Krishna Nigalye

Software Developer

Learning C-Sharp
Constructors

WHAT IS A CONSTRUCTOR?

➢ Constructor is a special method which is invoked when an instance of the class is created.
➢ Constructors plays very important role in Classes and Structs in the initialization process. We are going to keep this tutorial specific to a class.

Note: This is just a sample code for explanation point of view.

HOW TO USE CONSTRUCTORS?

➢ Name of the constructor is always same as the name of the class.
➢ Constructors do not have any return type.

```csharp
public class Staff
{
    private string _fullName;
    private string _title;

    public Staff(string fullName, string title)
    {
        _fullName = fullName;
        _title = title;
    }
}
```
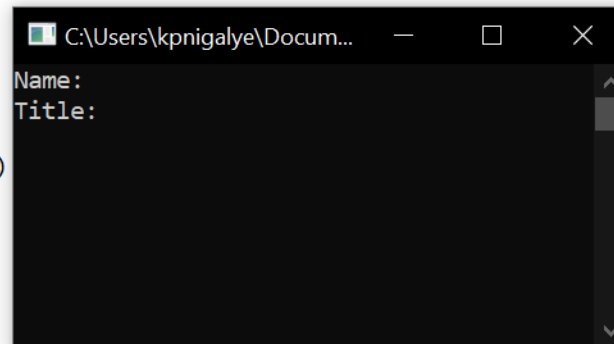
➢ The constructor defined in the above code accepts parameters so it is called as a **Parameterized Constructor**.
➢ If you don't declare constructor for your class, the compiler automatically creates a parameterless constructor which instantiates the instance and assign member variables to their default values. Remember, default constructors are always parameterless.

```csharp
public class Staff
{
    public string FullName;
    public string Title;

    public void PrintInfo()
    {
        Console.WriteLine($"Name: {FullName} \nTitle: {Title}");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Staff staff = new Staff();
        staff.PrintInfo();

        Console.ReadLine();
    }
}
```

```
C:\Users\kpnigalye\Docum...      —    □    ✕
Name:
Title:
```

As you can see, our class 'Staff' don't have any constructor so compiler provides a parameterless constructor to initialize the instance of the class.

A constructor that takes no parameters is called a parameterless constructor. **Parameterless Constructors** are invoked whenever an object is instantiated by using the *new* operator and no arguments are provided to *new*.

Check out the code below.

```csharp
public class Staff
{
    private string _fullName;
    private string _title;

    public Staff(string fullName, string title)
    {
        _fullName = fullName;
        _title = title;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Staff staff = new Staff();

        Console.ReadLine();
    }
}
```

Staff.Staff(string fullName, string title)

There is no argument given that corresponds to the required formal parameter 'fullName' of 'Staff.Staff(string, string)'

Show potential fixes (Alt+Enter or Ctrl+.)

As you can see, our class has a parameterized constructor and if we try to create an object without passing a parameter, compiler will show you an error.

Therefor to create an object of this class, you have to pass parameters during object initialization like the code below.

```csharp
Staff staff = new Staff("John", "Cena");
```

➤ A class can have any number of constructors. In the code shown below, you can see that our class has two parameterized constructors. Like that there can be n number of constructors but with different signatures.

```csharp
public class Staff
{
    private string _fullName;
    private string _title;
    private int _salary;

    public Staff(string fullName, string title)
    {
        _fullName = fullName;
        _title = title;
    }

    public Staff(string fullName, string title, int salary)
    {
        _fullName = fullName;
        _title = title;
        _salary = salary;
    }
}
```
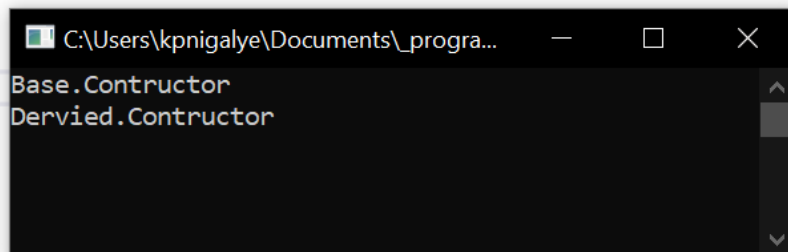
The two constructors defined above are called **Instance constructors** because they are used to create a new object of the class.

CALLING A BASE CLASS CONSTRUCTOR FROM DERIVED CLASS CONSTRUCTOR

- ➤ C# allows you to call base class contractor from derived class constructor using '*base*' keyword.
- ➤ As you can see in the output, call to the base class constructor is made first and then control goes to derived class constructor.
- ➤ Base class can be used with or without parameters depending on the situation.

```csharp
public class Vehical
{
    private string _vehicalNumber { get; set; }

    public Vehical(string vehicalNumber)
    {
        _vehicalNumber = vehicalNumber;
        Console.WriteLine("Base.Contructor");
    }
}

public class SportsCar : Vehical
{
    public SportsCar(string vehicalNumber) : base(vehicalNumber)
    {
        Console.WriteLine("Dervied.Contructor");
    }
}
```

```
C:\Users\kpnigalye\Documents\_program...        —    □    ✕
Base.Contructor
Dervied.Contructor
```

```csharp
public class Vehical
{
    public Vehical()
    {
        Console.WriteLine("Base.Contructor");
    }
}

public class SportsCar : Vehical
{
    public SportsCar()
    {
        Console.WriteLine("Dervied.Contructor");
    }
}
```
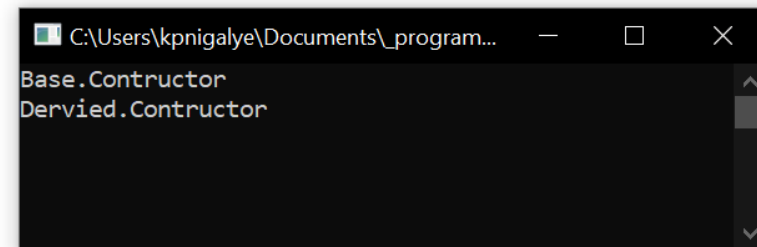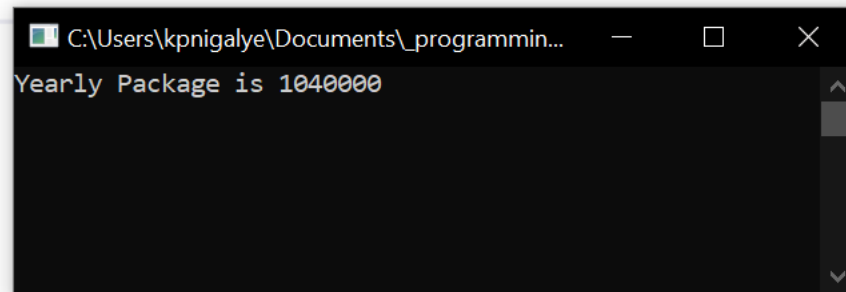
- ➤ In a derived class, if a base class constructor is not called explicitly by using the *base* keyword, the parameterless constructor, if there is one, is called implicitly. Check the example shown below.

```
C:\Users\kpnigalye\Documents\_progra...        —    □    ✕
Base.Contructor
Dervied.Contructor
```

CONSTRUCTOR CALLING ANOTHER CONSTRUCTOR OF THE SAME CLASS

➢ C# allows a constructor of a class to invoke another constructor of the same class using 'this' keyword.

```csharp
public class Employee
{
    public Employee(int weeklySalary, int numberOfWeeks)
        : this(weeklySalary * numberOfWeeks)
    {

    }

    private Employee(int yearlyPackage)
    {
        Console.WriteLine("Yearly Package is {0}", yearlyPackage);
    }
}
```

C:\Users\kpnigalye\Documents_programmin... — □ ✕

Yearly Package is 1040000

COPY CONSTRUCTOR

➢ Copy Constructor is a constructor used to create a new object by copying variables from another object.
➢ C# does not provide copy constructor itself but we can write our own copy constructor.

As you can see, copy constructor take the object of the same class as a parameter and copies radius value of object passed to the constructor to the instance variable of newly created object.

```csharp
public class Circle
{
    protected int _radius;

    public Circle(int radius)
    {
        _radius = radius;
    }

    // Copy Constructor
    public Circle(Circle shape)
    {
        _radius = shape._radius;
    }
}
```
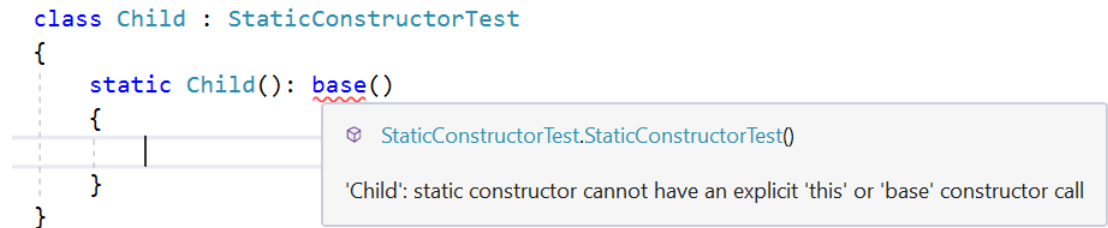
STATIC CONSTRUCTOR

➤ Static Constructor is used to initialize static data or execute actions that only needs to perform once.
➤ Static Constructor cannot accept any parameter i.e. it has to be parameterless.
➤ They are not allowed to have any access modifiers.
➤ A class can have only one static constructor.

```csharp
class StaticConstructorTest
{
    static readonly long _dateTimeTicks;

    static StaticConstructorTest()
    {
        _dateTimeTicks = DateTime.Now.Ticks;
    }
}
```

➤ We cannot use base or this keyword with a static constructor.

```csharp
class Child : StaticConstructorTest
{
    static Child(): base()
    {

    }
}
```

StaticConstructorTest.StaticConstructorTest()

'Child': static constructor cannot have an explicit 'this' or 'base' constructor call

➤ We cannot call the static constructor directly; it is meant to be called by the CLR. It is invoked automatically.
➤ User has no control over when the static constructor is executed in the program.
➤ If you don't provide static constructor to initialize the static variables, all the static fields are initialized to their default values.
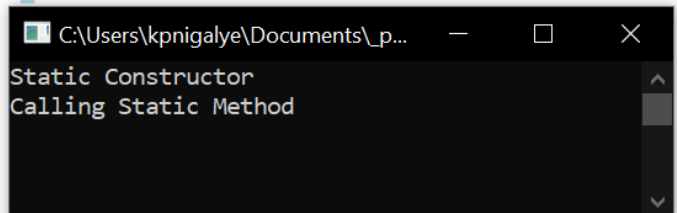
➢ If we have static and instance constructor in our class, static constructor is always executed before the instance constructor.

```csharp
class StaticConstructorTest
{
    static readonly long _dateTimeTicks;

    static StaticConstructorTest()
    {
        _dateTimeTicks = DateTime.Now.Ticks;
        Console.WriteLine("Static Constructor");
    }

    public StaticConstructorTest()
    {
        Console.WriteLine("Instance Constructor");
    }
}
```

```
C:\Users\kpnigalye\Documents\_program...        —    □    ✕
Static Constructor
Instance Constructor
```

```csharp
class StaticConstructorTest
{
    static readonly long _dateTimeTicks;

    static StaticConstructorTest()
    {
        _dateTimeTicks = DateTime.Now.Ticks;
        Console.WriteLine("Static Constructor");
    }

    public StaticConstructorTest()
    {
        Console.WriteLine("Instance Constructor");
    }

    public static void Testing()
    {
        Console.WriteLine("Calling Static Method");
    }
}
```
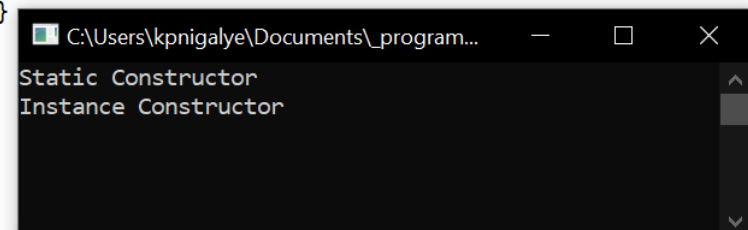
➢ A static constructor is called automatically to initialize the class before the first instance is created or any static members are referenced.

```
C:\Users\kpnigalye\Documents\_p...    —    □    ✕
Static Constructor
Calling Static Method
```

- Static constructors can be used when the class is using a log file and the constructor is used to write entries to this file.
- Static constructors are also useful when creating wrapper classes for unmanaged code.
- Static constructors are also a convenient place to enforce run-time checks on the type parameter that cannot be checked at compile time via constraints.

PRIVATE CONSTRUCTOR

- Private Constructor is a special type of instance constructor.
- It is generally used in classes that has static members only.
- Singleton Pattern makes use of Private constructor.
- If a class has one or more private constructors and no public constructors, other classes (except nested classes) cannot create instances of this class.
- Once you declare a constructor as private, you cannot create an object of this class. See the following code.

```
public class CoffeeMachine
{
    private static int _baseLimit = 100;

    private CoffeeMachine()
    { }
}
```

```
CoffeeMachine machine = new CoffeeMachine();
```

class c_sharp_constructors.CoffeeMachine (+ 1 overload)

'CoffeeMachine.CoffeeMachine()' is inaccessible due to its protection level

But you can have a parameterized constructor and object can be created. Usually coders don't do that but this is just to show you that object can be created of a class which has both parameterized and private constructors.

- Declaration of private constructor prevents generation of parameterless constructor by the compiler. Without the access modifier it is always to be private but it is a good practice to specifically mention using 'private' modifier.
- If we do not have any instance fields or methods in our class definition, we can make our constructor private to avoid instantiation of the class.
- We will get compiler error if we try to inherit the class with private constructor.

```
public class CoffeeMachine
{
    private CoffeeMachine() { }
}

public class PremiumCoffeeMachine: CoffeeMachine
{
    public PremiumCoffeeMachine() { }
}
```

PremiumCoffeeMachine.PremiumCoffeeMachine()

'CoffeeMachine.CoffeeMachine()' is inaccessible due to its protection level

CLASSES WITH PRIVATE CONSTRUCTOR VS ABSTRACT CLASS

A question may come to your mind that we cannot create an instance of a class if there is a private constructor so how it is different from Abstract Class?

➢ If you want to force non-instantiability, you should go for private constructor and if you want users to extend the class functionality you should go for abstract classes.
➢ Private constructors are mostly used in implementing singleton pattern which enables a single object of the class to be shared across the application.
➢ Also, you cannot inherit a class with a private constructor.

```csharp
public class CoffeeMachine
{
    private CoffeeMachine() { }
}

public class PremiumCoffeeMachine: CoffeeMachine
{
    public PremiumCoffeeMachine() { }
}
```

⬡ PremiumCoffeeMachine.PremiumCoffeeMachine()

'CoffeeMachine.CoffeeMachine()' is inaccessible due to its protection level

➢ Abstract classes are meant for inheritance. So, if you want your class functionality to be extended, you have to think of making your class abstract.