Constructors

In this lesson, the world of constructors is explored and you'll learn why they are necessary for creating a class.

WE'LL COVER THE FOLLOWING

What Is a Constructor?

> A constructor is a method that is called to create an instance or object of a class.

As the name suggests, the constructor is used to *construct* the object of a class. It is a special method that outlines the steps that are performed when an instance of a class is created in the program.

> A constructor's **name** must be exactly the **same** as the name of its class.

We have already discussed that to model the *state* of an object we declare member variables or fields in a class. The intention of implementing a constructor is to put an object in a *predictable initial state*, i.e., to initialize some or all the fields of a class to values that can be specified by the user of the class. A constructor **does not have a return type**. Not even `void`! It is a good practice to declare/define it as the first member method.

Let's study the different types of constructors and use them to create class objects.

Default Constructor

The default constructor is the most basic form of a constructor. It is called *default* because when we don't implement a constructor in our class, the compiler, by-default implements one for us at compile time which initializes the fields of that class to their default values. Think of it this way:

> When there is no constructor implemented in a class, a default constructor is always implemented by the compiler automatically. When called, it creates an object in which the fields are initialized to their default values.

This will make sense when we look at the code below. Here, we have our `VendingMachine` class, with its default constructor, and we'll create an object out of it in our `Main()` method:

```
class VendingMachine {

  private bool _onOff;
  private int _count;
  private int _capacity;
  private int _moneyCollected;

  // No constructor implemented

  // A simple print function
  public void PrintFields(){

    Console.WriteLine("Is the machine turned on? {0}", _onOff);
    Console.WriteLine("The count of products is: {0}", _count);
    Console.WriteLine("The capacity of machine is: {0}", _capacity);
    Console.WriteLine("The total money collected till now is: {0}", _moneyCollected);
  }

}

class Demo {

  public static void Main(string[] args) {
    var vendingMachine = new VendingMachine(); // Object created with default constructor!
    vendingMachine.PrintFields();
  }

}
```

> When the above code is executed we get some warnings because we haven't initialized any of the fields ourselves and they will *by-default* have the default values assigned to them.

Parameter-Less Constructor

The parameter-less constructor is pretty much similar to the default constructor. The difference is that we implement it on our own and assign some initial values to the fields. By using the parameter-less constructor we can make sure that whenever an object of a class is created using this constructor its fields should have those initialized values stored in them. Let's understand this using the below code:

```
class VendingMachine {

  private bool _onOff;
  private int _count;
  private int _capacity;
  private int _moneyCollected;

  public VendingMachine() { // A parameter-less construtor implemented
    _onOff = false;
    _capacity = 100;
    _count = 0;
    _moneyCollected = 0;

  }

  // A simple print function
  public void PrintFields(){

    Console.WriteLine("Is the machine turned on? {0}", _onOff);
    Console.WriteLine("The count of products is: {0}", _count);
    Console.WriteLine("The capacity of machine is: {0}", _capacity);
    Console.WriteLine("The total money collected till now is: {0}", _moneyCollected);
  }

}

class Demo {

  public static void Main(string[] args) {
    var vendingMachine = new VendingMachine(); // Object created with parameter-less construc
    vendingMachine.PrintFields();
  }

}
```

Wouldn't a machine, when just installed, in its very initial state have these values? There are no products placed in the machine, nor is there any money collected yet. The machine is not functional, so it's turned off. However, the machine has a capacity of storing 100 products in it.

Parameterized Constructor

Moving one step further, suppose we have placed some products in the machine and want to initialize it to that state. We also want to turn on the machine. For this, we need a bit more freedom while creating a machine's object. With default or parameter-less constructors, all objects of a class start out in the same state. Parameterized constructors allow us to create objects of the same class in different initial states. Hence, the parameterized constructor will provide us this freedom.

> In a parameterized constructor, we pass arguments to the constructor and use these argument values to set the initial values of the object fields.

We are basically overloading the default constructor to accommodate our preferred values for the fields.

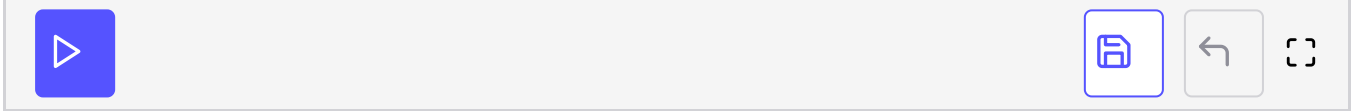
> We can also implement logical checks before initializing field values in a constructor so that invalid values are not inadvertently assigned.

It is a software engineering principle not to assume that the user will provide safe and sane values to the arguments. Always assume the worst and try to write safe code.

Let's try it out:

```
class VendingMachine {

  private bool _onOff;
  private int _count;
  private int _capacity;
```

```
  private int _moneyCollected;

  public VendingMachine() { // A parameter-less constructor implemented

    _onOff = false;
    _capacity = 100;
    _count = 0;
    _moneyCollected = 0;

  }

  public VendingMachine(bool onOff , int count, int moneyCollected) { // A parameterized cons
    _onOff = onOff;
    _capacity = 100;
    _moneyCollected = moneyCollected;
    if (count >= 0 && count <= 100) { //Count should always be lesser than or equal to capaci
      _count = count;
    }
    else Console.WriteLine("Invalid count vlaue!");


  }

  // A simple print function
  public void PrintFields(){

    Console.WriteLine("Is the machine turned on? {0}", _onOff);
    Console.WriteLine("The count of products is: {0}", _count);
    Console.WriteLine("The capacity of machine is: {0}", _capacity);
    Console.WriteLine("The total money collected till now is: {0} \n", _moneyCollected);
  }

}

class Demo {

  public static void Main(string[] args) {
    //passing the parameters:
    var vendingMachine = new VendingMachine(true,50,10); // Object created with parameterized
    vendingMachine.PrintFields();
    //invalid initialization:
    var vendy = new VendingMachine(true,-10,10);
    vendy.PrintFields();
  }

}
```



This is much more convenient than the default constructor! We can notice how easily we have initialized the fields of the class to put the object in the desired state using the parameterized constructor.

In the next lesson, we will learn a bit more about constructors.