Creating an ArrayList Object

In this lesson, you'll learn how to create an ArrayList object in Java.

We'll cover the following

Let's practically use an *ArrayList Object* in our code. As we already know, the `ArrayList` is a class in Java and to use a class's functionality we have to *instantiate* it i.e. to construct its *object*.
To construct an `object` of a `class` we use a constructor. Before we jump to the constructor part let's discuss a bit about the `wrapper classes`.

Wrapper classes

In Java, we cannot *directly* instantiate an **ArrayList** of the primitive data types like `int, char, boolean`. The reason for this is that primitive data types are not *objects*. For this purpose, Java has inbuilt `wrapper classes` which just *wrap* these primitive data types in a `class`. Below given is the list of *inbuilt wrapper classes*:

Primitive Data Type	Wrapper Class
`boolean`	`Boolean`
`byte`	`Byte`
`char`	`Character`
`double`	`Double`

`float`	`Float`
`int`	`Integer`
`long`	`Long`
`short`	`Short`

> **Note:** *`String`* is not a primitive data type. These are objects instantiated from a `String` base class in Java, so they **don't need** any wrapper class.

Generics in action

The above wrapper classes are passed to the ArrayList class as *type parameters* inside the *pair* of angle brackets `<>`. This is exactly the concept of Generics. Syntactically we can generalize the declaration of an ArrayList as:

```
ArrayList<Type> name;
```

Instantiation

The `ArrayList` objects can be instantiated using the *keyword* `new` and the ArrayList `class` are **three** types of constructors which are discussed below:

Empty ArrayList

The following is the basic and the most used way to instantiate an ArrayList of `Integer` data type.

```
ArrayList<Integer> myarrList =new ArrayList<>();
```

Instantiation of an Empty Integer ArrayList

The above line of code will instantiate an empty `Integer` ArrayList. The array on which this ArrayList is based has a `length` of `10` by-default at the time of instantiation and this size grows dynamically (*during runtime*) according to the required number of memory locations to store the elements.

ArrayList with initially defined Capacity

ArrayList with Initially defined Capacity

We can instantiate an ArrayList with an initially defined capacity to ensure the space allocation at the time of instantiation.

> The **capacity** is the number of elements the list can potentially accommodate without reallocating its internal structures.

Let’s instantiate an ArrayList of `Char` using the respective *Wrapper Class* with an initial capacity of `20` elements.

```
ArrayList<Character> chArrList = new ArrayList<Character>(20);
```

Instantiation of an ArrayList with Initial Capacity = 20

We can notice that the concept of constructor overloading is being implemented in the above line of code i.e. rather than calling an *empty* constructor, we are passing *initial capacity = 20* to it.

> The above declared ArrayList will reallocate its resources and grow dynamically once it runs out of these **20** spaces.

In the next lesson, let’s check out how ArrayLists come in handy by using their in-built methods.