Revision Note

Commenting

Comments are ignored when your code compiles or execute.
It mainly documents what your code does.

`// This is an example` - Single Line Comments
`/* This is an example*/` - Span Multiple Lines Comments

Statements

All statements in C# ends with `;`

```
Console.WriteLine;
```

Class & Methods

Both Class & Methods begins with `{` and ends with `}`

All C# console program must contain Main Method.
Main Method is the starting and ending of C# Console Program

```
using System;

namespace Example
{
        class Program
        {
            static void Main(string[] args)
            {
                Console.WriteLine("This is an example")
            }
        }
}
```

Using Statement

The `using` statement helps to simplify code written.

Example without `using` statement :

```
namespace Example
{
        class Program
        {
            static void Main(string[] args)
            {
                System.Console.WriteLine("Hello World!")
            }
        }
}
```

Example with `using` statement :

```
using System;

namespace Example
{
        class Program
        {
            static void Main(string[] args)
            {
                Console.WriteLine("Hello World!")
            }
        }
}
```

Namespace

The `namespace` keyword is used to declare namespace and help to control the cope of class and method names in larger programming projects

Example :

```
using System;

namespace SimpleHelloWorld
```

```
{
        class Program
        {
            static void Main(string[] args)
            {
                Console.WriteLine("Hello World!")
            }
        }
}
```

Console Input & Output Method

`Write()` - Write in the same line
`WriteLine()` - Write in a new line

`ReadLine()` - Read a line (string)

`ReadKey()` - Read a character or function key pressed

Console Input & Output Method

`Write()` - Write in the same line
`WriteLine()` - Write in a new line

`ReadLine()` - Read a line (string)

`ReadKey()` - Read a character or function key pressed

Date Types

Data Type	Description
int	integer number
double	decimal number
string	word or setences
char	single character

Data Type	Description
bool	true or false

Variables

- Store data during program execution
- Has name and data type
- Declared before used
- Must begin with a letter
- Can contain letters, digits and underscore
- Cannot exceed 255 characters
- Reserved words such as `int`,`string`,`if`,`for`,`main` cannot be used
- Blank spaces are not allowed
- Must be unique within its scope

Example :

```
int x;
int y = x * 2;
string message1 = "PRG 2";
```

Constants

- Does not change during the execution of the program
- Declared with `const` keyword

Example :

```
const int c1 = 5;
const string message = "You can't get rid of me!";
```

If you try to change a constant value,
you will prompted with an error

Type Conversion Methods

Method	Example
ToInt32()	int numBook = Convert.ToInt32(Console.ReadLine()); * Create a variable `numBook` with data type of `int` and convert user input to integer type
ToDouble()	double rentalRate = Convert.ToDouble(Console.ReadLine()); * Create a variable `rentalRate` with data type of `double` and convert user input to double type
ToString()	double amount = 1234.5678; Console.WriteLine(amount.ToString("$#,##0.00")); * Create a variable `amount` with data type of `double` and convert user input to string type with specified pattern and display it

Arithmetic Operators

Operator	Description	Example `int a = 10, b = 20;`
`+`	Adds two operands	`a + b` will give `30`
`-`	Subtracts second operand from the first	`a - b` will give `-10`
`*`	Multiplies both operands	`a * b` will give `200`
`/`	Divides numerator by denominator	`b / a` will give `2`
`%`	Modulus Operator and remainder of after an integer division	`b & a` will give `0`
`++`	Increment operator increases integer value by one	`a++` will give `11`
`--`	Decrement operator decreases integer value by one	`a--` will give `9`

Relational Operators

Operator	Description	Example `int a = 10, b = 20;`
`==`	Checks if values of two operands are equal or not, if yes then condition becomes true	`(a == b)` is `false`
`!=`	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true	`(a != b)` is `true`
`>`	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true	`(a > b)` is `false`
`<`	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true	`(a < b)` is `true`
`>=`	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true	`(a < b)` is `false`
`<=`	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true	`(a <= b)` is `true`

Logical Operators

Operator	Description	Example `bool a = true, b = false;`
`&&`	Logical AND operator. If both the operands are true then condition becomes true	`(a && b)` is `false`
`&&`	Logical OR Operator. If any of the two operands true then condition becomes true	`(a || b)` is `true`
`!`	Logical NOT Operator. Use to reverse the logical state of its operand. If a condition is true then Logical NOT operator will be false.	`!(a && b)` is `true`

Concatenation Operators

- Join multiple strings into a single string
- Concatenation operators, `+`

```
string str;
str = "School of" + " " + "InfoComm Technology";
// str equals "School of InfoComm Technology"
```

- Operators can also concatenate string variables

```
string code = "EW23" ;
string name  = "Clementi" ;
string station;
station = code + name;
//station equals "EW23Clementi"
```

Control Structures - `if`

```
if (condition_1)
{
    statement_1
}

else if (condition_2)
{
    statement_2.1;
    statement_2.2
}

else
{
    statement_n
}
```

The `if` evaluate one or more condition.

- Along with a statement or statements to be executed if the condition is determined to be true, and
- Optionally, other sattements to be executed if the condition is determined to be false

Example :

```
int mark=67;
string result;

if (mark < 50)
    result = "Grade is F";
else if (mark < 60)
    result = "Grade is D";
else if (mark < 70)
     result = "Grade is C";
else if (mark < 80)
     result = "Grade is B";
else
     result = "Grade is A";
```

Control Structures - `while`

`while` executes a group of statements so long if the condition is true.
while Loop first tests condition

- If condition is `false`, it skips past all the statements
- If the condition is `true`, it executes the statements and goes back to the `while` statement and test the condition again

Loop can execute any number of times until the condition is `false`.

Statements never execute if condition is initially `false`.

Example :

```
int count = 1, sum = 0;
string result;

while (count < 10)
{
    sum += count;
    count++;
}
result = "1 + 2 + …+ 9 = " + sum;
```

Control Structures - `do-while`

It executes a group of statements so long if the condition is true

The statement will be executed at least once

Example :

```
do
{
    // statements;
} while (condition);
```

Control Structures - `for`

Executes a group of statements for a specific number of times

```
for ( initialization; conditional; increment/decrement )
{
    // statements to be executed
}
```

Example :

```
int sum = 0;
    string result;
        for (int count=1; count<10; count++)
        {
        sum += count;
        }
        result = "1 + 2 + …+ 9 = " + sum;
```

Control Structures - `foreach`

Executes a group of statements for a specific number of times

```
foreach ( initializer in string )
{
    // statements to be executed
}
```

Example :

```
string title = "programming";
    foreach (char c in title)
    {
        Console.WriteLine(c);
    }
```

List

- Zero indexed
- Initializing a list that contains number

```
List<int> numbers = new List <int> { 8, 3, 2 };
```

- Initializing an empty list and adding item subsequently

```
List<string> names = new List <string>();
names.Add("Matt");
names.Add("Joanne");
names.Add("Robert");
```

- Count property tells the number of items in List

```
int numberOfNames = names.Count;
```

Given `List<string> names = new List<string> {"Peter", "John", "Mary", "David"};`

You can display the content of list by using `for` or `foreach`.

Example :

```
for (int i=0; i<names.Count; i++)
{
   Console.WriteLine(names[i]);
}
```

```
foreach (string n in names)
{
   Console.WriteLine(n);
}
```

To modify the list content, `(listname)[list content number] = (value);`

Example :

```
List<int> marks = new List <int> {89, 77, 55, 69};

for (int i=0; i<marks.Count; i++)
{
   marks[i] = marks[i] + 5;
}
```

Array

- Stores fixed-size sequential collection of elements of the same type
- To declare and create an array

  ```
  int[] numbers = new int[10];
  ```

- To assign value into an array element

  ```
  numbers[0] = 99;
  ```

- You can also assign value into an array while declaring it

  ```
  int[] numbers = {1,2,3,4};
  ```

Given `string[] names ={"Peter", "John", "Mary", "David"};`

You can display the content of list by using `for` or `foreach`.

Example :

```
for (int i=0; i<names.Length; i++)
{
   Console.WriteLine(names[i]);
}
```

```
foreach (string n in names)
{
   Console.WriteLine(n);
}
```

Method

- Code block that contains a series of statement
- Allows programmer to modularize the program

Structure :

```
[static] return_type method_name ( [parameters] )
{
   …
}
```

Method which does not return a value, specify `void` as the `return_type`

```
static void Main(string[ ] args)
{
   DisplayMenu();
    //calling method that does not return value
}

static void DisplayMenu()
{
   Console.WriteLine("Menu");
   Console.WriteLine("1. Circle");
   Console.WriteLine("2. Square");
   Console.WriteLine("3. Triangle");
}
```

Method which returns a value, specify the `data type` as the `return_type`

```
static void Main(string[ ] args)
{
   int total = Sum(3, 5);        // calling method with return value
    // and stored in variable total
}

static int Sum(int n1, int n2)
{
    return (n1 + n2);
}
```

Method with Optional Parameter

- Allow calling of method with varying number of arguments to pass

- Optional parameter must specify a default value
- All optional parameters must be placed after the non-optional parameters

Example :
Method heading with default value defined for 2nd parameter

```
static int Sum( int n1, int n2 = 2 )
```

To call the method

```
int result = Sum(); // COMPILATION ERROR as int n1 value is not specified
int result = Sum(10); // int n1 will be 10 while int n2 will use the default value of 2
int result = Sum(10, 3); // int n1 will be 10 and int n2 will be 3, default value will be overridden
```

Date

To set dates in C#, we use `DateTime` class

- DateTime value is between 12:00:00 midnight Jan 01,0001 to 11:59:59PM Dec 31 9999.
- To create a specfic date and store it in a variable

  ```
  DateTime exampleDate = new DateTime(2021,1,1);
  ```

- To create the current date and time and store it in a variable

  ```
  DateTime currentDate = DateTime.Now;
  ```

- To print the date

  ```
  Console.WriteLine(exampleDate.ToString());
  Console.WriteLine(currentDate.ToString("dd/MM/yyy"));
  ```

You can use `Convert.ToDateTime(s)` to convert the string `s` into format of `yyyy/mm/dd` to a `DateTime` object

Example :

```
Console.Write("Enter date of birth (yyyy/mm/dd): ");
DateTime dob = Convert.ToDateTime(Console.ReadLine());
Console.WriteLine("Your birthdate is " + dob.ToString("dd/MM/yyyy");
```

AddDays() method in DateTime class

- Returns a new DateTime object that adds the specified number of days to the value of a DateTime object

Example :

```
DateTime currentDate = DateTime.Now;
DateTime oneWeekLater = currentDate.AddDays(7);
Console.WriteLine("Current Date: " + currentDate.ToString("dd/MM/yyyy"));
Console.WriteLine("One week later: " + oneWeekLater.ToString("dd/MM/yyyy"));
```

Subtract() method in DateTime class

- Returns a new TimeSpan object after subtracting the DateTime object specified in the parameter from the value of the DateTime

Example :

```
DateTime currentDate = DateTime.Now;
DateTime xmas = new DateTime(2020,12,25);
int diff = xmas.Subtract(currentDate).Days;
Console.WriteLine("Today is" + currentDate.ToString("dd/MM/yyyy"));
Console.WriteLine("There is " + diff + " days to xmas");
```

Reading Date from Text File

To read the whole text file into one string

```
using System.IO;
....
string text = File.ReadAllText("testing.txt");
Console.WriteLine(text);
```

To read the whole text file into a string array

```
using System.IO;
....
string[] lines = File.ReadAllLines("testing.txt");
foreach (string line in lines)
{
```

```
    Console.WriteLine("\t"+line);
}
```

Reading Data from csv File

To read the csv file with comma delimited into String Array

```
using System.IO;
....
string[] csvLines = File.ReadAllLines("testmarks.csv");

// Display the file contents together with average using a for loop
string[] heading = csvLines[0].Split(',');
Console.WriteLine("{0,10}  {1,10}  {2,10}  {3,10}",
        heading[0], heading[1], heading[2], "Average");
for (int i=1; i<csvLines.Length; i++)
{
    string[] marks = csvLines[i].Split(',');
    double average = (Convert.ToDouble(marks[1]) +
                Convert.ToDouble(marks[2])) / 2;
    Console.WriteLine("{0,10}  {1,10}  {2,10}  {3,10}", marks[0],
            marks[1], marks[2], average.ToString("0.00"));
}
```

Reading Data using StreamReader

```
using (StreamReader sr = new StreamReader("testmarks.csv"))
{
    string s = sr.ReadLine(); // read the heading
    // display the heading
    string[] heading = s.Split(',');
    Console.WriteLine("{0,10}  {1,10}  {2,10}  {3,10}",
            heading[0], heading[1], heading[2], "Average");
    // repeat until end of file
    while ((s=sr.ReadLine()) != null)
    {
        string[] marks = s.Split(',');
        double average = (Convert.ToDouble(marks[1]) +
            Convert.ToDouble(marks[2])) / 2;
```

```
        Console.WriteLine("{0,10}  {1,10}  {2,10}  {3,10}",
            marks[0], marks[1], marks[2], average.ToString("0.00"));
    }
}
```

Writing Data to Text File

To write an array of strings to a text file

```
using System.IO;
....
string[] lines = {"First line", "Second line", "Third line"};
File.WriteAllLines("WriteLines.txt", lines);
```

To write one string to a text file

```
using System.IO;
....
string text = "This is a string.";
File.WriteAllText("WriteText.txt",text);
```

Writing Data to Text File using StreamWriter

To write each string in a list to a text file

```
List<string> aList1 = new List<string> { "one", "two", "three" };
using (StreamWriter sw = new StreamWriter("WriteLines2.txt", false))
{
    foreach (string s in aList1)
    {
        sw.WriteLine(s);
    }
}
```

Append Data to Existing csv File

```
while (true)
{
```

```
    Console.Write("Enter student ID (or Exit to terminate): ");
    string id = Console.ReadLine();
    if (id == "Exit") break;
    Console.Write("Enter test 1 mark: ");
    string test1 = Console.ReadLine();
    Console.Write("Enter test 2 mark: ");
    string test2 = Console.ReadLine();
    string data = id + "," + test1 + "," + test2;
    using (StreamWriter sw = new StreamWriter("testmarks.csv", true))
    {
        sw.WriteLine(data);
    }
}
```

What are Classes

A class is like a "cookie-cutter", it defines the shape of objects.
Objects are like cookies; they are instances of the class.

In C#, To define a class we have

Characteristics

```
1. Name of a Class
2. Field(s)
3. Property or Properties
```

Responsibilities

```
4. Constructor(s)
5. Method(s)
```

Name of a Class

```
class Student
{

}
```

Field(s)

```
class Student
{
    private string firstname;
    private string lastname;
}
```

Property or Properties

```
class Student
{
    private string firstname;

    public string Firstname
    {
        get { return firstname; }
        set { firstname = value; }
    }
}
```

Constructors(s)

```
class Student
{
    private string firstname;

    public string Firstname
    {
        get { return firstname; }
        set { firstname = value; }
    }

    private string lastname;

    public string Lastname
    {
        get { return lastname; }
        set { lastname = value; }
    }

    public Student(string fn, string ln)
```

```
    {
        Firstname = fn;
        Lastname = ln;
    }
}
```

Method(s)

```
public Student(string fn, string ln)
{
    Firstname = fn;
    Lastname = ln;
}

public string StudentFullName()
{
    return Firstname + " " + Lastname;
}
```

Creating an Object from a Class

To create an object from a class, you need to invoke a constructor of the class using the new operator as follows

```
ClassName ObjectName = new ClassName(argument…);
```

Example :

```
Student student1 = new Student("Peter", "Pan");
Student student2 = new Student("John", "James");
```

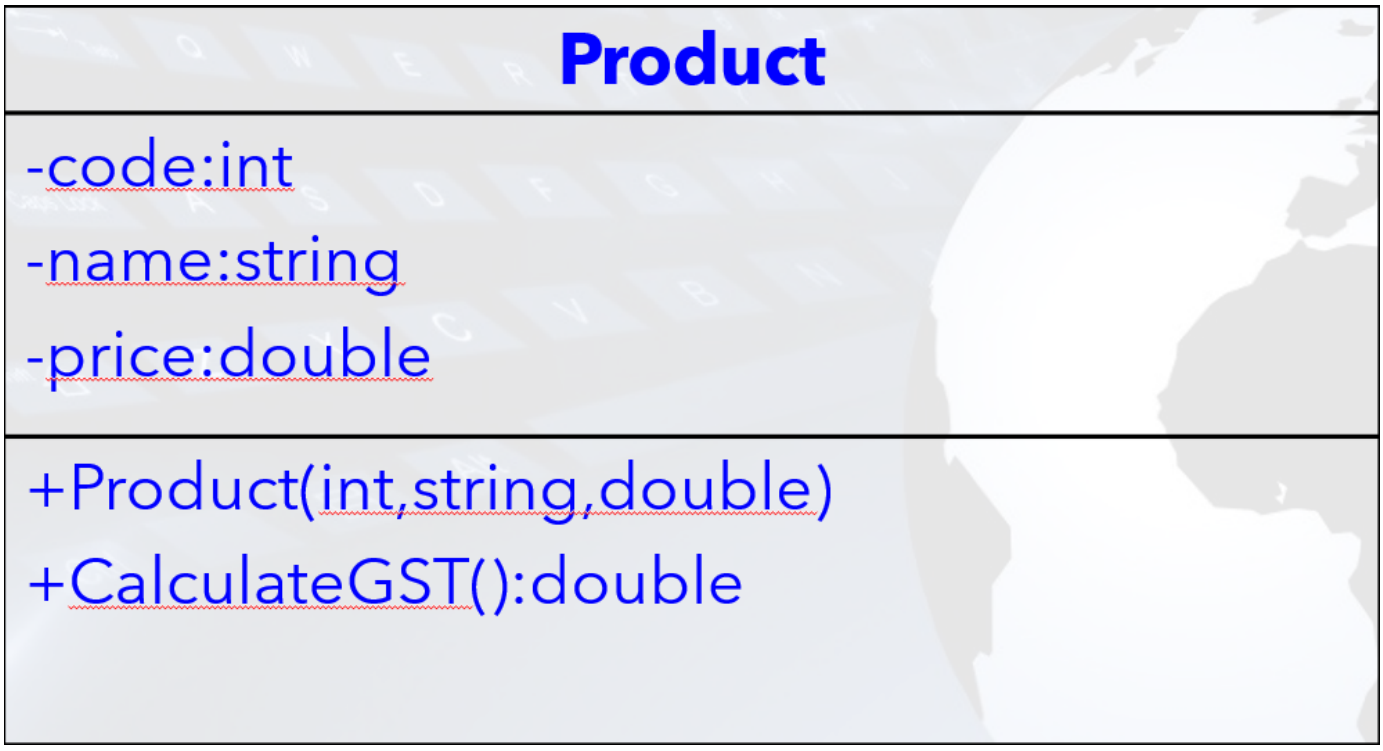
Both student1 and student2 object are instances of Student Class

---

Class Diagram

A Class can be represented using the Class Element diagram in the UML notation as shown below

Example :



`- private` (not acessible by other class, only accessible by it's own methods)
`+ public` (accessible by other classes)

Accessing data in Object

Data can be accessed via Property

```
Product p1 = new Product(1001, "iPhoneX", 1088.00);
double oldPrice = p1.Price;      // read (get) value
p1.Price = 888.00;               // update (set) value
```

Accessing methods in Object

Methods can be accessed directly by using the `.` operator

Example :

```
double gst = p1.CalculateGST();
Console.WriteLine("GST of p2 = " + p2.CalculateGST());
```

Encapsulation

- Hides irrelevant data from user
- Classes use encapsulation to hide its members that are not relevant for an outside class

Access Specifiers	Visible to objects of other classes
private	No
public	Yes

Example :

Not acceessible by objects of other classes

SavingAccount
-accountNo:string
-accountName:string

SavingAccount
-balance:double

Accessible by objects of other classes

SavingAccount
+SavingsAccount()
+SavingsAccount(string,string,double)
+Deposit(double)
Withdraw(double):bool
+ToString():string

Implementing the Attributes

- Implemented using instance variables
- Declared `private`

Example :

```
private string accountNo;
private string accountName;
private double balance;
```

Implementing the Properties

1 Property for each attribute.
Both have the same name.
Attribute name starts with lowercase.
Property name starts with Uppercase.

```
// private attributes
private string accountNo;
private string accountName;
private double balance;

// public properties
```

```
public string AccountNo
{
   get { return accountNo; }
   set { accountNo = value; }
}
public string AccountName
{
   get { return accountName; }
   set { accountName = value; }
}
public double Balance
{
   get { return balance; }
   set { balance = value; }
}
```

Simplified Version of Properties

```
// public properties
public string AccountNo { get; set; }
public string AccountName { get; set; }
public double Balance { get; set; }
```

Implementing the Behaviour

- 3 types of methods in a user-defined class
  - Constructors // for creating objects of the class
  - Other methods // for performing some specific task
  - ToString() method // for returning the information of an object if possible

---

Implementing the Constructor

- Special method of a class that is executed whenever an object is created from the class
  - Must have same name as the class
  - Constructors have NO return type (not even `void`)
  - No-argument constructor (default)
    - Constructor with no parameters
  - Parameterized constructor

    - Constructor with parameters
- Invoked using the new operator

Syntax

```
<access modifier> ClassName(){...}
```

Example :

```
// default constructor - does not have parameters
public SavingsAccount() { }

// parameterized constructor - with parameters
public SavingsAccount(string no, string name, double bal)
{
     AccountNo = no;
     AccountName = name;
     Balance = bal;
}
```

Implementing the other Methods

```
public void Deposit(double amount)
{
     Balance = Balance + amount;
}
public bool Withdraw(double amount)
{
     if (amount <= Balance)
     {
          Balance = Balance - amount;
          return true;
     }
     else
          return false;
}
```

Calling Methods of an Object

- Use the `.` notation to call your method

```
acc1.Withdraw(50);
```

- Able to accesss the methods because the methods have been declared `public`
  - `public` methods can be accessed by all classes
  - `private` data can be accessed only within the class itself

Implementing the `ToString()` method

- Converts an object to its stirng representation so that it is suitable for display
- Can override `ToString()` method to return values that are meaning for those types

Example :

```
public override string ToString()
{
    return "AccountNo:" + AccountNo +
          " AccountName:" + AccountName +
          " Balance:" + Balance;
}
```

To use `ToString()`

```
Console.WriteLine(acc1.ToString());
```

Accessing data and methods of an object

```
class Program
{
    static void Main(string[] args)
    {
        // create 2 objects, acc1 and acc2, of SavingsAccount class
        SavingsAccount acc1 = new SavingsAccount("111", "John", 200.0);
        SavingsAccount acc2 = new SavingsAccount("222", "Mary", 100.0);
        // withdraw $50 from acc1
        acc1.Withdraw(50);
        // print the balance of acc1
```

```
            Console.WriteLine("Balance of Account 1 : " + acc1.Balance);
            // deposit $100 to acc2
            acc2.Deposit(100);
            // print the detail of acc2
            Console.WriteLine(acc2.ToString());
        }
}
```