Advanced Functional Thinking

Dr Surbhi Saraswat
School of Computer Science

SCALA CLASSES AND OBJECTS

- Classes:
  - A class is a user-defined blueprint or prototype from which objects are created.
  - A class combines the fields and methods(member function which defines actions) into a single unit.
  - In Scala, a class declaration contains the class keyword, followed by an identifier(name) of the class.

```
class Class_name{
// methods and fields
}
```

```
class Student{
   var id:Int = 0;
   var name:String = null;
}
object MainObject{
   def main(args:Array[String]){
      var s = new Student()
      println(s.id+" "+s.name);
   }
}
```

SCALA CLASSES AND OBJECTS

- Objects:
  - It is a basic unit of Object Oriented Programming and represents the real-life entities.
  - A typical Scala program creates many objects.
  - An object consists of :
    - **State:** It is represented by attributes of an object. It also reflects the properties of an object.
    - **Behavior:** It is represented by methods of an object. It also reflects the response of an object with other objects.
    - **Identity:** It gives a unique name to an object and enables one object to interact with other objects.
- Once you define a class, you can create objects from the class blueprint with the keyword **new**.

SCALA CLASSES AND OBJECTS

- Objects:
  - Once you define a class, you can create objects from the class blueprint with the keyword **new**.

```
class Dog(name:String, breed:String, age:Int, color:String )
{
        println("My name is:" + name + " my breed is:" + breed);
        println("I am: " + age + " and my color is :" + color);

}
object Main
{

    // Main method
    def main(args: Array[String])
    {

        // Class object
        var obj = new Dog("tuffy", "papillon", 5, "white");
    }
}
```

```
My name is:tuffy my breed is:papillon
I am: 5 and my color is :white
```

SCALA CLASSES AND OBJECTS

- Following is a simple syntax to define a class in Scala:

```
class Point(xc: Int, yc: Int) {
    var x: Int = xc
    var y: Int = yc
    def move(dx: Int, dy: Int) {
    x = x + dx
    y = y + dy
    println ("Point x location : " + x);
    println ("Point y location : " + y);
        }
}
```

- This class defines two variables **x** and **y** and a method: **move,** which does not return a value. Class variables are called fields of the class and methods are called class methods.
- The class name works as a class constructor, which can take a number of parameters. The above code defines two constructor arguments, **xc** and **yc**; they are both visible in the whole body of the class.

SCALA CLASSES AND OBJECTS

- Constructors:
  - Scala provides primary and any number of auxiliary constructors.
  - In scala, if you don't specify primary constructor, compiler creates a constructor which is known as primary constructor.
  - All the statements of class body treated as part of constructor. It is also known as default constructor.

```
class Student{
println("Hello from default constructor");
}
```

```
Hello from default constructor
```

SCALA CLASSES AND OBJECTS

You can create objects using a keyword **new** and then you can access class fields and methods as shown below in the example:

```
class Student(id:Int, name:String){     // Primary constructor
   def show(){
      println(id+" "+name)
   }
}
object MainObject{
   def main(args:Array[String]){
      var s = new Student(100,"Martin")   // Passing values to constructor
      s.show()              // Calling a function by using an object
   }
}
```

SCALA CLASSES AND OBJECTS

- Auxiliary Constructors:
  - In scala, you can define any number of auxiliary constructors.
  - You must call primary constructor or any previous constructor inside the auxiliary constructor.
  - For this you have to use **this** keyword.
  - When calling other constructor make it first line in your constructor.

```
101 Rama 20
```

```
class Student(id:Int, name:String){
   var age:Int = 0
   def showDetails(){
      println(id+" "+name+" "+age)
   }
   def this(id:Int, name:String,age:Int){
      this(id,name)       // Calling primary constructor, and it is first line
      this.age = age
   }
}

object MainObject{
   def main(args:Array[String]){
      var s = new Student(101,"Rama",20);
      s.showDetails()
   }
}
```

SCALA CLASSES AND OBJECTS

- Constructor overloading:
  - In scala, you can overload constructor. Let's see an example.

```
101
100
100 India
```

```
class Student(id:Int){
    def this(id:Int, name:String)={
        this(id)
        println(id+" "+name)
    }
    println(id)
}

object MainObject{
    def main(args:Array[String]){
        new Student(101)
        new Student(100,"India")
    }
}
```

SCALA CLASSES AND OBJECTS

- Polymorphism:
  - It is the ability of any data to be processed in more than one form.
  - Scala implements polymorphism through virtual functions, overloaded functions and overloaded operators.
  - In Scala, the function can be applied to arguments of many types, or the type can have instances of many types.

```
First Execution:120
Second Execution:120
Third Execution:300
```

```
class example
{

    // This is the first function with the name fun
    def func(a:Int)
    {
        println("First Execution:" + a);
    }

    // This is the second function with the name fun
    def func(a:Int, b:Int)
    {
        var sum = a + b;
        println("Second Execution:" + sum);
    }

    // This is the first function with the name fun
    def func(a:Int, b:Int, c:Int)
    {
        var product = a * b * c;
        println("Third Execution:" + product);
    }
}

// Creating object
object Main
{
    // Main method
    def main(args: Array[String])
    {
        // Creating object of example class
        var ob = new example();
        ob.func(120);
        ob.func(50, 70);
        ob.func(10, 5, 6);
    }
```

SCALA CLASSES AND OBJECTS

- **Inheritance** is an important pillar of OOP(Object Oriented Programming). It is the mechanism in Scala by which one class is allowed to inherit the features(fields and methods) of another class.
- Important terminology:
- Super Class: The class whose features are inherited is known as superclass(or a base class or a parent class).
- Sub Class: The class that inherits the other class is known as subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- Reusability: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

SCALA CLASSES AND OBJECTS

- The keyword used for inheritance is extends.

```
class Employee{
   var salary:Float = 10000
}

class Programmer extends Employee{
   var bonus:Int = 5000
   println("Salary = "+salary)
   println("Bonus = "+bonus)
}

object MainObject{
   def main(args:Array[String]){
      new Programmer()
   }
}
```

Difference between method overloading and overridding

Different types of inheritence

Differenece between method and field overloading

Public and private

SCALA CLASSES AND OBJECTS

- Scala Method Overriding

```
class Vehicle{
   def run(){
      println("vehicle is running")
   }
}

class Bike extends Vehicle{
    override def run(){
      println("Bike is running")
   }
}

object MainObject{
   def main(args:Array[String]){
      var b = new Bike()
      b.run()
   }
}
```

- Scala Field Overriding

```
class Vehicle{
     val speed:Int = 60

}
class Bike extends Vehicle{
   override val speed:Int = 100     // Override keyword
    def show(){
      println(speed)
    }
}
object MainObject{
    def main(args:Array[String]){
      var b = new Bike()
      b.show()
    }
}
```