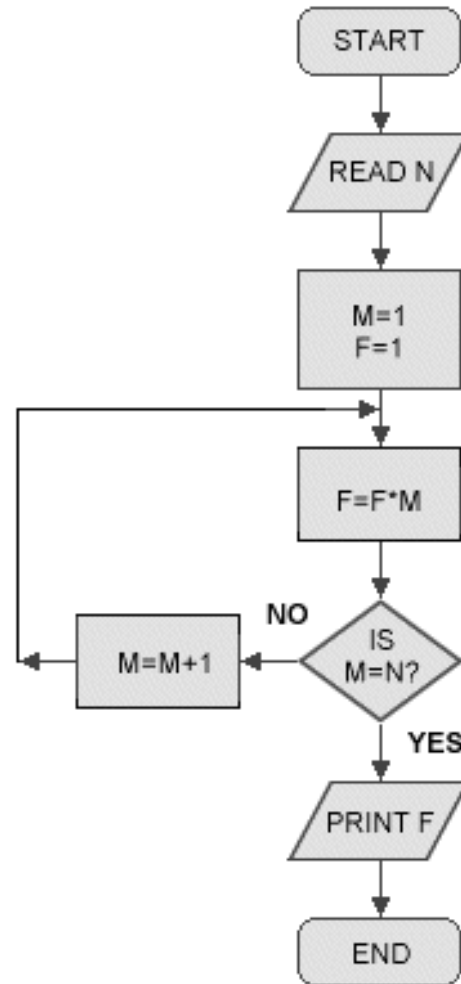
- A flowchart is a way of visually representing the flow of data through an information processing system, the operations performed within the system, and the sequence in which they are performed.
- A program flowchart describes the sequence of operations required to solve a given problem. A programmer will often create a flowchart before writing a program. The flowchart is drawn according to defined rules, using standard flowchart symbols.
- Flowcharts help us to understand the logic of complex problems, and make it easier to write programs in a high level programming language. Some commonly used flowcharting symbols are shown below.
- Flowcharts are a good way of communicating the logic of a system to others, and allow problems to be analysed effectively. They can also form part of the program's documentation, and are useful for facilitating efficient coding and debugging.
- Flowcharts can, however, become complex and unwieldy if the program in question is large and complex. They may also need to be completely redrawn if changes are required, depending on how they have been produced.

ANSI Flowchart Symbols		
Symbol	Name	Description
	Flowline	Lines that connect other flowchart symbols and indicate the direction of logic
	Terminal	Indicates the start or end of a task
	Processing	Arithmetic or data-manipulation operations
	Input/Output	Indicates an input or output operation
	Decision	Decision making and branching
	Connector	Used to join different parts of a program
	Offpage	Indicates that the flowchart continues on another page.
	Predefined	Represents a group of statements that perform some processing task
	Annotation	Provides additional information about another flowchart symbol

# Flowchart guidelines

- Before you begin, list the requirements in a logical order
- The flowchart should be clear, unambiguous and easy to follow
- Normal direction of flow is left to right or top to bottom
- Only one flow line should come from a process symbol
- Only one flow line should enter a decision symbol. Several flow lines (one for each possible answer) may exit the decision symbol
- Only one flow line is used in conjunction with the terminal symbol
- Use only brief comments within symbols (use the annotation symbol to describe data or computational steps more clearly)
- For a complex flowchart, use connector symbols to reduce the number of flow lines. Avoid intersecting flow lines
- Ensure the flowchart has a logical beginning and end point
- Test the validity of the flowchart with simple test data

# An example flowchart - A program for computing the factorial of N



An example flowchart

# Using Pseudocode

- Pseudocode is a compact and informal high-level description of a computer programming algorithm that uses the structural conventions of programming languages, but omits detailed subroutines, variable declarations or language-specific syntax.
- The programming language is augmented with natural language descriptions of the details, where convenient.
- Flowcharts can be thought of as a graphical form of pseudocode.
- As the name suggests, pseudocode does not need to obey the syntax rules of a specific programming language, and there is no standard format, although the programmer may imitate the appearance of a particular programming language.
- Details not relevant to an algorithm are usually omitted. Blocks of code (for example that contained within a loop) may be described in a single natural language sentence.
- Pseudocode can thus vary widely in style. Programming textbooks often use pseudocode to describe algorithms so that all programmers can understand them, whatever programming language they use, although the conventions used are usually clearly defined. A programmer will often begin with a pseudocode description of a program or algorithm, and then simply translate this into their chosen programming language.

## A pseudocode example:

```
if credit card number is valid  
    execute transaction based on number  
    and order  
else  
    show a generic failure message  
end if
```



# A Pseudocode Standard

- Pseudocode is a kind of structured English for describing algorithms that allows the programmer to focus on the logic of the algorithm without being distracted by the details of programming language syntax.
- Writing the program is then simply a matter of translating the pseudocode into source code.
- The vocabulary used is usually that of the problem domain, with the only stipulation being that the logic must be specified in sufficient detail to allow the source code to be derived without breaking the problem down further.
- There is no universal standard for pseudocode, but it is helpful to follow certain conventions, some of which are described below.

# Program flow:

- sequence - a linear progression where tasks are performed sequentially
- if then else - a choice is made between two alternative courses of action
- while - a loop with a simple conditional test at the beginning
- repeat until - a loop with a simple conditional test at the bottom
- case - choose from several actions, depending on the value of an expression
- for - a counting loop

- **Sequence** - a sequence is indicated by writing one action after another, each on a separate line, and each at the same level of indentation. The actions are performed in the order in which they appear, e.g.

```
READ height of rectangle
```

```
READ width of rectangle
```

```
COMPUTE area as height times width
```

- ***If then else*** - this construct defines a simple two-way choice based on whether or not a condition equates to true or false, e.g.

```
if hoursWorked > normalMax then  
    display overtime message  
  
else  
    display regular time message  
  
end if
```

- **While** - this construct specifies a loop with a test at the beginning. The loop is only entered if the condition is true, and the specified action (or sequence) is performed for each iteration of the loop. The condition is evaluated before each iteration.

```
while population < limit  
    compute population as population +  
    births - deaths  
  
end while
```

- ***Repeat until*** - similar to the while loop, except that the test is performed at the end of the loop, so that the specified sequence will be performed at least once. The loop repeats if the condition is false, and terminates when it becomes true. The general form is:

```
repeat  
  
    sequence  
  
until condition
```

- **Case** - this construct allows the program to execute one of a number of alternative sequences, depending on which of a set of mutually exclusive conditions is true. It takes the form:

```
case expression of  
    condition 1 : sequence 1  
    condition 2 : sequence 2  
    ...  
    condition n : sequence n  
otherwise:  
    default sequence  
end case
```

- The otherwise clause is optional, and the same sequence may be associated with more than one condition.

```
case grade of  
    A: points = 4  
    B: points = 3  
    C: points = 2  
    D: points = 1  
    E: points = 0  
end case
```



- **For** - this construct (often called a counting loop) causes a loop to be repeated a specific number of times. The general form is:

```
for iteration bounds  
    sequence  
end for
```

- Nested constructs - Constructs can be nested inside each other, and this should be made clear by appropriate use of indentation, e.g.

- ```
set total to zero

repeat

    read temperature

    if temperature > freezing then

        increment total

    end if

until temperature < zero

print total
```

- Invoking sub-procedures - Use the call keyword, e.g.

```
CALL AvgAge with StudentAges
```

```
CALL Swap with CurrentItem and TargetItem
```

```
CALL Account.debit with CheckAmount
```

```
CALL getBalance RETURNING aBalance
```

# Visual Basic.Net 2019

- In this first set, we will do an overview of how to build a Windows application using Visual Basic .NET.
- You'll learn a new vocabulary, a new approach to programming, and ways to move around in the Visual Basic .NET environment.
- Once finished, you will have written your first Visual Basic .NET program.

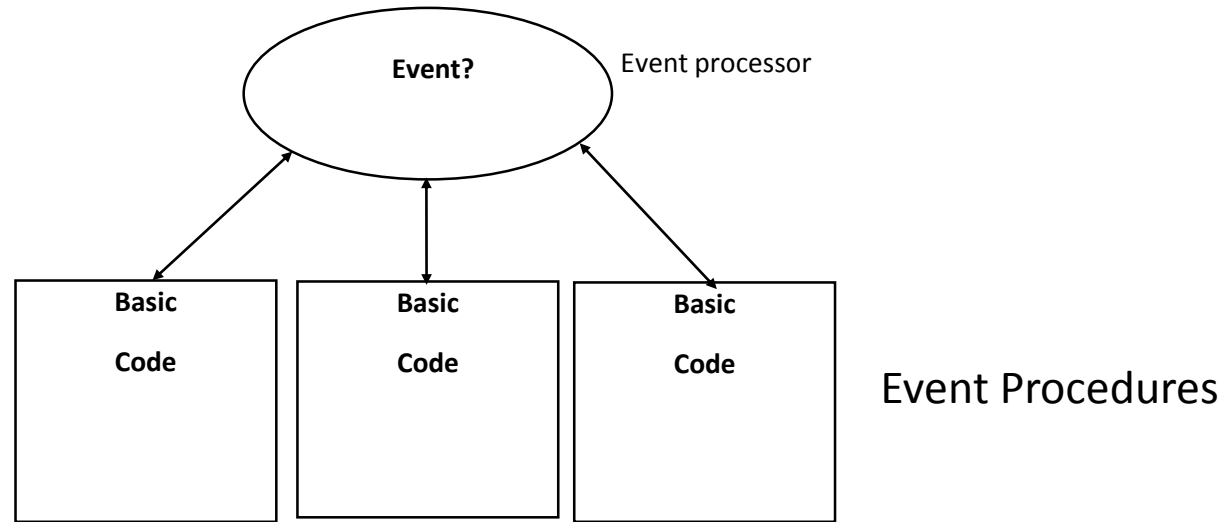
# What is Visual Basic .NET?

- **Visual Basic .NET** is part of a grand new initiative by Microsoft. It is a complete re-engineering of Visual Basic for the Microsoft .NET framework. With Visual Basic .NET, you are able to quickly build Windows-based applications (the emphasis in this course), web-based applications and, eventually, software for other devices, such as palm computers.
- Windows applications built using Visual Basic.NET feature a Graphical User Interface (GUI). Users interact with a set of visual tools (buttons, text boxes, tool bars, menu items) to make an application do its required tasks. The applications have a familiar appearance to the user.

- As you develop as a Visual Basic .NET programmer, you will begin to look at Windows applications in a different light. You will recognize and understand how various elements of Word, Excel, Access and other applications work. You will develop a new vocabulary to describe the elements of Windows applications.
- Visual Basic .NET Windows applications are **event-driven**, meaning nothing happens until an application is called upon to respond to some event (button pressing, menu selection, ...). Visual Basic .NET is governed by an event processor.

- As mentioned, nothing happens until an event is detected. Once an event is detected, a corresponding event procedure is located and the instructions provided by that procedure are executed.
- Those instructions are the actual code written by the programmer. In Visual Basic .NET, that code is written using a version of the BASIC programming language. Once an event procedure is completed, program control is then returned to the event processor.





- All Windows applications are event-driven. For example, nothing happens in Word until you click on a button, select a menu option, or type some text. Each of these actions is an event.
- The event-driven nature of applications developed with Visual Basic .NET makes it very easy to work with. As you develop a Visual Basic .NET application, event procedures can be built and tested individually, saving development time. And, often event procedures are similar in their coding, allowing re-use (and lots of copy and paste).

# Some Features of Visual Basic .NET

- All new, easy-to-use, powerful Integrated Development Environment (IDE)
- Full set of controls - you 'draw' the application
- Response to mouse and keyboard actions
- Clipboard and printer access
- Full array of mathematical, string handling, and graphics functions
- Can easily work with arrays of variables and objects

- Sequential file support
- Useful debugger and structured error-handling facilities
- Easy-to-use graphic tools
- Powerful database access tools
- Ability to develop both Windows and internet applications using similar techniques
- New common language runtime module makes distribution of applications a simple task

# Visual Basic .NET versus Visual Basic

- Let's get something straight right now – Visual Basic .NET is **not** a new version of Visual Basic.
- Visual Basic .NET is an entirely new product. If you are familiar with Visual Basic, Visual Basic .NET will look familiar, but there are many differences.
- And, for the most part, the differences are vast improvements over Visual Basic.
- When you realize that the BASIC language has not undergone substantial changes in 20 years, you should agree it was time for a clean-up and improvement.
- A few of the features of Visual Basic .NET, compared to Visual Basic:

## Features of Visual Basic .NET, Compared to Visual Basic:

- New Integrated Development Environment
- Uses Object-Oriented Programming (OOP) methods
- New controls and control properties
- Redesigned code window
- Zero-based arrays (no adjustable first dimension)
- Easier to use common dialog boxes
- Structured error-handling (no more On Error Go To)

- New menu design tools
- New techniques for working with sequential files
- All new graphics methods
- New approaches to printing from an application
- Improved support to incorporating help systems in applications
- New web forms for internet applications
- ADO.NET for database access

# A Brief Look at Object-Oriented Programming (OOP)

- Since Visual Basic was first introduced in the early 1990's, a major criticism from many programmers (especially those using C and C++) was that it was not a true object-oriented language. And, with that limitation, many dismissed Visual Basic as a “toy” language. That limitation no longer exists!
- Visual Basic .NET is fully **object-oriented**. For this particular course, we don't have to worry much about just what that means (many sizeable tomes have been written about OOP). What we need to know is that each application we write will be made up of **objects**.
- Just what is an object? It can be many things: a variable, a font, a graphics region, a rectangle, a printed document. The key thing to remember is that these objects represent reusable entities that are used to develop an application. This 'reusability' makes our job much easier as a programmer.

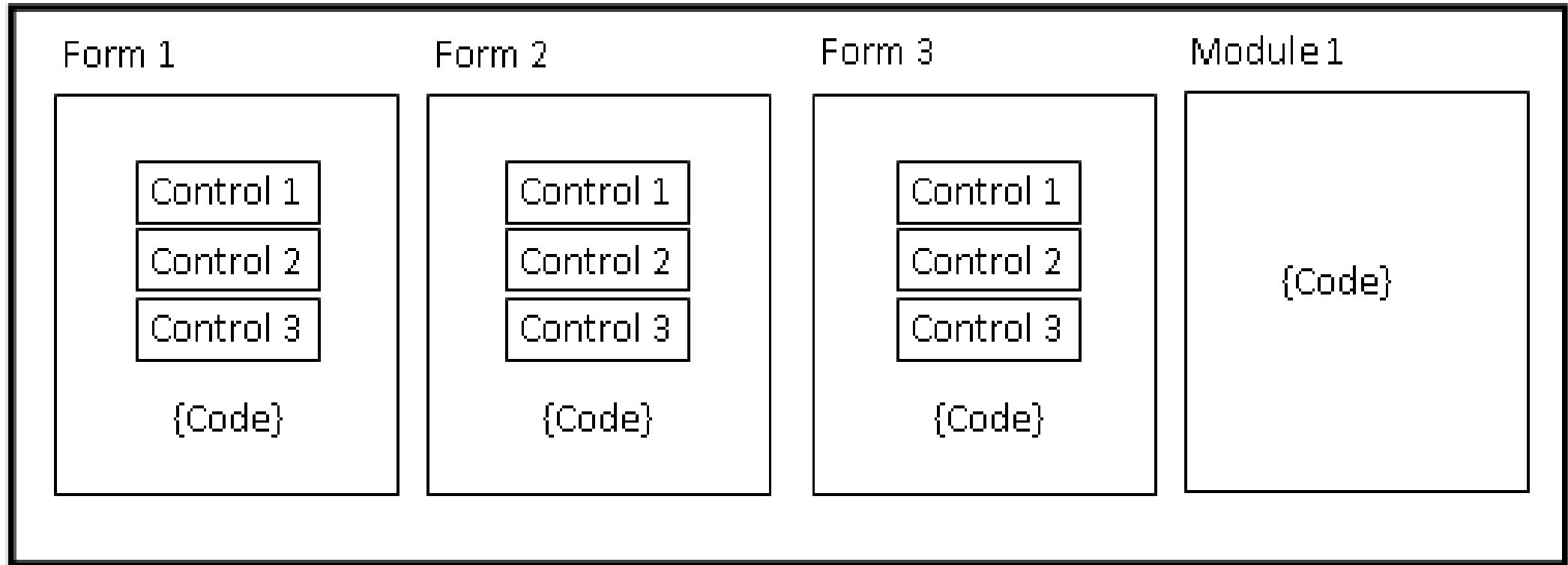
- In Visual Basic .NET, there are three terms we need to be familiar with in working with object-oriented programming: **Namespace**, **Class** and **Object**. **Objects** are what are used to build our application.
- We will learn about many objects throughout this course. Objects are derived from **classes**.
- Think of classes as general descriptions of objects, which are then specific implementations of a class. For example, a class could be a general description of a car, where an object from that class would be a specific car, say a red 1965 Ford Mustang convertible (a nice object!).
- Lastly, a **namespace** is a grouping of different classes used in the .NET world. One namespace might have graphics classes, while another would have math functions. We will see several namespaces in our work.



- For this course, if you remember **namespace**, **class** and **object**, you have sufficient OOP knowledge to build applications. The primary use for these terms is when searching for help on a particular topic. When seeking help, you need to know that an object comes from a class which comes from a namespace. Once you complete this course, you can further delve into the world of OOP. Then, you'll be able to throw around terms like inheritance, polymorphism, overloading, encapsulation, and overriding.
- The biggest advantage of the object-oriented nature of Visual Basic .NET is that it is no longer a “toy” language. In fact, Visual Basic .NET uses the same platform for development and deployment (incorporating the new Common Language Runtime (CLR) module) as the more esoteric languages (Visual C++ and the new Visual C#). Because of this, there should be no performance differences between applications written in Visual Basic .NET, Visual C++ .NET, or Visual C# .NET!

# Structure of a Visual Basic .NET Windows Application

- We want to get started building our first Visual Basic.NET Windows application. But, first we need to define some of the terminology we will be using. In Visual Basic .NET, a Windows **application** is defined as a **solution**.
- A solution is made up of one or more **projects**. Projects are groups of forms and code that make up some application. In most of our work in this course, our applications (solutions) will be made up of a single project. Because of this, we will usually use the terms application, solution and project synonymously.
- As mentioned, a project (application) is made up of forms and code. Pictorially, this is:



# Application (Project) is made up of:

- **Forms** - Windows that you create for user interface
- **Controls** - Graphical features drawn on forms to allow user interaction (text boxes, labels, scroll bars, buttons, etc.) (Forms and Controls are **objects**.)
- **Properties** - Every characteristic of a form or control is specified by a property. Example properties include names, captions, size, color, position, and contents. Visual Basic .NET applies default properties. You can change properties when designing the application or even when an application is executing.

- **Methods** - Built-in procedures that can be invoked to impart some action to a particular object.
- **Event Procedures - Code** related to some object or control. This is the code that is executed when a certain event occurs.
- **General Procedures - Code** not related to objects. This code must be invoked or called in the application.
- **Modules** - Collection of general procedures, variable declarations, and constant definitions used by an application.

- The application displayed above has three forms and a single module. Visual Basic.NET uses a very specific directory structure for saving all of the components for a particular application. When you start a new project (solution), you will be asked for a **Name** and **Location** (directory).
- A folder named **Name** will be established in the selected **Location**. That folder will be used to store all solution files, project files, form and module files (**vb** extension) and other files needed by the project.
-

- Two subfolders will be established within the Name folder: **Bin** and **Obj**. The **Obj** folder contains files used for debugging your application as it is being developed.
- The **Bin** folder contains your compiled application (the actual executable code or **exe** file). Later, you will see that this folder is considered the 'application path' when ancillary data, graphics and sound files are needed by an application.

In a project folder, you will see these files (and possibly more):

- **AssemblyInfo.vb** Information on how things fit together
- **SolutionName.sln** Solution file for solution named Solution Name
- **ProjectName.vbproj** Project file – one for each project in solution
- **ProjectName.vbproj.user** Another Project file – one for each project in solution
- **FormName.resx** Form resources file – one for each form
- **FormName.vb** Form code file – one for each form



# Steps in Developing a Windows Application

- The Visual Basic .NET Integrated Development Environment (IDE) (referred to as the Microsoft Development Environment, MDE, if using Visual Basic.NET 2003) makes building an application a straightforward process.
- There are three primary steps involved in building a Visual Basic.NET application:
  - ✓ **Draw** the user **interface** by placing controls on a Windows form
  - ✓ **Assign properties** to controls
  - ✓ **Write code** for control events (and perhaps write other procedures)

- These same steps are followed whether you are building a very simple application or one involving many controls and many lines of code.
- The event-driven nature of Visual Basic .NET applications allows you to build your application in stages and test it at each stage. You can build one procedure, or part of a procedure, at a time and try it until it works as desired.
- This minimizes errors and gives you, the programmer, confidence as your application takes shape.

- As you progress in your programming skills, always remember to take this sequential approach to building a Visual Basic .NET application. Build a little, test a little, modify a little and test again. You'll quickly have a completed application.
- This ability to quickly build something and try it makes working with Visual Basic .NET fun – not a quality found in some programming environments!
- Now, we'll start Visual Basic .NET and look at each step in the application development process.

# Visual Basic .NET Integrated Development Environment (IDE)

- The **Visual Basic .NET IDE** is where we build and test our application via implementation of the three steps of application development (draw controls, assign properties, write code).
- As you progress through this course, you will learn how easy-to-use and helpful the IDE is. There are many features in the IDE and many ways to use these features.
- Here, we will introduce the IDE and some of its features. You must realize, however, that its true utility will become apparent as you use it yourself to build your own applications.
- Several windows appear when you start Visual Basic.NET. Each window can be viewed (made visible or active) by selecting menu options, depressing function keys or using the displayed toolbar. As you use the IDE, you will find the method you feel most comfortable with.

- The title bar area shows you the name of your project and the name of the form currently displayed. Also shown in brackets is one of three words: **Design**, **Run** or **Break**. This shows the mode Visual Basic.NET is operating in. Visual Basic.NET operates in three modes.
  - ✓ **Design** mode - used to build application
  - ✓ **Run** mode - used to run the application
  - ✓ **Break** mode - application halted and debugger is available
- We focus here on the **design** mode. You should, however, always be aware of what mode you are working in.

- Under the title bar is the **Menu**. This menu is dynamic, changing as you try to do different things in Visual Basic .NET. When you start working on a project, it should look like this:



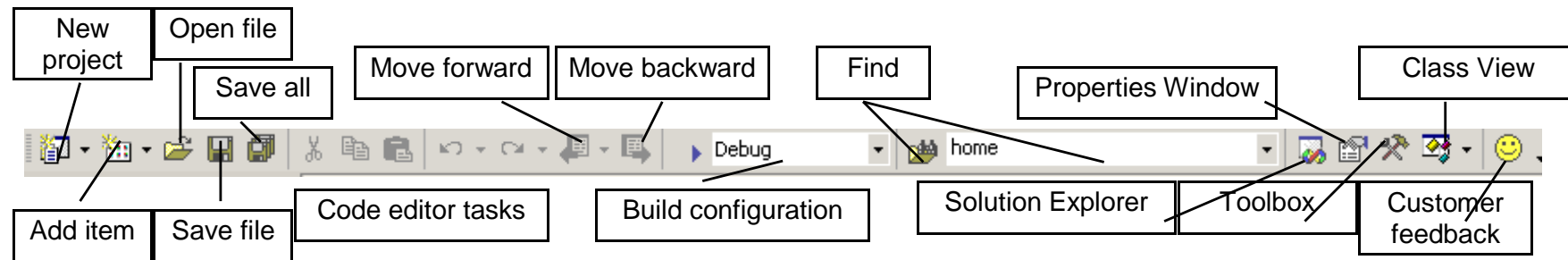
You will become familiar with each menu topic as you work through the course. Briefly, they are:

- **File** - Use to open/close projects and files. Use to exit Visual Basic .NET
- **Edit** - Used when writing code to do the usual editing tasks of cutting, pasting, copying and deleting text
- **View** - Provides access to most of the windows in the IDE
- **Project** - Allows adding files and objects to your application
- **Build** - Allows you to compile and run your completed application (go to Run mode)

- **Debug** - Comes in handy to help track down errors in your code (works when Visual Basic .NET is in **Break** mode)
- **Data** - Used when building database applications (not covered in this course)
- **Format** - Useful for manipulating controls placed on your forms
- **Tools** - Allows custom configuration of the IDE. Be particularly aware of the **Options** choice under this menu item. This choice allows you to modify the IDE to meet any personal requirements.
- **Window** - Lets you change the layout of windows in the IDE
- **Help** - Perhaps, the most important item in the Menu. Provides access to the Visual Basic .NET on-line documentation via help contents, index or search. Get used to this item!



- The View menu also allows you to choose from a myriad of toolbars available in the Visual Basic .NET IDE. Toolbars provide quick access to many features. The **Standard** (default) toolbar appears below the Menu:



- If you forget what a toolbar button does, hover your mouse cursor over the button until a descriptive tooltip appears. We will discuss most of these toolbar functions in the remainder of the IDE information.