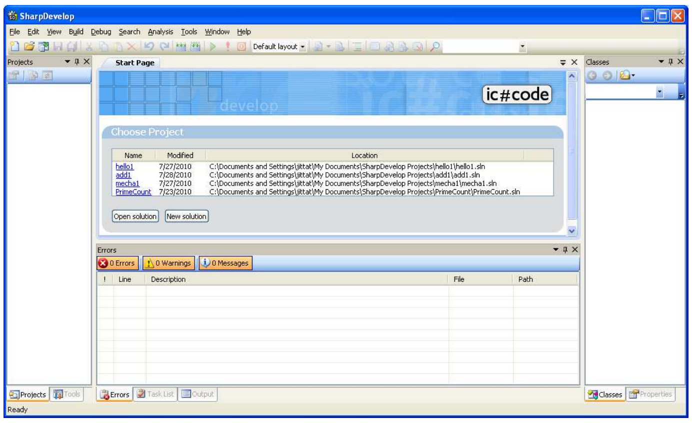
Name..ID.....................................Section...............

Lab 7 : Introduction to C# and Its Development Environment

In this lab, we start developing programs in C#. For Python programmers, C# program may look too tedious, but it is not very difficult to be comfortable with the language. The development environment that we will use for writing C# program is **Sharp Develop** (http://www.icsharpcode.net/opensource/sd/) .

Getting started with Sharp Develop

After calling Sharp Develop, the window as in Figure 1 appears.



Name	Modified	Location
hello1	7/27/2010	C:\Documents and Settings\jittat\My Documents\SharpDevelop Projects\hello1\hello1.sln
add1	7/28/2010	C:\Documents and Settings\jittat\My Documents\SharpDevelop Projects\add1\add1.sln
mecha1	7/27/2010	C:\Documents and Settings\jittat\My Documents\SharpDevelop Projects\mecha1\mecha1.sln
PrimeCount	7/23/2010	C:\Documents and Settings\jittat\My Documents\SharpDevelop Projects\PrimeCount\PrimeCount.sln

!	Line	Description	File	Path

Figure 1. Sharp Develop's first view.

A program in C# will be called a *solution* . In this course, we will develop two kinds of programs, i.e.,

Console Application

1. A **console application**, which is a program that interacts with users in text (as in Python).

และ

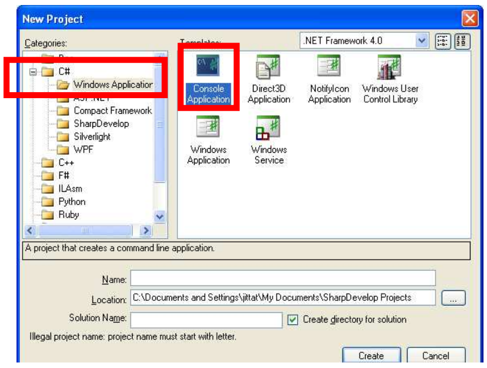
Windows Application

2. A **Windows application**, that interacts with users through graphical user interface, usually in windows.

We will start with console applications

1. A console application

We start by choosing **File > New > Solution**… from the menu. The **New Project** dialog appears as in Figure 2.



A project that creates a command line application.

Name:

Location: C:\Documents and Settings\jittat\My Documents\SharpDevelop Projects

Solution Name:

☑ Create directory for solution

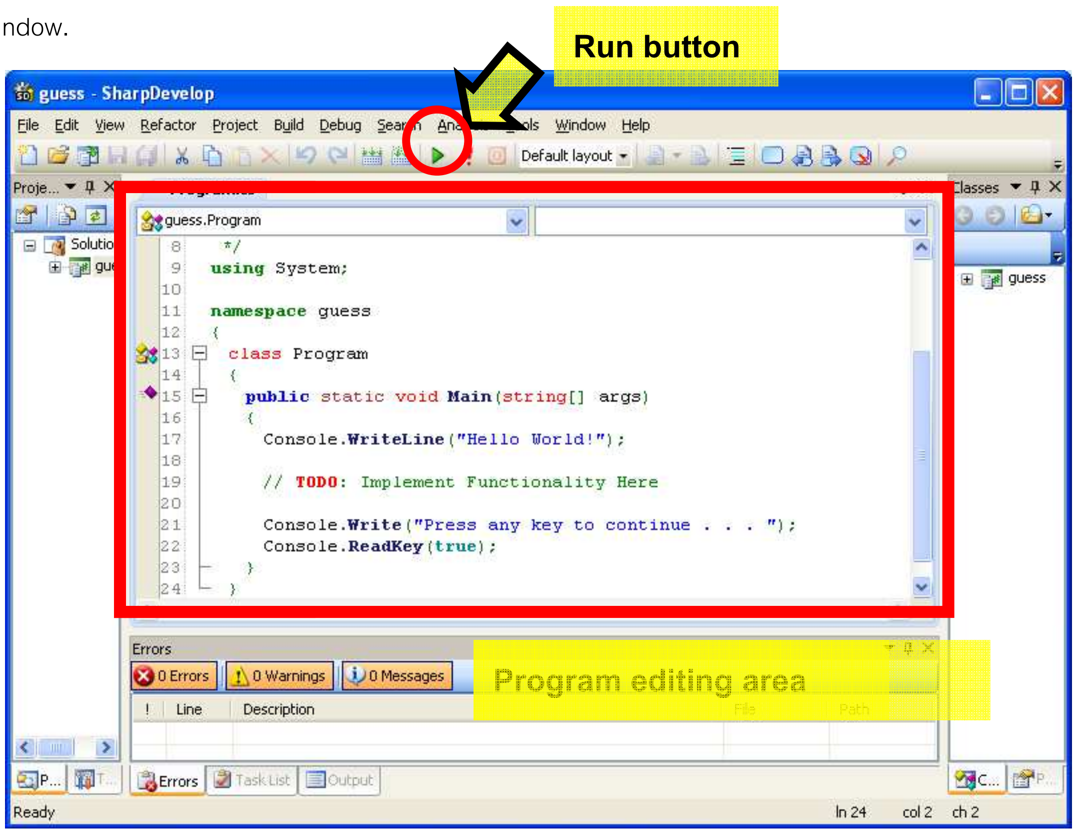
Illegal project name: project name must start with letter.

Follow these steps to create a solution:

1. Choose **C#** and **Windows Applications** in Categories.
2. In Templates area, pick **Console Application.**
3. Give the solution some name. For this solution, call it **`hello`**.
4. Click **Create** button.

A warning that a solution with this name exists may appear. In that case, just click “overwrite” so that Sharp Develop replaces the old copy with our new solution.

You will see a window as in Figure 3 after we have created a solution. We will edit our program in this window.



```
  */
using System;

namespace guess
{
  class Program
  {
    public static void Main(string[] args)
    {
      Console.WriteLine("Hello World!");

      // TODO: Implement Functionality Here

      Console.Write("Press any key to continue . . . ");
      Console.ReadKey(true);
    }
  }
```

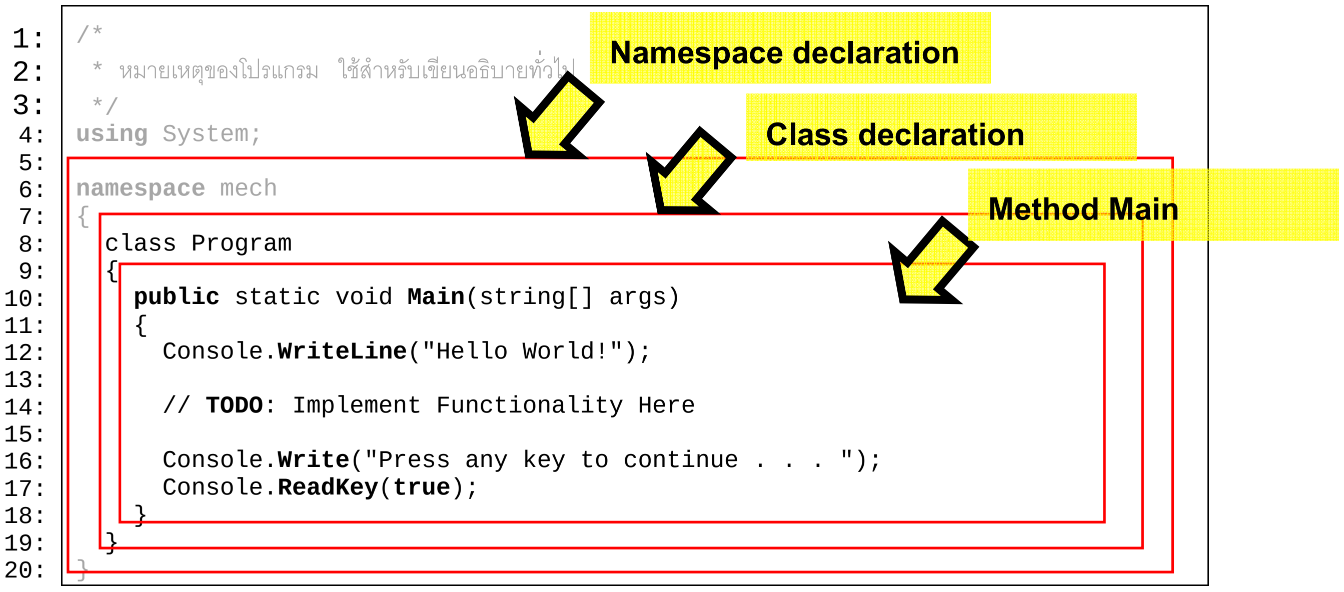
Figure 3 A Sharp Develop screen for editing and debugging programs

Sharp Develop creates a skeleton program based on the template we selected. You can see the program to be edited at the center. We can start the program by clicking the **Run** button ( ) from the toolbar.

Exercise 1: Click the Run button and write down the output. If the program stops, you can hit any key to end the program.

1.1 Program structures

The skeleton program is an example of standard C# program; it is shown in Figure 4.



```
 1: /*
 2:  * หมายเหตุของโปรแกรม  ใช้สำหรับเขียนอธิบายทั่วไป
 3:  */
 4: using System;
 5:
 6: namespace mech
 7: {
 8:  class Program
 9:  {
10:    public static void Main(string[] args)
11:    {
12:      Console.WriteLine("Hello World!");
13:
14:      // TODO: Implement Functionality Here
15:
16:      Console.Write("Press any key to continue . . . ");
17:      Console.ReadKey(true);
18:    }
19:  }
20: }
```

Figure 4. Basic structure of a C# program.

This program contains declaration of a namespace, a class, and a method, called **Main**. A method is very similar to a function in Python that it defines a new "command" that other parts of the program can call. This **Main** method is a special method that a C# program always starts at. In the beginning, we mainly write program in **Main** method. You should not worry too much about the other parts as we will look into that later.

Exercise 1.1.1: Good morning teacher.

From a program in Figure 4, delete lines 12 – 17 and replace them with the following statements.

```
Console.WriteLine("Good morning teacher.");
Console.WriteLine("How are you?");
Console.WriteLine("I'm fine thank you, and you?");
Console.ReadLine();
```

Click Run, and write down the output of the program.

Exercise 1.1.2: Good morning teacher again.

Change the program from Exercise 1.1.1 by replacing all occurrences of `Console.WriteLine` with `Console.Write.` What is the output?

From the last two exercises, what is the difference between the **`Console.Write`** statement and the **`Console.WriteLine`** statement.

Exercise 1.1.3 : Console.ReadLine()

Programs from exercises 1.1.1 and 1.1.2 end with statement **`Console.ReadLine();`** . Remove that line and try to run the program. What is the output?

Why do we put **`Console.ReadLine();`** at the end of method Main?

1.2 Easy program

Consider the following Python program.

```
name = input("Enter item's name: ")
price = int(input("Enter item's price: "))
pay = int(input("Enter the amount the customer pay: "))
print(name,"price is",price)
print("Customer pay", pay, ",change is", pay - price, "baht.")
```

The same program can be written in C# as follows.

```
using System;
namespace mech
{
  class Program
  {
     public static void Main(string[] args)
     {
       string name;
       int price, pay;
       Console.Write("Enter item's name: ");
       name = Console.ReadLine();
       Console.Write("Enter item's price: ");
       price = int.Parse(Console.ReadLine());
       Console.Write("Enter the amount the customer pay: ");
       pay = int.Parse(Console.ReadLine());
       Console.WriteLine("{0} price is {1}",name, price);
       Console.WriteLine("Customer pay {0},change is {1} baht.",pay,pay-price);
       Console.ReadLine();
     }
  }
}
```

These simple programs illustrates many differences between C# and Python. We shall consider each part of the program.

Lines	Statements	Explanations

8 - 9	`string name;`
	`int price, pay;`
		In C#, every variable must be declared before used. In the declaration, a **type** of the variable must also be specified; the variable then can only store or refer to data of that specific type through out the life of the variable.

10	`Console.Write`
	`("Enter item's name:");`
		Method **`Console.Write`** prints a message to the console.

11	`name =`
	`Console.ReadLine();`
		Method **`Console.ReadLine()`**reads a string from the user. We store the result in variable **`name`**.

12	`Console.Write`
	`("Enter item's price: ");`
		Statement **`Console.Write`** prints a message to the console.

13	`price = int.Parse`
	`  (Console.ReadLine());`
		Statement **`Console.ReadLine()`** returns a string as in function **`input()`** in python; therefore, if we want to read an integer, we will have to convert a string to an integer. In C#, we shall use method **`int.Parse`** in place of function **`int()`** in Python. We store the integer in variable **`price`**.

14 - 15	`Console.Write`
	`("Enter customer pay: ");`
	`pay = int.Parse`
	`(Console.ReadLine());`
		Same as in line 12 – 13.

16	`Console.WriteLine`
	`("{0} price is {1}",name,`
	`price);`
		Again, method **`Console.WriteLine`** prints a message to the console. In this case, we also ask it to print the values of variables **`name`** and **`price`**, which will appear in places of **`{0}`** and **`{1}`** in the template string "`{0} price is {1}`".

17	`Console.WriteLine`
	`("Customer pay {0},change`
	`is {1} baht.",pay`
	`,pay-price);`
		In method **`Console.WriteLine`**, the values that replace {0} or {1} need not be variables. We can use any expressions (e.g., `pay - price`). The program shall evaluate these expressions and pass the results to the method so that they get printed.

Exercise 1.2.1: What is wrong?

Change the program as follows and write down the error messages reported by the C# compiler. (Try each change independently.)

Changes	Error messages and your explanation of the problems
1. Remove ; at the end of a few lines.	
2. Remove `string name;` in line 8.	
3. Edit line 13, by removing **`int.Parse`** so that it becomes **`price = (Console.ReadLine());`**	

Answer these questions.

Questions	Answers
1. What is the output if we change **`{0}`** and **`{1}`** to **`{1}`** and **`{0}`**, respectively, in line 16? (I.e., we swap {0} and {1}.)	
2. What is the output if we swap **`name`** and **`price`** in line 16?	
3. What is the output if, in line 16, we change **`{0}`** to **`{1}`**, and **`{1}`** to **`{2}`**?	

1.3 Number guessing game: Method

In this section, we will develop a number guessing game. You should creating a new console solution. For this program, we will use method **`RandInt`** as shown in Figure 5 that returns random numbers. This method uses variable **`randGen`** which is a global variable. Note that the method and the variable have been declared with keyword **`static`**. We shall study the concepts of variable scopes soon. For now, you can just use the method.

Add the variable and method declarations in class **`Program`** but outside method **`Main`**. Figure 5 shows the code and position to be added in gray.

```
  class Program
  {

    static Random randGen = new Random();
    static int RandInt(int fr, int to)
    {
      return fr + (randGen.Next() % (to - fr + 1));
    }

    public static void Main(string[] args)
    {
      // ………
    }
  }
```

Figure 5 Declarations of `randGen` and method `RandInt`.

We will experiment with the method. Change method **`Main`** to be:

```
    public static void Main(string[] args)
    {
      Console.WriteLine(RandInt(1,100));
      Console.WriteLine(RandInt(1,100));
      Console.WriteLine(RandInt(1,100));
      Console.ReadLine();
    }
```

Run the program to see the output. We will see that the program prints a few numbers to the console and they should be different.

Method **`RandInt`** takes two parameters: **`fr`** and **`to`** and return random numbers between **`fr`** and **`to`**. Note that the method uses operator `%` that, for both C# and Python, returns the remainder of the division.

1.4 Number guessing game: main program

We shall write a simple number guessing game in C#. The program is similar to what we previously wrote in Python. Edit method **`Main`** to be as follows and try the program for a few times.

```
    public static void Main(string[] args)
    {
      int s = RandInt(1,100);
      int g = -1;

      while(g != s) {
        Console.Write("Please guess: ");
        g = int.Parse(Console.ReadLine());
        if(g > s)
          Console.WriteLine("Your guess is too high.");
        if(g < s)
          Console.WriteLine("Your guess is too low.");
      }
      Console.WriteLine("You guessed correctly.");
      Console.ReadLine();
    }
```

Exercise 1.4.1: Answer these questions

1. What is the range of the number to be guessed?	
2. Explain how the **`while`** statement works in this program. What is the condition that make the **`while`** statement to terminate? (*Hint: while statement works the same way as in Python.*)	
3. In the value returned by method `RandInt` is 45, and the user guesses 70, what is the message that the program print?	

Exercise 1.4.2: Number of guesses

We want the program to prints the number of time a user guesses. Fill out the blanks in the following code. Note that a few lines have been added. Relevant lines are shown in gray.

```
    public static void Main(string[] args)
    {
      int s = RandInt(1,100);
      int g = -1;

      int count = 0;

      while(g != s) {

        ___________________________________________________________

        Console.Write("Please guess: ");
        g = int.Parse(Console.ReadLine());
        if(g > s)
          Console.WriteLine("Your guess is too high.");
        if(g < s)
          Console.WriteLine("Your guess is too low.");
      }
      Console.WriteLine("You guessed correctly.");
      Console.WriteLine("You guessed for {0} times.”,_________________);
      Console.ReadLine();
    }
```

1.5 Practice addition

We shall write a game for practicing addition. Examples of two runs of the game are shown below.

```
42 + 72 = ? 114
Good.  You're correct.
56 + 75 = ? 7
Sorry.  The correct answer is 131.
```

Before you start, add method **`RandInt`** with the declaration of **`randGen`** from Figure 5 to your program

Exercise 1.5.1: the question

Complete the following method `Main` that prints a random question. We want the program to random two numbers, each in the range of 1 to 100. A correct program would print a random question and wait for the user to hit *Enter*.

```
    public static void Main(string[] args)
    {

      int x = ____________________________________;

      ___________________________________________;

      Console.Write(”{0} + {1} = ? ”,____, y);
      Console.ReadLine();
    }
```

Exercise 1.5.2: Checking solution

To check the output, we will use **`if`** statement which has the following syntax:

```
      if(  Boolean expression  )
          statement;
```

The major difference from Python is that the **`if`** statement only controls one statement after it. If we want it to control more statements, we will have to group them into one compound statement using {} quotes. We will learn more on this later.

```
      if(  Boolean expression  ) {
          statement1;
          statement2;
              …
          statement3;
      }
```

The condition is expressed as a Boolean expression. Boolean expressions in C# and Python are very similar. We can use operaters “==”, “>”, “<” to compare two values. However, C# does not use “`and`”, “`or`”, “`not`” as operaters.

Indentation does not have any syntactic meaning in C#. However, we should still stick with indentation as in Python because it increases program readability.

Finish the game by adding the code for checking solutions. Fill in the blanks below. Note that the parts that print a random question has be omitted.

```
    public static void Main(string[] args)
    {
      // ------
      // =====  The parts that print a random question has been omitted.   ==========
      // ------

      int ans = _____________________________

      if(_______________________)
        Console.WriteLine("Good.  You're correct.");
      else

        Console.WriteLine(__________________________________________);
    }
```

2. Interacting with users with graphical user interface

2.1 Basic ideas

Unlike a console program, in a program with graphical user interface (GUI) users can interact with the program in a wide range of possibilities. Think of a program with many buttons and text boxes. Therefore, it is very hard to write a purely sequential program to handle this interaction.

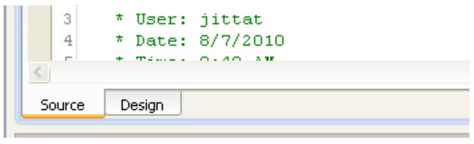
A usual model for writing GUI program is to write an event-driven program, i.e., we write a program for each "event" that we care about. For example, if a program has many buttons, we may write one method to handle the events that the user clicks at each button, separately.

2.2 First GUI program

We will experiment a simple GUI program. The program only has one window with a single button. After clicking the button, the program will say "`Hello, world`"

To create a GUI application, in Sharp Develop, choose "Windows application" as a template when creating a new solution. For this section, put **`winhello`** as the name; if it already exists, choose overwrite when the warning appears.

After that, we will see Sharp Develop environment which looks very similar to what it used to be when we are working on the console applications. However, there is a new tab, "Design" at the bottom side of the editor (as shown in Figure 5). If you do not see that tab, you might select the solution type incorrectly (e.g., you may not choose windows applications), so you should try to create the solution again.

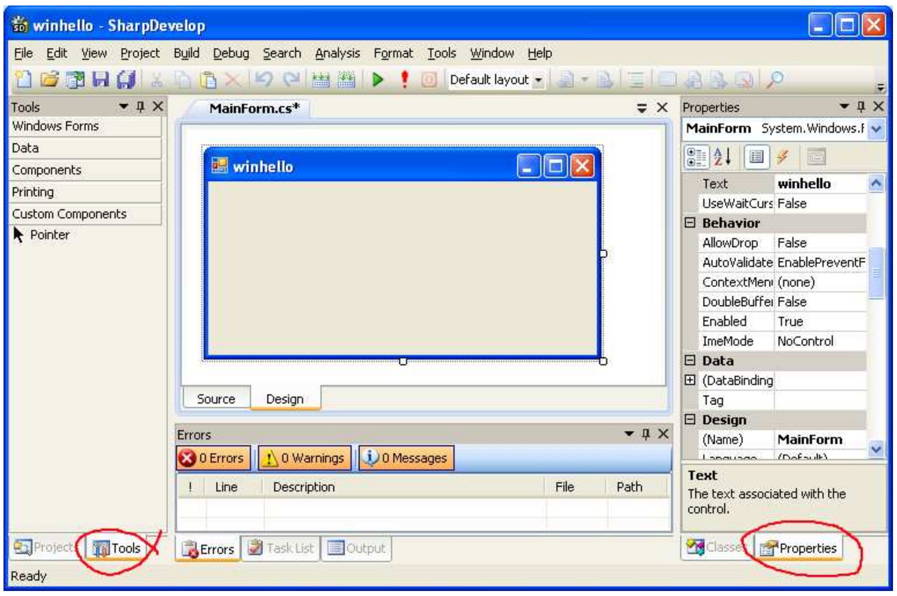


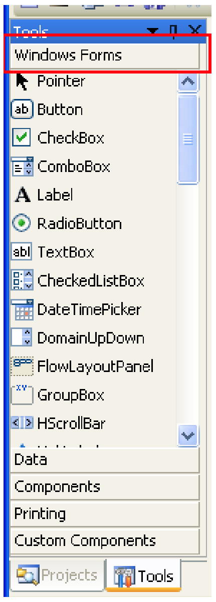
```
 * User: jittat
 * Date: 8/7/2010
```

Figure 5 Design tab

When you click on the "Design" button, you will see a new empty window at the center of the Sharp Develop environment. This window is for designing the user interface. There are many tools on the hidden tabs in the windows, so let's bring them up. Choose tab **Tools** on the left panel and tab **Properties** on the right panel. You can see their positions in Figure 6(a).

We will work with the window at the center of the screen. To avoid confusion, we will call it the **`winhello`** window.





(a) (b)

Figure 6 (a) GUI design window. Red circles on the left and right panels show positions of Tools and Properties tabs. (b) Tab Tools after choosing **Windows Forms** category. Note that there are many kinds of objects we can use.

Exercise 2.2: Exercute the program by click on Run button. What is the output?

Click the X button at the top-right corner of the window we have just created to get back to the editing mode of Sharp Develop.

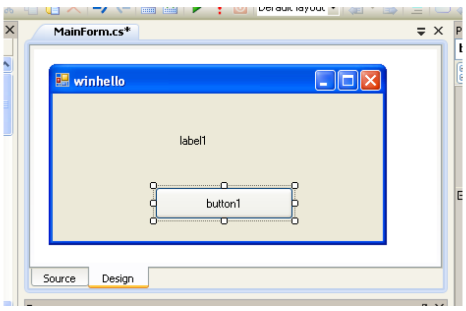
2.2.1 Windows and objects

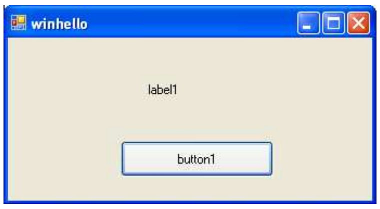
In the Design view, tab **Tools** on the left panel is where we pick user interface objects including buttons, text boxes, or labels, to be placed in our window.

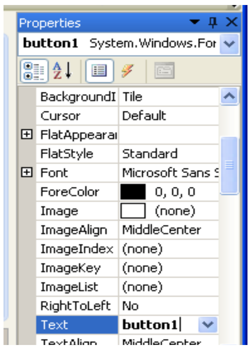
Right now, we will see that objects are grouped into many categories. Choose "**Windows Forms**," and notice that there many basic user interface objects as shown in Figure 6(b). We can click on these objects and place them in our window.

We will start by designing a very simple window. First, click on the **Button** object from the Tools tab and click again at the **winhello** window. Note that a new button appears on the window; on the button, it says **button1**. Try again with a **Label** object. We will see that another object appears on the window. This time it says **label1**. If you click on these object, you can select them and move them around inside the window to make them look nice. Example of `winhello` window shows in Figure 7 (a).

Then we are (partially) done! We can start the program by clicking the Run button on the toolbar. You will see a window similar to the one shown in Figure 7(b). Before continuing, do not forget to close the running window application; otherwise, you will not be able to edit your program.






Properties

button1 System.Windows.For

BackgroundI	Tile
Cursor	Default
⊞ FlatAppearai	
FlatStyle	Standard
⊞ Font	Microsoft Sans S
ForeColor	■ 0, 0, 0
Image	□ (none)
ImageAlign	MiddleCenter
ImageIndex	(none)
ImageKey	(none)
ImageList	(none)
RightToLeft	No
Text	button1
TextAlign	MiddleCenter

(a) (b) (c)

Figure 7 (a) Widow `winhello` in Sharp Develop, note that in this figure `button1` is selected.

(b) Example of the `winhello` application. (c) Properties of `button1`.

2.2.2 Properties

Objects on the window have many properties that we can change, e.g., their colors, texts appearing on them, their positions. We can modify how objects in `winhello` window look by changing their properties. When we select an object, a list of its properties appears in the **Properties** tab.

Click on `button1`. Its properties tab is shown in Figure 7(c). Try to change the `Text` property into **`Hello`**. You will see that the text on the button changes to **`Hello`** as well. We can change how our window looks mainly by setting appropriate properties. Try to experiment with these to get a bigger colorful button. Be careful not to change property "**Name**".

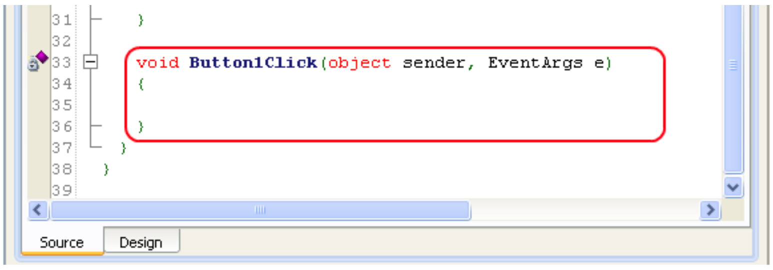
Exercise 2.2.2: Run the program, what do you see?

Note that the running window changes as we designed. However, nothing happens when we click at the button, since we have not write any program to handle the event. We shall write that in the next section. Do not forget to close the application before we continue.

2.2.3 The Hello program

We will add a program that handle to the event that the button is clicked.   We start by double-clicking at `button1` in the design tab.   After clicking on that, the Sharp Develop window would show us the program editor. Note that the tab at the bottom of the editor panel is now “Source.”

When we double-click at a button, Sharp Develop creates a new method for us called **`Button1Click`** so that we can write a program that respond to the button standard event, i.e., with it is clicked.   Its skeleton is shown is Figure 8.



```
        }

        void Button1Click(object sender, EventArgs e)
        {

        }
    }
}
```

Figure 8 Skeleton of method Button1Click that Sharp Develop writes for us.

After **`button1`** is clicked, we will change the text on **`label1`** to be “Helo, world.”   We write the method `Button1Click` as follows.

```
       void Button1Click(object sender, EventArgs e)
       {
         label1.Text = "Hello, world";
       }
```

Run the program and try clicking on the button.   Do you see the right behavior?

Exercise 2.2.3: If we want the program to say “Good-bye,” which part of the program that have to change? How?

2.3 Button clicking game

We will write a simple game. The game shows a button that moves away after we click it. The goal is to click the button as many times as possible.

2.3.1 Object positions

Every object on the window has properties related to its size and position, which are `Top`, `Left`, `Width`, and `Height` . The figure below illustrates the meaning.

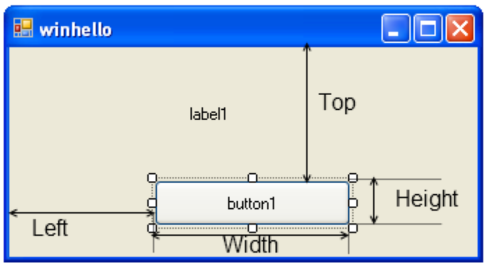


Figure 9 Properties related to object's size and position.

Change method `Button1Click` to be:

```
        void Button1Click(object sender, EventArgs e)
        {
          button1.Left += 10;
        }
```

Exercise 2.3.1: Try the program and answer the following questions before you continue

1. What does the program do when the user click at the button?	
2. If we want the button to move up, what should we do?	

2.3.2 Moving button

Copy mehod **`RandInt`** and the declaration of **`randGen`** from Figure 5 to class **`MainForm`**, before the declaration of method **`Button1Click`**. Modify method **`Button1Click`** to randomly change **`button1`** position in both x-axis and y-axis. The value of **`Top`** and **`Left`** should be between 0 to 100. (*Hint: you need method `RandInt`.*)

Exercise 2.3.2: Write method Button1Click here

```
        void Button1Click(object sender, EventArgs e)
        {




        }
```

2.3.3 A counter

We would like to count how many times a user click on the button. First, declare a global variable **`clickCount`** that keeps the number of times a button is clicked. You should put the declaration before method **`Button1Click`** but it must be inside class **`MainForm`**, as shown in Figure 10. Note that we initialize its value to 0.

```
namespace winhello
{
  public partial class MainForm : Form
  {
    //………
    //………

    static int clickCount = 0;

    void Button1Click(object sender, EventArgs e)
    {
      // ………
    }
  }
}
```

Figure 10 Declaration of `clickCount`

Add the following line at the end of method Buton1Clck. This statement change the text at the button.

```
      button1.Text = clickCount.ToString() + " click(s)";
```

Run the program, we will see a message "`0 click(s)`" when we click a the button.

Exercise 2.3.3: Answer the following questions

1. Remove "`.ToString()`" from the statement above then run the program. What is the error message? Why do we get that error message?	
2. Currently our counter does not work. If we want the number to increase every time a button is clicked, how should we modify **`Button1Click`**?	