Class, Object and Properties in Kotlin

In this tutorial, you will learn class, object and property in Kotlin. Topics like class and object are not new for you but property is a new concept introduced by Kotlin. Let's explore them in detail.

Class in Kotlin

You are already familiar with the definition of a class which is "a class is like a blueprint using which we can create objects." It is a way to achieve two important principles of object-oriented programming which are- **encapsulation and abstraction**. In Kotlin, a class can have properties, functions, and inner class.

Syntax of a class

```
visibility-modifier class class_name{
   properties
   functions
}
```

Example of a class

```
class Student(n: String){
   val name: String = n
   fun show(){
      println(name)
   }
}
```

In the above example, we have created a class named `Student` that has two members- `name` and `show()` function.
`(n: String)` is a primary constructor. Don't worry if you are seeing constructor like this for the first time, constructor is explained in detail in the next chapter.

Object in Kotlin

The object is an instance of a class. If you are a Java developer, then you might think that `new` keyword is used to instantiate a class. But in Kotlin, there is no `new` keyword. To create an instance of a class, call the constructor as a regular function.

You can access members of a class with the help of dot(.) operator.

Syntax of object

```
val objectname = ClassName(parameters)
OR
var objectname = ClassName(parameters)
```

Example of object

```
class Employee(n: String){
   val name: String = n
   fun show(){
      println(name)
   }
}
fun main(args: Array<String>){
   val s1 = Employee("Govind")
   var s2 = Employee("Keshav")
   println("First Employee- ${s1.name}")
   println("Second Employee- ${s2.name}")
}
```

Output

```
First Employee- Govind
Second Employee- Keshav
```

Properties in a class

Fields or variables defined in a class are used to store data. To access that data, you use getter and setter which are usually called accessor methods.

In Kotlin, combination of the field and its accessors is known as property. You declare a property in the same way you declare a variable. Declaring a property with `val` creates a read-only property. And declaring a property with `var` creates a mutable property.

Syntax of property in a class

```
visibility-modifier val propertyname: BasicType //This is a read-only
property
OR
visibility-modifier var propertyname: BasicType //This is a mutable
property
```

In Kotlin, when you declare a property, accessor methods are provided to you automatically. If you have declared a read-only property then field and getter are provided and if you have declared a mutable property then field, getter and setter are provided. Let's understand it with the help of an example-

Example of property in a class

```
class Student(i: Int, n: String){
   var name: String = n
   val id: Int = i
}
fun main(args: Array<String>){
   val s = Student(10, "Java")
   s.id = 20 //It is similar to s.setId()
   println("Name- ${s.id}") //It is similar to s.getId()
   s.name = "Kotlin" //It is similar to s.setName()
   println("Name- ${s.name}") //It is similar to s.getName()
}
```

In the above example, name is a mutable property. Instead of writing `s.getName()` to access getter, simply by writing `s.name` you can access getter of name property. By writing `s.name = "Kotlin"` , you can access setter of name property, no need to call `s.setName("Kotlin")` .
id is a read-only property. Rather than writing `s.getId()` to access getter of id property. Simply, by writing `s.id` you can access getter. Since, it is a read-only property, you are only provided with getter and writing `s.id = 20` will give compile-time error. You must comment it to run the program.