Contents

Program structure and order

First entry point for code execution is main method. Only the commands or methods from main method are executed.

```
static void Main(string[] args)
 {
     int a = 6; //a has value but nothing is printed out
 }
```

Creation of class objects or calling out methods happens in main method!

Curly braces indicate the content of a code block; a method, class, if sentence, loop etc.

All methods and classes have to be written after the previous one, not inside another.

Correct:

```
static void Main(string[] args)
 {//Start of Main method
     int a = 6; //a has value but nothing is printed out

 }//end of Main method

 static void MyMethod()
 {//Start of MyMethod
     Console.WriteLine("test");
 }//end of MyMethod
```

Wrong:

```
static void Main(string[] args)
{//Start of Main method
     int a = 6; //a has value but nothing is printed out

     static void MyMethod() //method cannot be inside another method
     {//Start of MyMethod
         Console.WriteLine("test");
     }//end of MyMethod
 }//end of Main method
```

Variables

Variables have different types. Variables store values. Variable type needs to be defined only once. Variable names can be used only once (you cant use two different variables with name „a“). Variable names can be any word but should be descriptive of its state.

Variable values are set with '='

Variable names start with small letter.

Syntax: <variable type> <name> = <value>;

Examples:

```
int a; //declaring we have an integer with name a, no value
a = 5; //setting value of a to 5
a = 6;//setting value of a to 6, overwriting previous value 5

int j = 5;//declaring we have an integer with name j and value 5

string test = "test"; //defining new string variable named test and with value test
```

Common mistakes:

Variable type needs to be defined only once. Variable names can be used only once.

```
int i = 5;
int i = 5; //dont define the type twice!

i = "tere"; //int cant be string, in "" is always string, correct is i=5
string a = 5; //string cant be numeric, correct would be string a="5"
```

Additional reading:

https://www.tutorialspoint.com/csharp/csharp_variables.htm

Methods

Defining

Methods are re-usable code blocks; stored procedures. Void means that method does not return anything; only fulfills commands. Method body is stored in between {}. Method names start with capital letters and are a verb. Methods do smth; they change the variables.

Method signature is method parameters.

Method syntax:

<accessibility><return type><method name>(<parameters>)

{

<method body>

}

Example:

```
public int SumOfTwoNumbers(int a, int b)
{
    return a + b; //returns value of type int
}
```

In this example:

<accessibility>- public; means that method can be called out anywhere

<return type>– int; means that method has to return int type value. If return type is not void then methods always have to return a value (indicated by “return” statement).

<method name>– SumOfTwoNumbers; method can be called out by this name

<parameters> - - `int a, int b;` Parameters values that are used during method execution. Parameters are always separated by comma. Parameters can range from 0 to infinity but good practice is not to use more than 4.

<method body> - `return a + b;` the task that the method fulfills. In this example sum of a and b is returned. Method body can contain 1 to n sentences; for example Console.WriteLine(), also calling out other methods.

If method is defined but not executed (called out) then nothing happens!

Method execution

Method is executed by its name, then followed by (). If there are parameters, they are inside (). Parameters are always concrete values.

Syntax: <method name>(<concrete parameter values which we want to use>)

If we had method:

```
            public void  SumOfTwoNumbers(int a, int b)
            {
                Console.WriteLine(a + b); //returns value of type int
            }
```

The execution would look like:

```
       SumOfTwoNumbers(3,4); //prints sum of 3 and 4; number 7
```

What happens in the background: when calling out the method. 3 is used as the first parameter so a=3, 4 is used as the second parameter so b=4. Inside method body the statement would be:

```
Console.WriteLine(3 + 4);
```

Common mistakes

- Wrong parameters.

If we had method:

```
            public void DoSmth(int a, string b)
            {
                Console.WriteLine(a + b); //returns value of type int
            }
```

We can only call it out by its correct parameters (first int, then string) so:

```
DoSmth(4, “tere”);
```

We cannot do:

```
DoSmth(4, 2); DoSmth(“4”, “tere”); DoSmth(4); DoSmth();
```

- Using ‘;’ after method declaration

```
         public void DoSmth(int a, string b); //NEVER EVER PUT ';' HERE
        {
           Console.WriteLine(a + b); //returns value of type int
        }
```

- No return statement when method is not *void*

Wrong:

```
        public int DoSmth(int a, string b)

        {
            Console.WriteLine(a + b);
            //no return sentence!
        }
```

Correct:

```
        public int DoSmth(int a, int b)
             {
                 Console.WriteLine(a + b);
                 return a+b; //has return sentence
             }
```

- Calling out method and using variable with no value as parameter

Wrong:

```
static void Main(string[] args)
{
        DoSmth(string b); //b has no value, we cannot use this as method parameter
}

static void DoSmth(string value)
{
     //do smth
}
```

Correct:

```
static void Main(string[] args)
{
    string b = "5";
    DoSmth(b); //b has value of "5"

    DoSmth("give"); //call out method with direct value
}

static void DoSmth(string value)
{
    //do smth
}
```

Static vs non-static methods

Static methods – can be called out without creating a class instance

Non-static methods – can be called out only from class instance

Example:

```
class Program
{
    class TestingClass
    {
        static void Main(string[] args)
        {
            Example ex = new Example();
            ex.IncreaseNumber(5); //non-static method

            IncreaseNumber(4); //static method
        }

        static void IncreaseNumber(int number)
        {
            Console.WriteLine("Static " + number + 1);
        }
    }


    class Example
    {
        public void IncreaseNumber(int number)
        {
            Console.WriteLine("Non-statcic" + number + 1);
        }
    }
```

Additional reading:

https://www.tutorialspoint.com/csharp/csharp_methods.htm

If-sentence

An if statement identifies which statement to run based on the value of a Boolean expression. If sentence can be without else sentence or multiple else if sentences.

```
bool condition = true;

if (condition)
{
    Console.WriteLine("The variable is set to true.");
}
else
{
    Console.WriteLine("The variable is set to false.");
}
```

Syntax:

```
        if (condition)
        {
           //if condition is true
        }
        else
        {
           //if condition is false
        }
```

Operands: (there cannot be space between operands, ‘!=’ cannot be ‘!   =’)
== equal
!= not equal
|| or
&& and

Condition examples:

```
        if (a>=5 && a<10) //if a is between 5 and 9
        if(a==4 || a==5) // if a is 4 or 5
```

Common mistakes

- Writing operators in singular way.

Wrong:

```
        if(a=4) //Comparison is always with ‘==’ sign, ‘=’ means giving value
```

Correct:

```
        If(a==4)
```

Additional reading:

https://www.dotnetperls.com/if

Loops

Loops are used for executing a same command multiple times. Possible usages: printing out list items (for, foreach, while), populating list (foreach, while), asking user for input (while).

Often different loops can be used for solving same tasks but some of them are easier to write than others.

Foreach

Simplest loop. Example usages: iterating through list items and printing them out. If foreach loop is sufficient (does what you need), then its good to use it.

Syntax:

foreach(<itemType> <itemName> in <collectionName>)

{

    //do smth with <itemName>

}

Example:

```
List<string> names = new List<string>() { "John", "Tom", "Peter" };
foreach (string name in names)
{
    Console.WriteLine(name);
}
```

```
Output:

"John",

 "Tom",

"Peter"
```

Additional materials: http://www.csharp-examples.net/foreach/

For-loop

Can do anything a foreach loop can do and more. Adds possibility of doing a certain task x times. For example populating a list with x numbers. Mistakes can be made in setting the condition.

Syntax:

```
  for (<starting value>; <condition>; <incremental operation>)

   {

     //do smth

   }
```

Example:

```
for (int i = 0; i < 10; i++)
{
    Console.WriteLine(i);
}
```

Prints out numbers from 0 to 9.

Example 2; adding 100 values to list:

```
            static void Main(string[] args)
            {
                List<int> listOfInts = new List<int>();

                for (int i = 0; i < 100; i++)
                {
                    listOfInts.Add(i);
                }

                Console.WriteLine(listOfInts.Count); //prints 100 since we have 100 items
in the list
            }
```

Additional reading: https://www.dotnetperls.com/for

While

The most complex loop in a sense that its easiest to do mistakes and create infinite loops. Good to be used for example in a case where you expect a correct input from the user (ask a question until the user gives a correct answer).

Syntax:

```
while (condition)
{
      //do smth
}
```

Example:

```
 int i = 0;
 while (i < 10)
 {
    Console.WriteLine(i);
    i++;
 }
```

Output is numbers from 0 to 9.

Common mistakes:

- Condition is not changed and infinite loop is created

Wrong:

```
        int j = 0;
        while (j < 10) // j is always j, value of j is not changed. Infinite loop
        {
           Console.WriteLine(j);
        }
```

Correct:

```
int j = 0;
while (j < 10)
{
    Console.WriteLine(j);
    j++;// value of j is increased; loop is exited when value of j is 10
 }
```

Classes

Classes represent objects. Class consists of fields, variables and constructors.

Class declaration is like stating that we have a factory that can produce objects with certain capabilities (methods) and properties (fields).

Class objects are created in main method. Unless we create an object and call out methods, class code is not executed.

Class declaration starts with word “class” and is followed by class name and{}. Class name is always with capital letter.

Example:

```
class Test
{

}
```

Variables

Classes have variables that store value. These variables are called fields. They describe the objects. For example if object was car, the fields could be color, width, number of wheels, power, mileage, mark, make, height etc.

Class variables are usually private and set by methods or constructor.

Methods

Class methods can modify class variables. They are the same as methods in point 2. Class methods indicate action and their name is also a verb. Methods do smth.

For example for class Car method Drive could increase mileage.

There can be multiple methods with same name, as long as they have different parameters. Its called method overloading. Method to execute is chosen by parameters. Here is method Drive with tow different signatures:

One takes no parameters and the other one parameter with type int.

Example:

```
class Program
{
    static void Main(string[] args)
    {
```

```
                Car wolkswagen = new Car(); //creation of the new object from class car
with name 'wolkswagen'
                wolkswagen.Drive(); //drive method with no parameters is called, mileage
is 4
                wolkswagen.Drive(10); //drive method with int parameter is called,
mileage is now 14
            }
        }


        class Car
        {
            public int mileage=0; //this is class field

            public void Drive() //drive method with no parameters
            {
                mileage += 4; //adds always 4 to mileage
            }

            public void Drive(int amountToDrive) //drive method with int parameter
            {
                mileage += amountToDrive; //adds the value of method parameter to mileage
            }

        }
```

Constructor

Constructors are used for creating objects. Constructors are called out with keyword new.
If you do not specify a constructor, then an empty constructor is used by code.

Constructors are usually public since they are used to create objects.
Constructor syntax is very similar to method syntax but:

- They don’t have return type
- Their name is always the same as class name

Constructors can have parameters the same way as methods.

Class can have multiple constructors. Correct constructor to use is chosen by parameters.

Syntax:

<access specifier> <class name> (<parameters>){

//body

}

Example:

```
    class Test
    {
        public Test(string a) //constructor with one string parameter
        {
            //do smth
        }
    }
```

Creating objects

By defining a class nothing is printed out or done yet. It just means that you have a capability of creating that type of objects.

In order to do so, you need to create a new object. Object is created by keyword ‘new’. Multiple objects can be created.

Syntax:

<ClassName> <yourObjectName> = new <ClassName>();

Calling out methods:

<yourObjectName>.MethodName();

Calling out fields:

<yourObjectName>.<fieldname>;

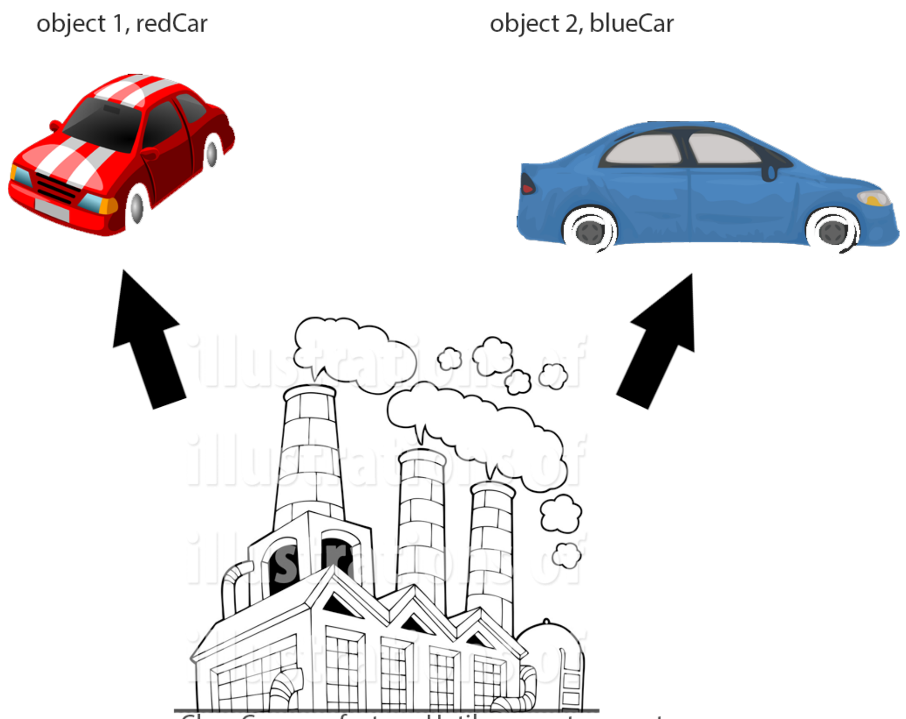
Example of a class car with multiple constructors and methods.

We can imagine creating a class Car as creating a factory that is capable of producing cars that have properties mileage and color. The cars also have methods for driving and painting and also for adding the turbo. When we create a new car we can also specify the color (different constructors) and weather to add turbo.

Describing this factory does not create any objects.

In main method we can create real objects and then the code from Car class is executed. We can imagine this that our production lines build real concrete physical cars; first car is called redCar and second one blueCar. RedCar and blueCar have different color and mileage.

Class Car is equal to the factory and real objects (red car and blue car) are created (the factory is used) in main method.



Class Car - our factory. Until we create conrete objects, factory is not used.

Class code:

```
        class Program
        {
            static void Main(string[] args)
            {
                Car redCar = new Car("red"); //we create a new car called redCar with
constructor that takes string color as parameter
                                             //mileage is set to 0 and power is 100

                redCar.AddTurbo();// we add turbo, now power is 110kw
                redCar.Drive(20); //we drive 20 km so mileage is now 20km
                redCar.Drive(4); //we drive 4km more so mileage is now 24

                Car blueCar = new Car(200); //we create a new car called blueCar with
constructor that takes int power a parameter
                                            // mileage is 0, color is not set and power
is 200

                blueCar.PaintCar("blue");  //car color is now blue
                blueCar.Drive(); //we call out drive method without parameters, mileage
is now 10
```

```
                redCar.PrintOutCarInfo();
                //Output is:
                //car color is red, power is 100 and mileage is 24

                blueCar.PrintOutCarInfo();
                //Output is:
                //car color is blue, power is 200 and mileage is 4
            }

        }

        class Car
        {
            private int mileage; //field, car mileage in km
            private string color; //field, color as string
            private int power; //field, power in kw

            public Car(string carColor) //construcor that takes color as parameter
            {
                color = carColor; //color is the value that is given during object
creation
                mileage = 0; //we set mileage to 0
                power = 100; //all default cars have a power of 100kw
            }

            public Car(int carPower) //construcor that takes power as parameter
            {
                mileage = 0; //we set mileage to 0
                power = carPower; //mileage that is given during object creation
            }

            public void AddTurbo() //method for adding turbo
            {
                power += 10; //we add +10 kw to power if turbo is added
            }

            public void PaintCar(string carColor) //method for changing the color of the
car
            {
                color = carColor; //we set a new color for the car
            }

            public void Drive() //drive method with no parameters
            {
                mileage += 4; //adds always 4 to mileage
            }

            public void Drive(int amountToDrive) //drive method with int parameter
            {
                mileage += amountToDrive; //adds the value of method parameter to mileage
            }

            public void PrintOutCarInfo() //method for printing out car info
            {
                Console.WriteLine("Car color is {0}, power is {1} and mileage {2}",
color, power, mileage);
            }
        }
```

Additional reading:

Watch this video http://www.codinghomework.com/c-exercise-vending-machine-using-oop-objects-classes-methods/

Try this example: https://www.juanantonioripoll.es/practice-exercises-c-sharp/practice-exercises-c-sharp-exercise-7-5-Complex-numbers.aspx

Lists

Lists is a class in C# which is meant for handling of objects of same type.

List has many pre-defined functions, for example sorting.

List items are indexed starting from zero and can be accessed through indexes.

List items can be added with method List.Add(valueToAdd).

Syntax:

List<itemType> <listName> = new List<itemType>();

Example:

```
static void Main(string[] args)
{
    List<string> myList = new List<string>();
    myList.Add("tomato");
    myList.Add("banana");
    myList.Add("potato");

    myList.Sort(); //sorts the values alfabetically, so list is now:
                   //myList[0] = "banana";
                   //myList[1] = "potat0";
                   //myList[2] = "tomato";

    foreach (string a in myList) //printing out all items with foreach loop
    {
        Console.WriteLine(a);
    }
}
```

Common mistakes:

- Calling out methods not from concrete list object

List is a class so in order to call methods (sort, add) we need to do it from a concrete list instance. Otherwise the code does not know which list to use.

Wrong:

```
List<string> myList = new List<string>();
```

```
                List.Add("tomato"); //there is no list with name List, its class name
                List.Add("banana");
                List.Add("potato");

                List.Sort();
```

```
Right:

                List<string> whateverIsTheNameOfYourList = new List<string>();
                whateverIsTheNameOfYourList.Add("tomato"); //you have to use the name of
the list you want to use
                whateverIsTheNameOfYourList.Add("banana");
                whateverIsTheNameOfYourList.Add("potato");

                whateverIsTheNameOfYourList.Sort();
```

General rules and common mistakes:

1) Always use ';' sign at the end of a statement (statement is setting variable value or writing a line etc); for example int a=5; Console.WriteLine("someText");
   This sign indicates that sentence is complete. (Like a '.' at the end of a sentence in text)
2) NEVER EVER use a ';' in between () and {}, for example
   Fsfs

```
       if (a > b) ; //never ever use a ';' here, this is code structure

       {

          Console.WriteLine(); //this is a statement so we need ';' here to
indicate a end of the line

       }
```

3) Don't declare variables twice. Types need to be indicated only once.

```
int i = 5;
int i = 5; //dont define the type twice!
i=6; //we can assign a new value just by setting it, type is already declared before

i = "tere"; //int cant be string, in "" is always string, correct is i=5
string a = 5; //string cant be numeric, correct would be string a="5"
```

4) After methods there always have to be brackets (indicating it's a method).
   DoSmth(); //execution of a method with no parameters
   DoSmth(4, 5); //execution of a method with 2 int type parameters

   Wrong:
   DoSmth()=4 //we cant use = to set method values, we can only set variable values like this

5) Trying to print out all list items without a loop. We can only print out using all list items using a loop.

Wrong:

```
List<string> myList = new List<string>();
myList.Add("tomato");
myList.Add("banana");
myList.Add("potato");

Console.WriteLine(myList); //prints only the type of the list, not items
```

Right:

```
List<string> myList = new List<string>();
myList.Add("tomato");
myList.Add("banana");
myList.Add("potato");

foreach (string a in myList) //printing out all items with foreach loop
{
    Console.WriteLine(a); //prints out item
}
```

6) Blaming the code for “not doing anything”. Only the things you tell the code to do are done.

Example:

```
class Program
{
    static void Main(string[] args)
    {
        int a = 6; //int a value is 6
        int b = 7; //int b value is 7
      }

    static void PrintOutText()
    {
        Console.WriteLine("Tere");
    }
}
```

This program prints out nothing. Value of a is set to 6, value of b is set to 7. A method is defined but not executed (i.e. a procedure is defined but not called out)

```
class Program
{
    static void Main(string[] args)
    {
        PrintOutText(); //prints "Tere"
        int a = 6; //int a value is 6, nothing is printed
        int b = 7; //int b value is 7, nothing is printed
        Console.WriteLine(a); //prints value of a, 6
        Console.WriteLine(b); //prints value of b, 7
        PrintOutText(); //prints "Tere"
    }

    static void PrintOutText()
    {
        Console.WriteLine("Tere");
    }
```

```
        }
```
This code outputs:

```
Tere

6

7

Tere
```

Exercises:

How to try exercises.

1) Try to think the solution
2) If you really dont have any ideas then peek the solution and copy to visual studio
3) Try running the exercise and changing different parts to see what happens
4) Delete the solution and try to write it yourself

Try out different exercises (also change topics) from here: https://www.w3resource.com/csharp-exercises/for-loop/index.php