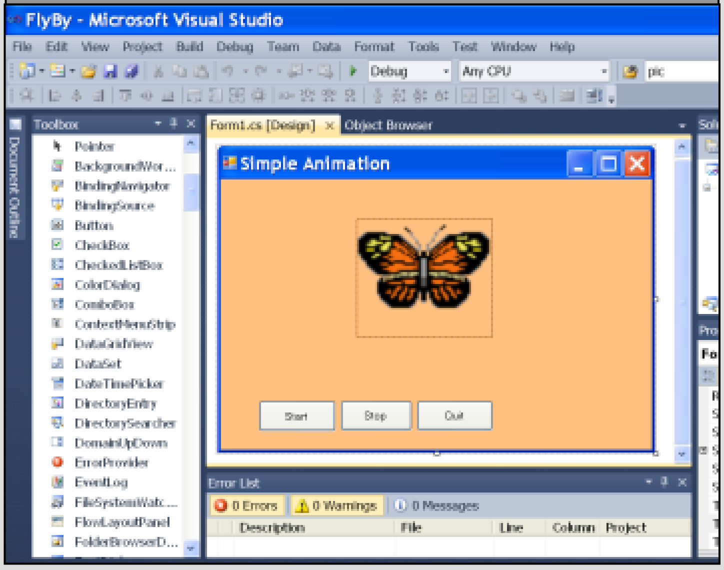
Windows Programming

bucks new university

Console & Windows Programming Using C#



Application Programming (CO453)

Part A Weeks 1-3

Application Programming CO453 C# Console Units 4, 5 & Project

C# .NET Windows & Project Directed Study

Unit 4: Methods with Parameters

Classwork (2 Tasks)

4.1 Converter

Look at project **Task5_1.csproj**. Compile and run it. You see that it asks you to enter a number (**miles**) and then converts this to **feet**.

- Look at the code (see next page) and notice that the program has one class called **Converter**.
- The Converter class has a method called **milesToFeet**. It has one parameter (**miles**) and returns a result (**feet**) so it does the following:

$$\begin{tikzcd}\textbf{miles} \arrow[r] & \boxed{\textbf{milesToFeet}} \arrow[r] & \textbf{feet}\end{tikzcd}$$

Note: there are 1760 yards in a mile and 3 feet in a yard.

- Converter has another method called **getNumber** that also has a parameter (**prompt**). The getNumber method is very flexible because it can use any string prompt to ask for any number and return the result (**num**) .. so it does the following:

$$\begin{tikzcd}\textbf{prompt} \arrow[r] & \boxed{\textbf{getNumber}} \arrow[r] & \textbf{num}\end{tikzcd}$$

- Notice how the **test()** method is used. It has the following algorithm:
  1. use the getNumber() method to get the enter a number of miles
  2. use the milesToFeet() method to convert this to feet
  3. Print out both numbers in a suitable sentence

Task 4.1

1. Add another method called **feetToMiles()** and modify the **test()** method so it does another conversion from feet into miles.
2. Note that if you feed in the result that came from **milesToFeet()** you should get the number you started with if everything is working correctly!
3. Modify the **test()** method so that it displays a small menu of choices (as shown here) and the user can make a selection from the menu. The menu of choices is repeated using a loop until the user chooses the quit option 3.

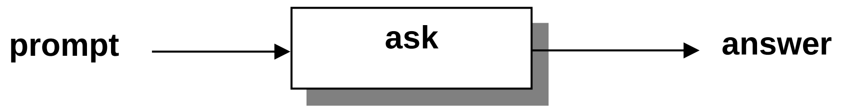
```
Converter Test
============
1. Convert miles to feet
2. Convert feet to miles
3. Quit test
```

```
class Converter
{

    private double  numMiles, numFeet;

    public static void Main()              // program starts executing here
    {
        Converter myConverter = new Converter ();        // create a new object
        myConverter.test();                         // call the object's  test method
    }

    public void test()
    {
         numMiles = getNumber("miles");    //  input number of miles
         numFeet = milesToFeet(numMiles); // use the milesToFeet method
         Console.WriteLine (numMiles + " miles is " + numFeet + " feet");
    }

    public double getNumber(string prompt)
    {
          double num;                                       // local variables for input
          string input;
          Console.Write("Please enter the number of " + prompt + " : ");
          input = Console.ReadLine();
          num = Convert.ToDouble(input);    // convert to a double
          return num;                                 // return value back
    }

    public double milesToFeet(double miles)
    {
          double feet;
          feet = 1760 * 3 * miles;
          return feet;
    }

} // end of Converter class
```

4.2 Horror Story

Look at project **Task5_2.csproj**. Compile and run it. You see that you are asked to enter 6 pieces of information ... these are then used in chapter 1 of a Horror story.

- Look at the code (see next page) and notice that the program has one class called **Book**.
- The Book class has a method called **ask**. It has one parameter (**prompt**) and uses this string to ask for a piece of information. This (**answer**) is then returned so it can be used later.
  So the **ask()** method does the following:



- Also note that the Book class has methods called **getDetails()** and **writeChapter1()**
  - **getDetails()** asks for all the information for the chapter.
  - **writeChapter1()** uses this information in a spooky story!

Task 4.2

1. Add another method called **writeChapter2()** for this riveting story. You must use some of the information already gathered but you must also use getDetails() to pick up 3 extra pieces of information .. including the **gender** of the person in the story and use these in your new chapter.
2. You will have noticed that chapter 1 (and perhaps chapter 2) uses the word 'he' .. but what if the person in the story is female? If you have completed part 1 above, you have entered the gender of the person .. you need to change 'he' to 'she' .. but only if the person is female!! Here are some hints ..
   - Define a new string variable called **gender**. Use this in your story chapters wherever he or she appears.
   - Create a new method called **checkGender()**. This should be used to set the value of **gender** to either "he" or "she" depending on the gender that was entered.
   - Now you should be able to get your story to print correctly .. test it with both genders and see that it works.

Independent Studies (2 Tasks)

The following exercises are to be done individually and independently, in your own time.

ABOUT YOUR LOGBOOK
For this independent study enter in your logbook:

- **Input-Output Diagram**
- **Class Diagram**
- **Method algorithms**
- **Source Code**
- **Test Plan (with results)**

4.3 Constructors

- A **constructor** is a special method which has the **same name** as the class it is in. The constructor is called automatically when a **new** object is created.
- The constructor can be used to initialise a new object in various ways.
- In this example the constructor for the **Book** class would look like this:

```
public Book()
{
}
```

Task 4.3

- Create a **Constructor** method for the Book class that will set default values for all the variables in the story.
- In your **Main()** method you should then allow the user to choose either to enter values or to use the defaults

```
class Book
{

    private string author;
    private string name, town, animal, weapon, job;

    public static void Main()                    // program starts executing here
    {
            Book myBook = new Book();            // create a new Book object
            myBook.getDetails();                 // call its getDetails() method
            myBook.writeChapter1();              // call its writeChapter1() method
    }

    public string ask(string prompt)             // use prompt to ask for information
    {
            string answer;
            Console.Write(prompt);
            answer = Console.ReadLine();
            return answer;
    }

    public void getDetails()              // keep using ask() to get information
    {
            author = ask("Please type your name : ");
            name = ask("Please type a friend's name : ");
            town = ask("Now give me the name of a town : ");
            animal = ask("Now a type of animal : ");
            job = ask("Now a type of job : ");
            weapon = ask("And your weapon of choice : ");
    }

    public void writeChapter1()           // write chapter using information gathered
    {
            Console.Clear();
            Console.WriteLine("A Horror Story : by " + author);
            Console.WriteLine("===============================");
            Console.WriteLine("It was a dark night in " + town + " and " + name
                    + " could hear " + animal + "s screaming in the distance.");
            Console.WriteLine(name + " staggered terrified through the
                    streets of " + town + ", realising he had been followed.");
            Console.WriteLine("In the shadow of a doorway, a demented " + job
                  + " waited, clutching a menacing " + weapon
                    + " in his hand.");
    }
} // end of Book class
```

4.4 Body Mass Index

1. Create a new project with a class called **BMI**.
2. Set up appropriate variables (read below first)
3. Add a new method called **getDetails()** which will allow the user to enter their **weight** (in kgs) and **height** (in metres)
4. Another method called **calcBMI()** returns a value for the BMI using this formula :
   BMI = weight in kg/(height in metres)^2
5. Another method displays the BMI value and a message depending on the value of the BMI

BMI	*less than 18.5*	*… Underweight*
BMI	*18.5 up to 25*	*… Desirable weight for size*
BMI	*25 up to 30*	*… Overweight*
BMI	*30 up to 40*	*… Obese*
BMI	*40 or over*	*… Severely Obese*

6. Your **Main()** method should create a new BMI object and then call the appropriate methods to enter the user's weight and height and then calculate and display the BMI value and message

Extra:

- Modify the program so that it can cope with either metric measurements (kg and metres) or imperial (pounds and inches)

Calculating BMI (Body Mass Index)
- BMI = weight in kg/(height in metres)^2
- BMI = weight in pounds x 703/(height in inches)^2

Unit 5: Arrays

Classwork (3 Tasks)

5.1 Tournament Scores

Look at project **Task6_1.csproj**. Compile and run it.

- Look at the code below
- Notice that the program has one class called **Tournament**.
- It defines an integer **array** called **scores** to hold all the scores in the tournament.
- The constructor for the class then actually creates the array of the required size.

```
class Tournament
{

    int[ ] scores;               // define scores as an integer array
    const int MAX = 6;           // set a constant size

    public static void Main()            // program starts executing here
    {
            Tournament  myTournament = new Tournament();  // create a new object
             myTournament.getScores();                  // call its getScores method
    }

    public Tournament()                          // the class constructor
    {
             scores = new int[MAX];              // create a new array of size MAX
    }

    public void getScores()
    {
             Console.WriteLine("Inputting the Tournament Scores");
             Console.WriteLine("=======================");
             for (int i = 0; i < MAX; i++)
             {
                Console.Write("Enter score number " + (i + 1) + " : ");
                scores[i] = Convert.ToInt32(Console.ReadLine());
             }
    }

} // end of Tournament class
```

Task 5.1

1. Add another method to the class, called **showScores()** .. this should clear the screen and then display all the scores in the form:
   Tournament Scores
   ================
   Player 1 scored < >
   Player 2 scored < > etc.

2. Change the size of the tournament to **12**. Check that it still works OK.
3. Put source code and sample outputs in your logbook

5.2 MP3 Chart Voter

Look at project **Task6_2.csproj**. Compile and run it.
The program presents you with a list of song tracks and you have to vote for your favourite.
Of course it is far from finished!

- Examine the existing code on the next page
- modify the program to complete these tasks:

Task 5.2

The program has to keep a count of all the votes for each song track.

1. Create a new integer array called **votes** that will be used to count the votes for each track
2. Now inside the **getVotes()** method you need to <u>add one</u> to the appropriate vote in the **votes** array. There are several ways of doing this. e.g. if the vote was for track 5 then add 1 to **votes[4]** (remember arrays start counting from 0)
3. Get the program to repeat for many voters by using a <u>loop</u> inside the **run()** method
4. Add another method called **showVotes()** which displays all the vote counts like this:
   MP3 Track Votes
   ==============
   Track 1 had < > votes
   Track 2 had < > votes etc.

   Total Number of Votes : < >
5. Expand the program to work for <u>10 tracks</u> and add your own favourites to the list.
   Check that everything works OK.
6. Use **MAX** inside the **getVotes()** method so that it always asks you to vote correctly depending on the number of tracks e.g. Choose 1 – 10 etc.

Extra

- Can you get **showVotes()** to display the track titles as well as the votes?

Put source code and sample outputs in your logbook

```
class Mp3Chart
{
        string[ ]  topTen;              // define a string array called topTen
        const int MAX = 5;

        public static void Main()              // program starts executing here
        {
                Mp3Chart  myChart = new Mp3Chart();            // create new object
                myChart.run();                                 // call its run method
        }

        public Mp3Chart()             // constructor
        {
                    topTen = new string[MAX];     // create a new array of correct size
                    topTen[0] = "Revolution";  // initialise the array values
                    topTen[1] = "Mera Dil Tuta Hain";
                    topTen[2] = "CandyMan";
                    topTen[3] = "Ruby Tuesday";
                    topTen[4] = "Old Man";
        }

        public void run()
        {
                    showMusicList();
                    getVotes();
        }

        public void showMusicList()
        {
                    Console.Clear();
                    Console.WriteLine("\tMusic List");
                    Console.WriteLine("\t==========");
                    for (int i=0; i < MAX; i++)
                    {
                         Console.WriteLine("\tSong " + (i + 1) + " is " + topTen[i]);
                    }
        }

        public void getVotes()
        {
                    int  userVote;
                    Console.WriteLine("\tSelect your favourite Song");
                    Console.WriteLine("\t==========================");
                    Console.Write("\tChoose 1 - 5 : ");
                    userVote = Convert.ToInt32(Console.ReadLine());
        }
} // end of Mp3Chart class
```

5.3 Tournament Names

Modify the Tournament class of 5.1 as follows to deal with names as well as scores

1. Add a string array called **names** to the class.
2. Change the name of the **getScores()** method to **getDetails()** and use it to input all the names as well as the scores, like this:

```
Input Names and Scores
====================
Enter player 1 name :   <     >
Enter player  <       > score :   <     >
Enter player 2 name :   <     >
Enter player  <       > score :   <     >    etc.
```

3. Add a new method called **showDetails()** and use it to display all the names and scores, like this:

```
Tournament Results
=================
Player  <    >  scored  <     >
Player  <    >  scored  <     >   etc.
```

4. Add a new method called **showBest()** .. it should look through the scores to find the highest score and then print out the **name** and **score** for this person.

Put source code and sample outputs in your logbook

Independent Study (3 Tasks)

The following exercises are to be done individually and independently, in your own time.

5.4 Sorting

Computers spend a lot of time sorting things into order and there are many different sorting algorithms to choose from. One of the simplest (and slowest) is called the **Bubble Sort**. It has one loop contained inside another loop as shown here.

Bubble Sort for N items

```
loop N times
      loop from 0 up to N-1
            if current item > next item
                  swap the two items
            end if
      end loop
end loop
```

Your Tasks

1. Create a new project for this task with a class called **Bubble**
2. Add a new method called **inputNumbers()** which inputs 6 numbers into an array
3. Add a second method called **display()** which clears the screen and displays all the numbers one above the other.
4. Make sure the program works correctly so far
5. Now add a third method called **sortNumbers()** which applies the Bubble Sort algorithm to sort the numbers into numerical order
6. Call the methods in the right order and get the sorting to work.
7. Try it for 20 numbers
8. Add 3 more methods to apply a similar technique to sorting a list of names into alphabetical order.

5.5 Traffic Survey

It has been decided to do a **traffic survey** at a particularly busy section of road.
Traffic is counted automatically during **24** 1-hour time periods in a typical day and the counts are then stored in an array in the program for later analysis.
You are to simulate this using an **array** for the 24 periods.

- Create a new project with a class called **Traffic**.
- Set up an integer array called **trafficCount** with 24 elements.
- Define a method called **enterCounts()** which allows the user to enter 24 counts into the array.
- Another method called **showTotal()** should calculate and display the total number of cars in the array.
- A third method called **busiest()** should work out and display the busiest time of day.
- A forth method called **showData()** should output **all** the data in a suitable table with the percentage of the total.
- Provide a **report()** method that does the following:
    - calls **enterCounts()**
    - calls **showData()**
    - calls **showTotal()**
    - calls **busiest()**

Your results should look something like the following:

```
Traffic Report
------------------
Hour          Car Count           Percentage of Total
-----------------------------------------------------------------
1             1200                  7.7%
2             1155                  6.4%
etc.

Total Car Count for the day = 15546
Busiest hour = 7
-----------------------------------------------------------------
```

Put source code and sample outputs in your logbook

5.6 The Bates Motel

Look at project **Task7_2.csproj**. Compile and run it. This simulates an incomplete booking system for the **Bates Motel**. Your task is improve the functionality of the program.

This is a **menu-based** program for booking and vacating rooms.

There are 5 options available on the repeating menu:

1. Book a room
2. Vacate a room
3. Display ALL Room Details
4. Vacate ALL rooms
5. Quit

But only item 1 (Book a Room) is currently implemented

NOTES

- The motel has 20 rooms.
- an **integer array** called **rooms** is used to store the number of guests in each room.
- Notice the size of this array has been set to **MAX+1** so we can use room numbers 1 to 20

Task 5.6

1. Start by implementing item **3** of the menu. Do this with a method called **showAllRooms()** This should display all the room details as follows:
   ```
   Bates Motel Room Status
   ====================
   Room 1      0 guests
   Room 2      2 guests      etc.
   ```
2. Implement menu item **2** with a method called **vacateOneRoom()** which asks you which room you want to vacate and then puts 0 into this position of the array
3. Now implement menu item **4** which should allow you to vacate all the rooms. Use a method called **vacateAll()** which should put 0 into every position in the rooms array.
4. Check that all is working correctly.

Put source code and sample outputs in your logbook

Extra: The Bates Motel (contd)

1. Notice that you can currently double-book a room in the motel. If you choose a room that is already booked, the old booking is overwritten! Add some code to prevent this from happening in a user-friendly way.
2. Add a new Item **5** : **Management Information** on the menu. This should use a suitable method and provide useful information such as how many rooms are booked, how many guests are in the hotel and also the room numbers of all the empty rooms.
3. It is possible to book any number of guests into a room but the room limit is 4. Provide a user-friendly mechanism which only allows up to 4 guests per room.

```
class Motel
{
        int[ ]  rooms;                  // define an integer array called rooms
        const int MAX = 21:

        public static void Main()              // program starts executing here
        {
                Motel  BatesMotel = new Motel();  // create new object BatesMotel
                BatesMotel.runMotel();                 // call its runMotel method
        }

        public Motel()              // constructor
        {
                  rooms = new int[MAX];       // allow room numbers from 1 to 20
        }

        public void runMotel()
        {
                string choice = "";
                do
                {
                   Console.Clear();
                   Console.WriteLine("The Bates Motel");
                   Console.WriteLine("===============");
                   Console.WriteLine("1. Book a room");
                   Console.WriteLine("2. Vacate a room");
                   Console.WriteLine("3. Display ALL Room Details");
                   Console.WriteLine("4. Vacate ALL rooms");
                   Console.WriteLine("5. Quit");
                   Console.Write("Enter your choice : ");
                   choice = Console.ReadLine();
                   if (choice == "1")
                   {
                      bookRoom();
                   }
                }
                while (choice != "5");     // repeat until 5 is chosen from the menu
        }

        public void bookRoom()
        {
              int  roomNumber, guests;
              Console.WriteLine("\nThe Bates Motel");
              Console.WriteLine("===============");
              Console.WriteLine("Book a room");
              Console.Write("Enter the room number : ");
              roomNumber = Convert.ToInt32(Console.ReadLine());
              Console.Write("How many guests : ");
              guests = Convert.ToInt32(Console.ReadLine());
              rooms[roomNumber] = guests;     // make the booking
              Console.WriteLine("Room " + roomNumber + " booked for " + guests + " people");
              Console.ReadKey();      // wait for key press
        }
} // end of Motel class
```

Project Unit: C# Project: The Scissors-Paper-Stone Game

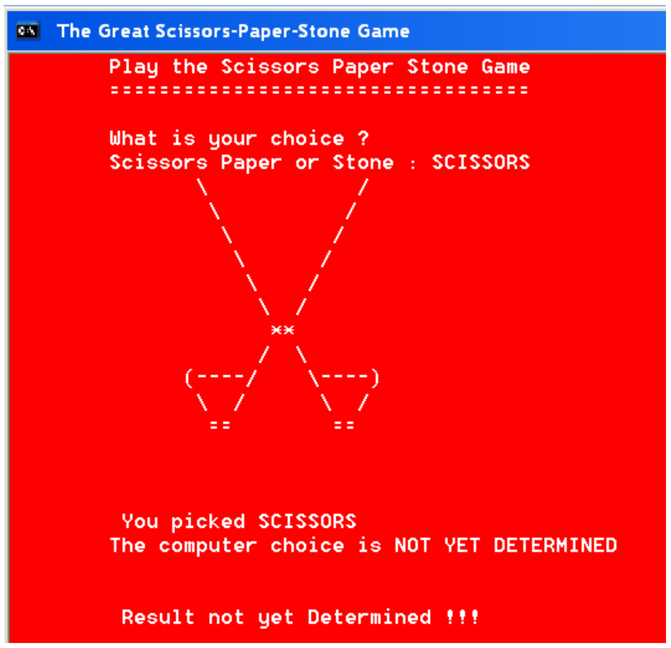
This week we shall begin a slightly larger exercise .. the Scissors-Paper-Stone Game. You should know the rules of the game before you start:

The Basic Rules (playing against the computer)
======================================
- The player chooses either: Scissors, Paper or Stone
- The computer also chooses one of these at random
- There are various possible results:
  - If player and computer choose the same thing, the result is a Draw.
  - Scissors win against Paper (because scissors cut paper)
  - Scissors lose against Stone (because stone blunts scissors)
  - Paper wins against Stone (because paper wraps round stone)

<u>Starting the Project</u>

- This project has already been started for you but it is <u>incomplete</u> and needs a lot of work to finish it.
- Start by opening the **SPSProject** and run it .. set the keyboard Caps Lock ON and when you are asked for your choice, type: **SCISSORS**.
- The computer will now make its random choice and you should then see a crude picture of your choice and a result that is either:
  - a <u>DRAW</u> or
  - <u>NOT YET DETERMINED</u>

  (depends what the computer picked)
- Run the program again and try choosing **PAPER** or **STONE**
- Examine the existing code for the program (see later pages)
- Clearly the program is nowhere near finished so you should try adding more code to achieve the following tasks:



```
The Great Scissors-Paper-Stone Game
Play the Scissors Paper Stone Game
==================================

What is your choice ?
Scissors Paper or Stone : SCISSORS
       \            /
        \          /
         \        /
          \      /
           \    /
            \  /
             **
            /  \
      (----/    \----)
       \  /      \  /
        ==        ==



 You picked SCISSORS
The computer choice is NOT YET DETERMINED


 Result not yet Determined !!!
```

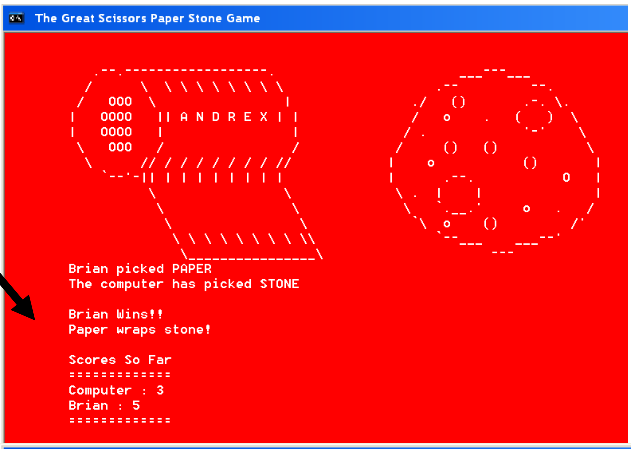
Basic Project

- Change the background and foreground colours to your own choice.
- Modify the program so that all the computer choices are described correctly, instead of "NOT YET DETERMINED" as above (e.g. The computer chose STONE)
- Get the program to show the result correctly for all possible situations (e.g. THE COMPUTER WINS or YOU WIN) .. instead of "not yet determined" as above.
- Get the program to draw the computer choice as well as the player choice.
- Add a variable for the player name and add code to pick up the name at the start of the program.
- The player name should be used wherever possible e.g

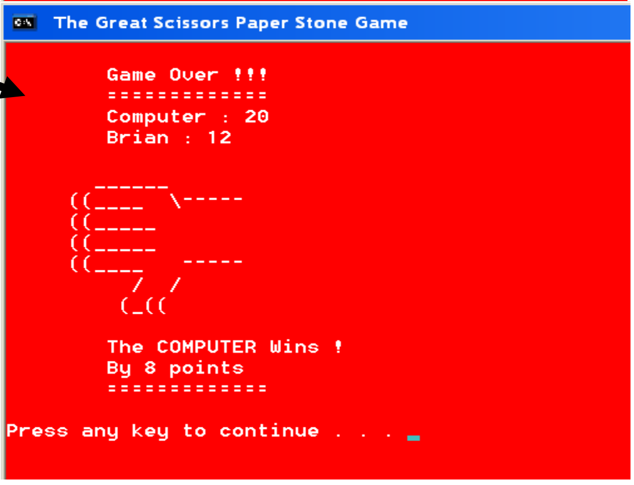
  What is your choice, *Brian*?
  Brian picked SCISSORS. The computer picked PAPER
  Brian WON!! Because Scissors Cut Paper.
- Get the program to work for both uppercase and lowercase inputs e.g. it should work if you choose SCISSORS or scissors or Scissors, etc.

Extension Work 1

- You are to use a scoring system in the game so you must add 2 variables for the ComputerScore and the PlayerScore.
- Implement the scoring as follows:
  - 2 points for a WIN
  - 1 point each for a DRAW
- Add a new method called **showScores()** which prints the scores for the player and computer as shown here:
- Now get the game play to repeat until one of the scores reaches 20.
- Create a new method called **finish()** which is called when the game loop ends.
- The finish() method should clear the screen and then print the results as shown here.
- Use an appropriate picture:
  - ThumbsUp (player win)
  - ThumbsDown (computer win)
  - Smile (draw)
- Note: you will find included some appropriate draw methods for you to use.



```
Brian picked PAPER
The computer has picked STONE

Brian Wins!!
Paper wraps stone!

Scores So Far
=============
Computer : 3
Brian : 5
=============
```



```
Game Over !!!
=============
Computer : 20
Brian : 12

The COMPUTER Wins !
By 8 points
=============

Press any key to continue . . .
```

Extension Work 2

- Make a copy of your complete Game folder so you don't lose your original game.
- Create a new **Class** in your project and call this **Pictures**
- Remove all the draw methods from your Game class and put them into your Pictures class.
- Get the game to work again using pictures from the Pictures class .. you will have to make several changes e.g. the draw methods should be public instead of private and you will need to create a Pictures object within the Game class.
- Most of the pictures can be positioned anywhere on the screen, but some of them can't .. can you modify these methods so they too can be drawn anywhere?

```
class  Game
{
        string compChoice;
        string playerChoice;
        Random randy;

        public static void Main()               // program starts executing here
        {
              Game myGame = new Game ();                  // create a new Game object
              myGame.play();                          // call its play method
        }

        public Game()                                 // game constructor
        {
               randy = new Random();                  //  create a new Random object
        }

        public void play()                      // play the game (unfinished)
        {
                  setupScreen();
                  introduction();
                  getPlayerChoice();
                  getComputerChoice();
                  drawPlayerChoice();
                  printChoices();
                  showResult();
                  Console.ReadKey();                  // wait for a key press
        }

        private void setupScreen()
        {
                  Console.Title = " The Great Scissors-Paper-Stone Game";
                  Console.SetWindowSize(100, 36);
                  Console.SetBufferSize(100, 36)
                  Console.BackgroundColor = ConsoleColor.Red;
                  Console.ForegroundColor = ConsoleColor.White;
                  Console.Clear();      // clear screen in chosen colour
        }

        private void introduction()
        {
                  Console.WriteLine("\tPlay the Scissors Paper Stone Game");
                  Console.WriteLine("\t==========================");
        }

        private void getPlayerChoice()
        {
                 Console.WriteLine("\n\tWhat is your choice ?");
                 Console.Write("\tScissors Paper or Stone : ");
                 playerChoice = Console.ReadLine();
        }

        // PTO
```

// Game class continued

```
private void getComputerChoice()    // unfinished
{
            int num;
            num = randy.Next(3);      // pick a random number (0, 1 or 2)
            if (num == 0)
            {
               compChoice = "SCISSORS";
            }
            else
            {
               compChoice = "NOT YET DETERMINED";
            }
}
```

```
private void drawPlayerChoice()
{
            if (playerChoice == "SCISSORS")
            {
                     drawScissors(10, 5);            // draw Scissors at 10, 5
            }
            else if (playerChoice == "PAPER")
            {
                     drawPaper(10, 5);
            }
            else if (playerChoice == "STONE")
            {
                     drawStone(10, 5);
            }
}
```

```
private void printChoices()
{
            Console.WriteLine("\n\t You picked " + playerChoice);
            Console.WriteLine("\tThe computer has picked " + compChoice);
}
```

```
private void showResult()
{
            if (playerChoice == compChoice)
            {
               Console.WriteLine("\n\tA DRAW!!");
            }
            else
            {
               Console.WriteLine("\n\n\t Result not yet Determined !!!");
            }
}
```

```
private void drawScissors(int x, int y)            // draw at x, y
{
            Console.SetCursorPosition(x, y++);  // set start position then add 1 to y
            Console.Write("     \\           /");         // etc.  for rest of
```

SPS Game Project Deliverables
==========================
Include the following in your logbook:

- Fully Commented Source Code
- Sample Screen shots
- Completed Test Plan
- Class Diagram(s)
- Commentary on success (or otherwise)

Some Extra Useful C# Stuff

Console Screen Instructions

```
Console.BackgroundColor = ConsoleColor.Blue;        // set a background colour
Console.ForegroundColor = ConsoleColor.Yellow; // set a foreground colour
Console.Clear () ;                                  // clear the screen
Console.SetCursorPosition(5, 10);               // x,y position on screen
```

Pausing the Program

```
Console.ReadKey () ; //  waits for a key to be pressed
```

Delay for a time

```
System.Threading.Thread.Sleep (1000);
 // gives a delay of 1 second (1000 milliseconds)
```

The Math.Pow() function

Use the Math.Pow() function to return the power of a number: e.g.

```
cube = Math.Pow(number, 3);
square = Math.Pow(number, 2);
```

Converting a string into upper case

```
choice = choice.ToUpper() ;
```

converts the string choice into upper case (e.g. yes becomes YES)

Random Number Generation

- First we must create a new object from the Random class .. e.g.
  Random rand = new Random();   *// creates a new object called **rand***
- Then you can use **rand.Next()** to pick the next number e.g.
  *// pick a random number between 0 and 5 and store in an int variable **n***
  // add 1 to get a number between 1 and 6
  n = rand.Next(6) + 1; *// puts either 1,2,3,4,5 or 6 into n*
  (or use **n = rand.Next() % 6 + 1**)

Output of Decimal Places

- If you want to print a <u>double</u> type of variable (num) to 2 decimal places:
  Console.WriteLine("The answer is " + num.ToString("0.00"));

Alternative Way of Printing Variables

- Instead of :
  Console.WriteLine("The total of " + n1 + " and " + n2 + " is " + total);
  You could write:
  Console.WriteLine("The total of {0} and {1} is {2}",   n1, n2, total);
  // n1, n2 and total are put in positions {0} {1} and {2} respectively
- ***or** an alternative if you want 2 decimal places:*
  Console.WriteLine("The total of {0:F2} and {1:F2} is {2:F2}",   n1, n2, total);
  // this formats all the numbers to Fixed 2 decimal places

Inputting Numbers

- First input into a string variable e.g.
  input = Console.ReadLine();
- Then convert this string to the right type and pop into a variable e.g.
  num = Convert.ToDouble(input);   or
  num = Convert.ToInt32(input);   etc.

Appendix A: The Basics

1. Console Input and Output

name = Console.ReadLine(); .. store input in a name variable (defined as string)
Console.WriteLine("I am " + name); .. output a message with text joined to a name variable
num1 = Convert.ToDouble (Console.ReadLine()); .. enter string and convert to a double
num2 = Convert.ToInt32 (Console.ReadLine()); .. enter string and convert to an integer

2. Variables

int count;	.. define a variable called count to store an integer number
double num;	.. define a variable called num to store a double (decimal) number
string name;	.. define a variable called name to store a string (text or words)

3. Assignments to Variables (must be defined first)

count = 0;	.. put 0 into the count variable (previously defined as int)
num = 5.67;	.. put 5.67 into the num variable (previously defined as double)
name = "Fred Bloggs";	.. put these 11 characters in the name variable (defined as string)

4. Calculations

count ++;	.. add 1 to the value of the count variable
count --;	.. subtract 1 from the value of the count variable
count = count + 3;	.. add 3 to value of the count variable (or use **count += 3;**)
count = count - 6;	.. subtract 6 from the count variable (or use **count -= 6;**)
av = (num1 + num2 + num3 + num4) / 4;	.. work our average of 4 numbers
tax = bill * 17.5 / 100;	.. work out 17.5 percent tax on your bill

5. Loops (iteration)

a. The while loop

```
int count = 0;            // initialise a loop counter to zero

while (count < 10)     // continue while loop counter is less than 10
{
        Console.WriteLIne (“The count is " + count);   // repeated
        count ++;       // keep loop going by adding 1 to counter
}
```

an infinite loop

```
while (true)     // continue the while loop forever
{
        Console.WriteLIne ("Yippeeee!!");   // repeated forever
}
```

b. The for loop

```
// initialise loop counter; continue while count less than 10 ;  add 1 at end of loop

for (int count = 0; count < 10; count ++)
{
        Console.WriteLine (“The count is " + count);   // repeated 10 times
}
```

c. The do while loop

```
int  count = 0;           // initialise a loop counter to zero

do
{
        count ++;      // keep loop going by adding 1 to loop counter
        Console.WriteLine (“The count is " + count);   // repeated message
}
while (count < 10);     // continue while loop counter is less than 10
```

6. Selection

a. The if statement

```
        if (count == 4)        // if count is equal to 4
        {
                Console.WriteLine (“We are half way" );
        }
```

b. The if else statement

```
        if (count >= 4)        // if count is greater or equal to 4
        {
                Console.WriteLine (“We have reached half way" );
        }
        else
        {
                Console.WriteLine ("We are NOT half way yet");
        }
```

c. The switch statement

```
        switch(count)        // use count value to switch to various cases below:
        {
                case 1:                                    // i.e. if  count value = 1
                        Console.WriteLine (“We are just starting" );          break;
                case 2:  case 3: case 4:
                        Console.WriteLine (“We are on our way" );             break;
                case 4:
                        Console.WriteLine (“We are half way" );               break;
                default:
                        // do nothing for any other values              break;
        }
```

7. Conditions

(a == b)	.. a is equal to b ?
(a > b)	.. a is greater than b ?
(a < b)	.. a is less than b ?
(a >= b)	.. a is greater or equal to b ?
(a <= b)	.. a is less than or equal to b ?
(a != b)	.. a is NOT equal to b ?

8. Multiple Conditions

```
(a == b  || a == c)            .. a is equal to b OR a is equal to c  ?
(a == b  || a == c || a == d)  .. a is equal to b OR a is equal to c OR a is equal to d ?
(a == b  && a == c)            .. a is equal to b AND a is equal to c  ?
(a <= 100  && a >= 0)          .. a is less than or equal to 100 AND a is greater or equal to 0  ?
```

9. Classes, Objects and Methods

```
class Meal                    // define a class called Meal
{
        private string food;  // the class has one class variable (attribute or field)

         public static void Main()          // program starts executing here
         {
              Meal myMeal = new Meal();          // create a new myMeal object
             myMeal.getFood();                   // call the object's getFood() method
         }

        public Meal()                       // this is the Meal class constructor
        {
                food = "Fish and Chips";    // this sets the default food
        }

        public void getFood()        // define a method getFood()which returns nothing (void)
        {
                Console.WriteLine("What would you like to eat?");
                food = Console.ReadLine();                // input into the class variable food
        }
}
```

*// this defines a simple class **Meal** which has one variable, one method, one constructor*

10. Methods with parameters

```
public double setTax(double  amount)
{
      double  taxAmount;           // local variable
       taxAmount = amount * 17.5/100;
       return  taxAmount;
}
```

- *this defines a method **setTax()** which has 1 parameter (amount) and returns a double value*
- *this method is defined inside a class e.g the Meal class above*
- *to use it, you can 'call' it like this:*

```
vat  =  myMeal.setTax(Bill);      // assume myMeal is the object created from Meal
```

this passes the value of Bill to the method and picks up a returned tax value from it.

11. try/catch *(simple version : to trap errors or exceptions)*

```
try
{
        // enter instructions to be checked here
}
catch
{
        // error message display here
}
```

Assessment of CO453 Application Programming

1. This module is assessed by coursework. There are three parts to this coursework (Part A, B and C). There are study packs for each of the three parts. The study packs contain both class exercises and independent exercises relating to the programming concept being taught that week. There is a project week in Part B that includes a series of related tasks.

2. **Class exercises will be assessed**. Each week contains between four to six class exercises. Your tutor will monitor your progress in these each week. **These class exercises are worth 40% of your Part A mark.**

3. **Independent studies will be assessed.** The code for these tasks will be assessed on their efficiency, syntax, correct use of concept, and whether the code fulfils the requirements of the task. Some tasks may also require additional documentation such as test plans and algorithms. Please include screenshots of your code running and comments where relevant**. You must complete these independent exercises on your own outside of the session. These exercises are worth 60% of your Part A mark.**

4. Create a logbook (for example: an MS Word document) to document your code. The logbook should contain your designs, algorithms, test plans, source code and results of your work. **This must be submitted electronically through the designated TurnItIn submission point** (your tutor will show you). **If there is a technical problem and you cannot submit through TurnItIn, please speak to someone from the administration office (E4.08).**

5. Your mark for this module will be based on your grades for each of the parts (A, B, C). Below shows the weighting for each part of the coursework:

Part A: 30% of module mark

Week 1 and 2 class exercises	= 30% of Part A mark
Week 1 and 2 independent exercises	= 40% of Part A mark
Week 3 Project (all exercises)	= 30% of Part A mark

Part B: 40% of module mark

Week 5 (TBD), Week 6 (TBD), Week 7 (TBD), Week 8 (TBD)

Part C: 30% of module mark

Week 10 (TBD), Week 11 (TBD), Week 12 (TBD)

Grade related criteria for Programming - CO453

A	Where the student has demonstrated clear evidence of an excellent understanding of the theories and principles together with a high degree of analytical accuracy, good design skills, implementing fully tested solutions that show reliability, maintainability, readability and minimal complexity and correct form of presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the lecture and practical sessions and attempt at least 85% of independent study for each week.*
B	Where the student has demonstrated clear evidence of a good understanding of the theories and principles together with a good analytical ability, good design skills, implementing solutions that show reliability, maintainability, readability and minimal complexity and correct form of presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the lecture and practical sessions and attempt at least 75% the independent study for each week.*
C	Where the student has demonstrated a reasonable understanding of the theories and principles together with a reasonable analytical ability, design skills, implementing solutions that appreciate the need for reliability, maintainability, readability and minimal complexity and reasonable presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the lecture and practical sessions and attempt at least 66% of the independent study for each week.*
D	Where the student has demonstrated an understanding of the theories and principles of analysis, design, implementation and presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the lecture and practical sessions and attempt at least 50% of the independent study for each week.*
E	Where the student has made a genuine attempt to acquire the knowledge and skills but requires further application and study to demonstrate an understanding of the theories and principles of analysis, design, implementation and presentation skills. *In order to demonstrate a genuine attempt the student will normally be expected to attend the lecture and practical sessions and attempt at least 40% of the independent study.*
F	Where the student has clearly not acquired sufficient knowledge and skills and not attempted or coped with the directed study with any degree of competence regarding theories, principles, analysis, design, and implementation and presentation skills or where the student has NOT attended for assessment or where the student has copied work from an alternative source.

Module Name and code	**Application Programming CO453**
Staff:	Richard Jones, Richard Mather, Carlo Lusuardi & Nick Day

Learning Outcomes:

- Analyse a simple requirement in a structured manner
- Design, document, implement and test reliable, maintainable programs as solutions to simple problems
- Use structured techniques of design and implementation and good documentation practice.
- Use software development tools.

Teach WK	**Uni WK**	**LECTURE/TUTORIAL**	**PRACTICAL**
1	19	**C# (Console) 4 Methods and Parameters**	C# Directed Study Unit 4
2	20	**C# (Console) 5 Arrays**	C# Directed Study Unit 5
3	21	**C# (Console) The Project**	C# Directed Study Project Unit
4	22	**Workshop week**	
5	23	**Windows C#1 Introduction & Splash Screen**	C# Directed Study Unit 1
6	24	**Windows C#2 SPS Game**	C# Directed Study Unit 2
7	25	**Windows C#3 Other .NET Controls**	C# Directed Study Unit 3
8	26	**Windows C#4 Multiform projects**	C# Directed Study Unit 4
9	27	**Workshop week**	
	28-30	**Spring Recess – (Easter)**	
10	31	**Windows C#5 Animation**	C# Directed Study Unit 5
11	32	**Windows C#6 Graphics**	C# Directed Study Unit 6
12	33	**Windows C#7 The .Net Project**	C# Directed Study Unit 7
13	34	**Workshop week**	

Course Texts:
Comprehensive Course Notes are provided
Bradley & Millspaugh, ***Programming in C#***, 2010, pub: McGraw Hill
Deitel & Deitel, ***Visual C# 2010 How to Program***, 2011, pub: Pearson