IT2010 – Mobile Application Development
BSc (Hons) in Information Technology
2nd Year
Faculty of Computing
SLIIT
2025 – Tutorial
OOP Basics

Classes

In kotlin classes are mentioned using the keyword class:

class Bird {/*……*/}

and specify the name of the class:

eg :- create a Bird class along with some properties (type and color)

```
class Bird {
        var type = " "
        var color = " "
    }
```

A property is basically a variable that belongs to the class.

Object

We use the class named Bird to create objects. We create an object of Bird called b1, and then we access the properties of b1 by using the dot syntax (.)

eg :-

```
// create a b1 object of the Bird class
    val b1 = Bird()
   // Access the properties and add some values to it
    b1.type = "Parrot"
    b1.color = "Green"
    println(b1.type)  //outputs Parrot
    println(b1.color) //outputs Green
```

Constructors

In kotlin, a class can have a primary constructor and one or more secondary constructors. The primary constructor is a part of the class header, and it goes after the class name and optional type parameters.

Constructor is like a special function, and it is defined by using two parantheses () after the class name.

eg :-

```
class Bird (var type : String , var color : String)
        fun main() {
                val b1 = Bird (“Parrot” , “Green”)
                println(b1.type)
                println(b1.color)
}
```

Inheritance

In kotlin, it is possible to inherit class properties and functions from one class to another. We mentioned the inheritance concept into two categories:

- Subclass (child) – the class that inherits from another class
- Superclass (parent) – the class being inherited from

eg :-

```
// superclass
        open class Bird {
                var color = “Green”
         }
         //subclass
         Class Parrot:Bird() {
                  Fun myFunction() {
                           println(color) // color is defined in the superclass
                  }
         }
         //create an object of the Parrot and call myFunction
         fun main () {
                  var myObj = Parrot ()
                   myObj.myFunction ()
          }
```

Interfaces

With the interface, you can define a set of properties and methods, that the concrete types must follow and implement.

eg :-

```
interface Vehicle {
	fun start()
	fun stop()
}

class Car : Vehicle {
	override fun start()
	{
		println("Car started")
	}

	override fun stop()
	{
		println("Car stopped")
	}
}

fun main()
{
	val obj = Car()
	obj.start()
	obj.stop()
}
```

Properties

It is the combination of accessories and the fields in the case of Java. In the case of Kotlin, properties are meant to be a first-class language feature. These features replace fields and accessor methods. A **property** in a class is declared the same as declaring a variable with ***val*** and ***var*** keywords. A property declared as var is mutable and thus, can be changed.

```
class Abc(
	val name: String,
	val ispassed : Boolean
)
```

Readable Property: Generates a field and a trivial getter
Writable Property: A getter, a setter, and a field

Basically, what happens is that the declaration of the property declares the related accessors (both setter and getter for writable and getter for readable property). A field stores the value.

```
class Abc(
        val name: String,
        val ispassed : Boolean
)

fun main(args: Array<String>) {

        val abc = Abc("Bob",true)
        println(abc.name)
        println(abc.ispassed)

        /*
        In Java
        Abc abc = new Abc("Bob",true);
        System.out.println(person.getName());
        System.out.println(person.isMarried());

        */
}
```

Output

Bob

true

Sealed classes

Sealed class is a class which restricts the class hierarchy. A class can be declared as sealed class using "sealed" keyword before the class name. It is used to represent restricted class hierarchy.

Sealed class is used when the object has one of the types from limited set, but cannot have any other type.

The constructors of sealed classes are private in default and cannot be allowed as non-private.

Declaration of sealed class

```
sealed class MyClass
        sealed class Shape{
            class Circle(var radius: Float): Shape()
```

```
        class Square(var length: Int): Shape()
        class Rectangle(var length: Int, var breadth: Int): Shape()
    object NotAShape : Shape()
}
```

Enum classes

Kotlin enums are classes, which means that they can have one or more constructors. Thus you can initialize enum constants by passing the values required to one of the valid constructors. This is possible because enum constants are nothing other than instances of the enum class itself.

```
enum class Day(val dayOfWeek: Int) {
    MONDAY(1),
    TUESDAY(2),
    WEDNESDAY(3),
    THURSDAY(4),
    FRIDAY(5),
    SATURDAY(6),
    SUNDAY(7)
}
```

Activity

1. Try this:

```
val names = arrayOf("Jane","Ted","Alice","John",1,2,3,4,34.223,3L,23.1f)

val numbers = intArrayOf(3,4,3,2,1,5,6,7,3)

fun main() {
    for (i in numbers){
        println(i)
    }
}
```

2. Create a Car class along with some properties (brand, model and year)
3. Create a c1 object of the Car class