Kotlin - Generics

Like Java, Kotlin provides higher order of variable typing called as Generics. In this chapter, we will learn how Kotlin implements Generics and how as a developer we can use those functionalities provided inside the generics library. Implementation wise, generics is pretty similar to Java but Kotlin developer has introduced two new keywords **“out”** and **“in”** to make Kotlin codes more readable and easy for the developer.

In Kotlin, a class and a type are totally different concepts. As per the example, List is a class in Kotlin, whereas List<String> is a type in Kotlin. The following example depicts how generics is implemented in Kotlin.

```
fun main(args: Array<String>) {
   val integer: Int = 1
   val number: Number = integer
   print(number)
}
```

In the above code, we have declared one “integer” and later we have assigned that variable to a number variable. This is possible because “Int” is a subclass of Number class, hence the type conversion happens automatically at runtime and produces the output as “1”.

Let us learn something more about generics in Kotlin. It is better to go for generic data type whenever we are not sure about the data type we are going to use in the application. Generally, in Kotlin generics is defined by **<T>** where “T” stands for template, which can be determined dynamically by Kotlin complier. In the following example, we will see how to use generic data types in Kotlin programming language.

Live Demo

```
fun main(args: Array<String>) {
   var objet = genericsExample<String>("JAVA")
   var objet1 = genericsExample<Int>(10)
}
class genericsExample<T>(input:T) {
   init {
      println("I am getting called with the value "+input)
   }
}
```

In the above piece of code, we are creating one class with generic return type, which is represented as **<T>**. Take a look at the main method, where we have dynamically defined its value at the run by proving the value type, while creating the object of this class. This is how generics is interpreted by Kotlin compiler. We will get the following output in the browser, once we run this code in our coding ground.

```
I am getting called with the value JAVA
I am getting called with the value 10
```

When we want to assign the generic type to any of its super type, then we need to use “out” keyword, and when we want to assign the generic type to any of its sub-type, then we need to use “in” keyword. In the following example, we will use “out” keyword. Similarly, you can try using “in” keyword.

Live Demo

```
fun main(args: Array<String>) {
   var objet1 = genericsExample<Int>(10)
   var object2 = genericsExample<Double>(10.00)
   println(objet1)
   println(object2)
}
class genericsExample<out T>(input:T) {
   init {
      println("I am getting called with the value "+input)
   }
}
```

The above code will yield the following output in the browser.

```
I am getting called with the value 10
I am getting called with the value 10.0
genericsExample@28d93b30
genericsExample@1b6d3586
```