C# - Ultimate Guide - Beginner to Advanced | Master class

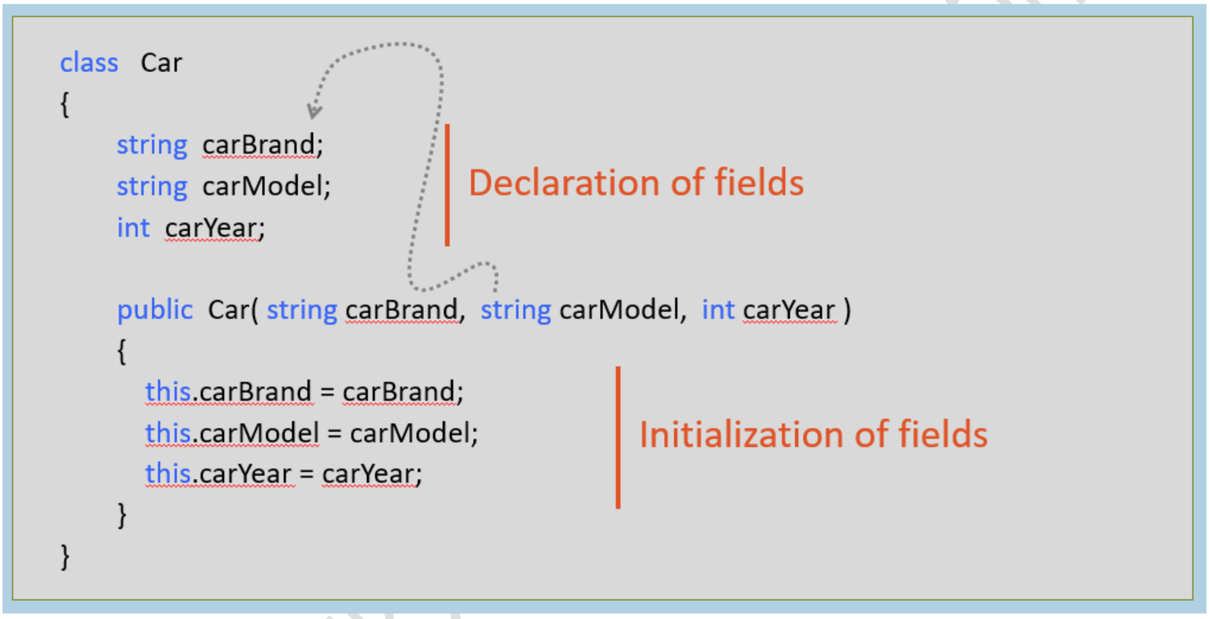
Section 8 - Constructors

Constructors

Special method of class, which contains initialization logic of fields.

Constructor initializes the fields and also contains the additional initialization logic (if any).

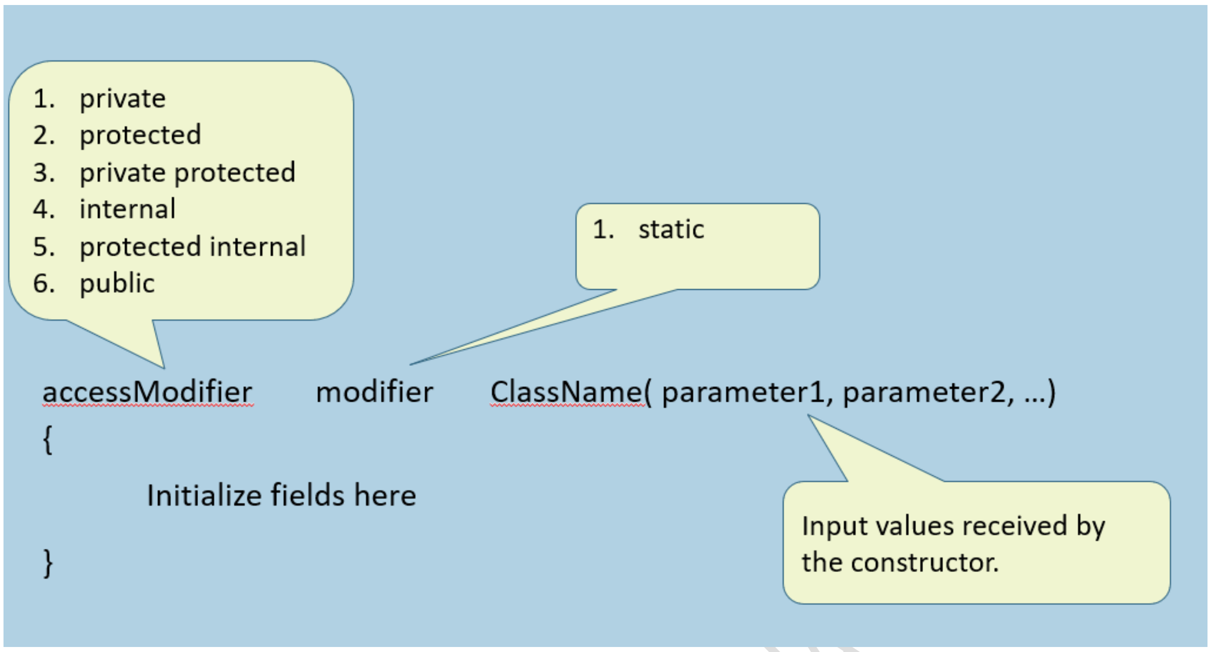
Eg:



```
class  Car
{
       string  carBrand;
       string  carModel;
       int  carYear;

       public  Car( string carBrand,  string carModel,  int carYear )
       {
          this.carBrand = carBrand;
          this.carModel = carModel;
          this.carYear = carYear;
       }
}
```

Syntax of Constructor



```
accessModifier     modifier     ClassName( parameter1, parameter2, ...)
{
          Initialize fields here
}
```

Rules of Constructors

- Constructor's name should be same as class name.
- Constructor is recommended to be "public" member or "internal" member;
- if it is a "private member", it can be called within the same class only; so you can create object of a class only inside the same class; but not outside the class.
- Constructor can have one or more parameters.
- Constructor can't return any value; so no return type.
- A class can have one or more constructors; but all the constructors of the class must have different types of parameters.

Instance (vs) Static Constructor

Instance Constructor

```
public ClassName( Parameter1, Parameter2, … )
{
   …
}
```

1. Initializes instance fields.
2. Executes automatically every time when a new object is created for the class.
3. "private" by default; We can use any of access modifiers.
4. Can contain any initialization logic, that should be executed every time when a new object is created for the class.

Static Constructor

```
static ClassName( )
{
   …
}
```

1. Initializes static fields.
2. Executes only once, i.e. when first object is created for the class or when the class is accessed for the first time during the execution of Main method.
3. "public" by default; Access modifier can't be changed.
4. Can contain any initialization logic, that should be executed only once i.e. when a new object is created for the class.

Parameter-less (vs) Parameterized Constructor

Parameter-less Constructor

```
public ClassName( )
{
  …
}
```

1. Constructor without parameters.
2. It generally initializes fields with some literal values (or) contains some general-initialization logic of object.

Parameterized Constructor

```
public ClassName( Parameter1, Parameter2, … )
{
  …
}
```

1. Constructor with one or more parameters.
2. It generally initializes fields by assigning values of parameters into fields.

Implicit (vs) Explicit Constructor

Implicit Constructor (after compilation)

```
public ClassName( )
{
}
```

1. If there is a class without constructor, then the constructor automatically provides an empty constructor, while compilation, which initializes nothing. It is called as "Implicit Constructor" or "Default Constructor".
2. It is just to satisfy the rule "Class should have a constructor".

Explicit Constructor (While coding)

```
public ClassName( with or without parameters )
{
  …
}
```

1. The constructor (parameter-less or parameterized) while is created by the developer is called as "Explicit Constructor".
2. In this case, the C# compiler doesn't provide any implicit constructor.

Constructor Overloading

Write multiple constructors with same name in the class, with different set of parameters (just like 'method overloading').

It is recommended to write a parameter-less constructor in the class, in case of constructor overloading.

Constructor Overloading (multiple constructors in the same class)

```
public ClassName( )
{
}

public ClassName( parameter1, parameter2, … )
{
  …
}
```

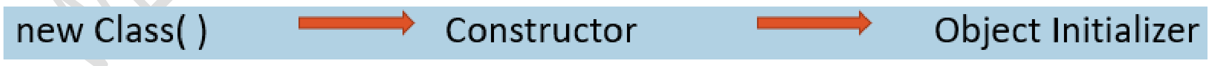
Object Initializer

Special syntax to initialize fields / properties of class, along with creating the object.

Executes after the constructor.

It is only for initialization of fields / properties, after creating object; it can't have any initialization logic.

Execution:



```
new ClassName( ) { field1 = value, field2 = value, … }
```

Use 'object initializer' when:

1. there is no constructor present in the class; but you want to initialize fields / properties.
2. (or) there is a constructor; but it is meant for initializing other set of fields, other than the fields that you want to initialize.

Key Points to Remember

1. 'Instance constructor' initializes 'instance fields'; but also can access 'static fields'.
2. 'Static constructor' initializes 'static fields'; can't access 'instance fields'.
3. Default (empty constructor) is provided automatically by C# compiler, if the developer creates a class without any constructor.
4. It is always recommended to write a parameter-less constructor first, if you are creating parameterized constructor.
5. Use 'object initializer', if you want to initialize desired fields of an object, as soon as a new object is created.