Session 17:
SCALA BASICS 4
Assignment 1

- Prachi Mohite

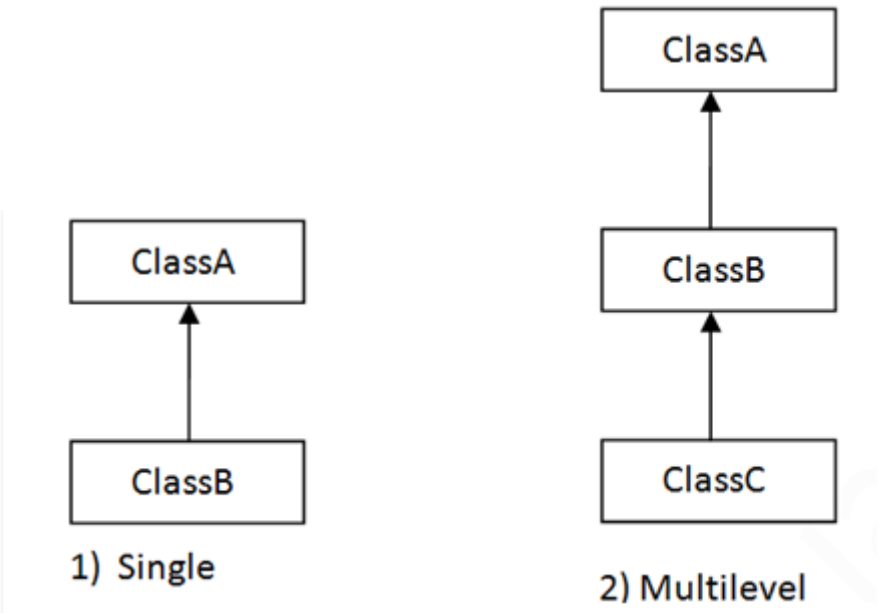
Inheritance

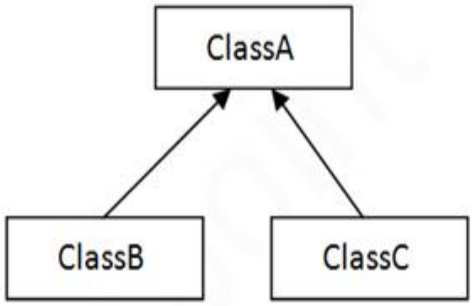
Inheritance is an object oriented concept which is used to reusability of code. You can achieve inheritance by using extends keyword. To achieve inheritance a class must extend to other class. A class which is extended called super or parent class. a class which extends class is called derived or base class.

```
class SubClassName extends SuperClassName(){
/* Write your code
*  methods and fields etc.
*/
}
```

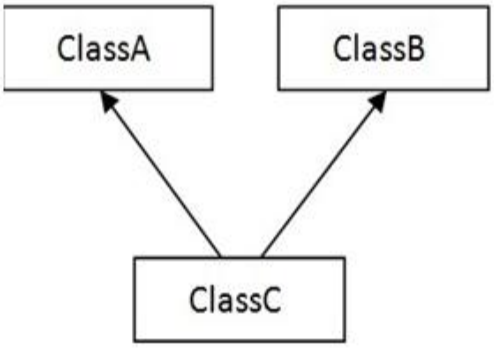
Types of Inheritance in Scala

Scala supports various types of inheritance including single, multilevel, multiple, and hybrid. You can use single, multilevel and hierarchal in your class. Multiple and hybrid can only be achieved by using traits. Here, we are representing all types of inheritance by using pictorial form.

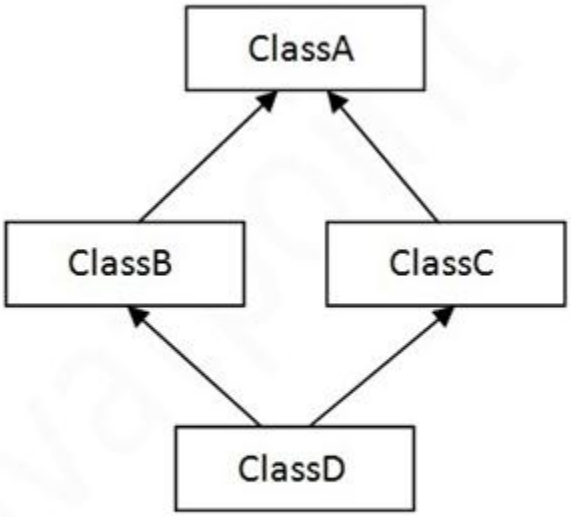




3) Hierarchical



4) Multiple



5) Hybrid

Traits in Scala

A trait encapsulates method and field definitions, which can then be reused by mixing them into classes. Unlike class inheritance, in which each class must inherit from just one superclass, a class can mix in any number of traits.

Traits are used to define object types by specifying the signature of the supported methods. Scala also allows traits to be partially implemented but traits may not have constructor parameters.

A trait definition looks just like a class definition except that it uses the keyword trait. The following is the basic example syntax of trait.

Syntax

```
trait Equal {
   def isEqual(x: Any): Boolean
   def isNotEqual(x: Any): Boolean = !isEqual(x)
}
```

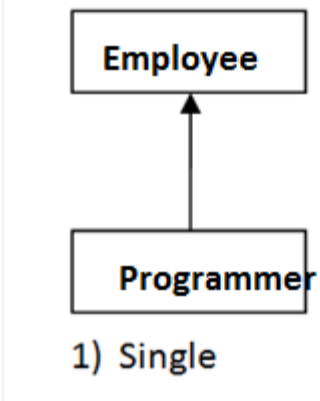
- If it might be reused in multiple, unrelated classes, make it a trait. Only traits can be mixed into different parts of the class hierarchy.

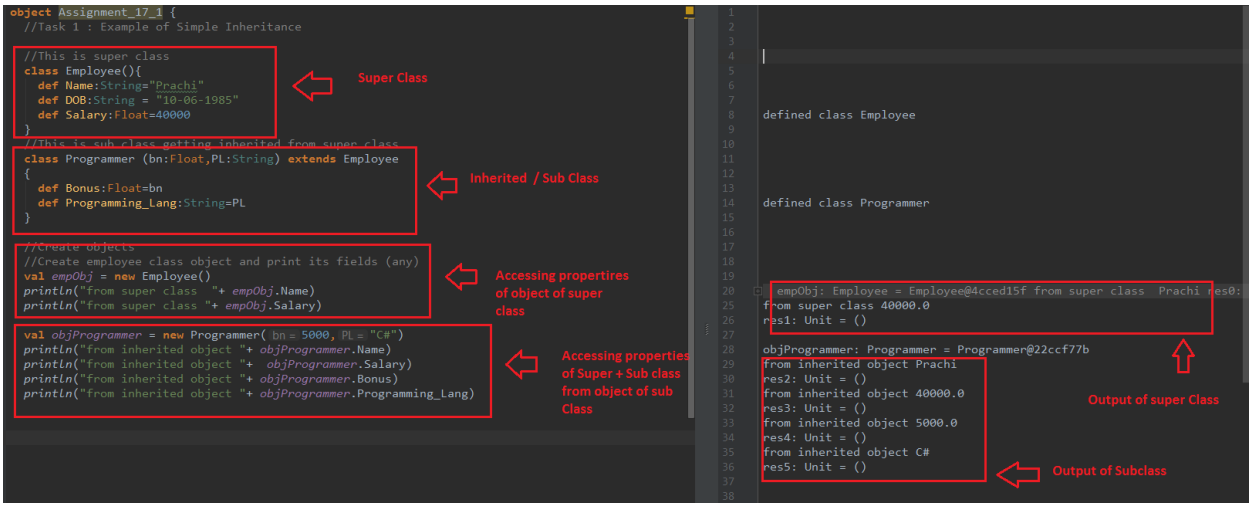
Task 1

Write a simple program to show inheritance in scala.

Simple Inheritance can be achieved using classes in scala
Here to understand simple inheritance we will take example of two classes

- Employee Class
  - It is Super Class as any one working any company is employee of that company with some common set of details
  - Add fields like Name, DOB , Salary
- Programmer Class
  - This can be inherited from Employee class with having some addition details, as a programmer is always employee of a company.
  - Add additional fields like Bonus , Programming_Language





Task 2

Write a simple program to show multiple inheritance in scala

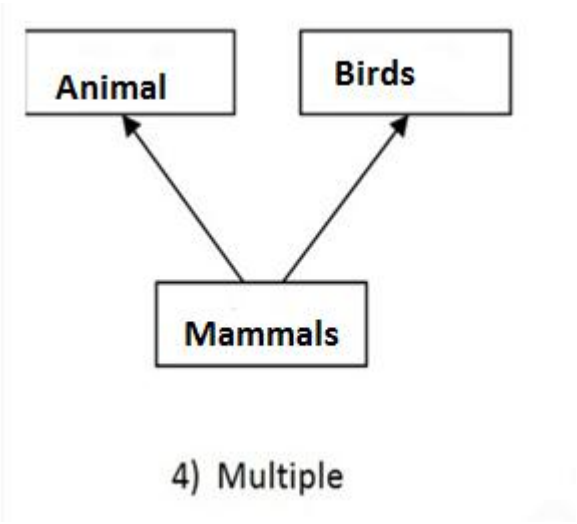
Multiple Inheritance

- Multiple Inheritance is implemented in scala by using Traits.

To Execute the multiple inheritance we have created two traits

  - Birds
  - Animals

And created a class named as Mammals which is extended from both the traits i.e. Multiple Inheritance as shown in below figure.



Code and its Output

```
object Assignment_17_1 {

  //first trait
  trait Birds
  {
    def sound:String="Flying"
    def Work() :String = {"Flying, Flying"}
  }
  //Second trait
  trait Animal
  {
    def Type:String="Pet"
    def IsBird(): String={
    if(Type=="Pet") "Bird" else "Animal"
    }
  }
 //Created a class which is extending the both the traits
  class Mammal extends Birds with Animal
  {
    //Here we have overridden the type from Animal Class
    override def Type: String = "Dog"
  }

  val objMammal = new Mammal()
  //Below function will call where Type = "Dog"
  val result=objMammal.IsBird()
  val result1 = objMammal.Work()

  println(result)
  println(result1)
}
```

Trait one

Trait 2

Class extending trait 1 and trait 2 --> multiple inheritance

```
defined trait Birds

defined trait Animal

defined class Mammal

objMammal: Mammal = Mammal@15bfd491

result: String = Animal
result1: String = Flying, Flying

Animal
res0: Unit = ()
Flying, Flying
res1: Unit = ()
```

Output

Task 3

Write a partial function to add three numbers in which one number is constant and two numbers can be passed as inputs and define another method which can take the partial function as input and squares the result.

Partial Functions

A *partial function* is a function that does not provide an answer for every possible input value it can be given. It provides an answer only for a subset of possible data, and defines the data it can handle. In Scala, a partial function can also be queried to determine if it can handle a particular value.

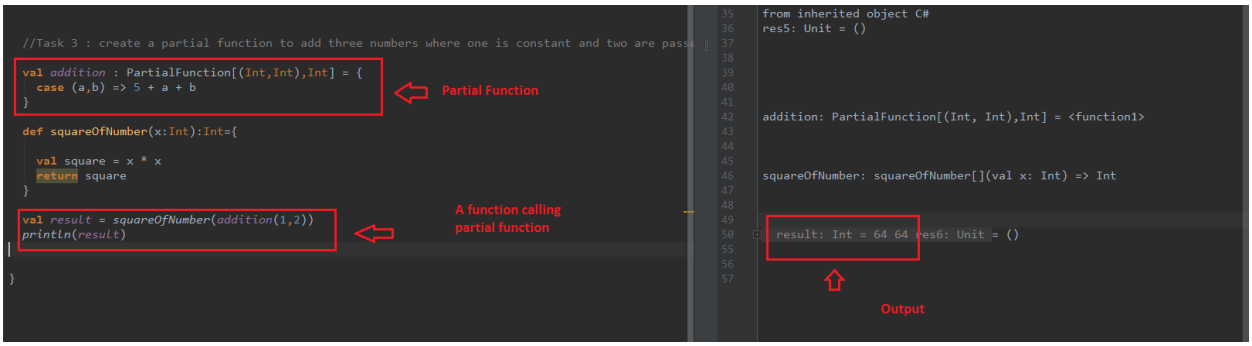
Ways to define Scala Partial Functions

- Using case statements
- Using Else, orElse
- Using Collect method

In our Example we will be using Case Method for creating the Partial Function

Higher Order Functions

Scala allows the definition of higher-order functions. These are functions that take other functions as parameters, or whose result is a function.



Task 4

Write a program to print the prices of 4 courses of Acadgild:
Android App Development -14,999 INR
Data Science - 49,999 INR
Big Data Hadoop & Spark Developer – 24,999 INR
Blockchain Certification – 49,999 INR
using match and add a default condition if the user enters any other course.

Solution Approach

We will be using case statement to get the desired result.

Case Statement / Match Expression in Scala

Match expressions are similar to “switch” statements of C and Java, in which a single input item is checked and the first pattern that is “matched” is executed and its value returned. Like “switch” statement of C and Java, match expressions in Scala support a default “catch-all” pattern. Unlike them, in match expressions only zero or one patterns can match. There is no break statement or “fall-through” from one pattern to the next one in line that would prevent this fall-through.

```
<expression> match {
case <pattern match> => <expression>
[case...]
}
```

Match expression in function body for different type value:

```
def getClassAsString(x: Any):String = x match {
case s: String => s + " is a String"
case i: Int => "Int"
case f: Float => "Float"
case l: List[_] => "List"
case p: Person => "Person"
case _ => "Unknown"
}
```

use ‘if’ expressions in case statements

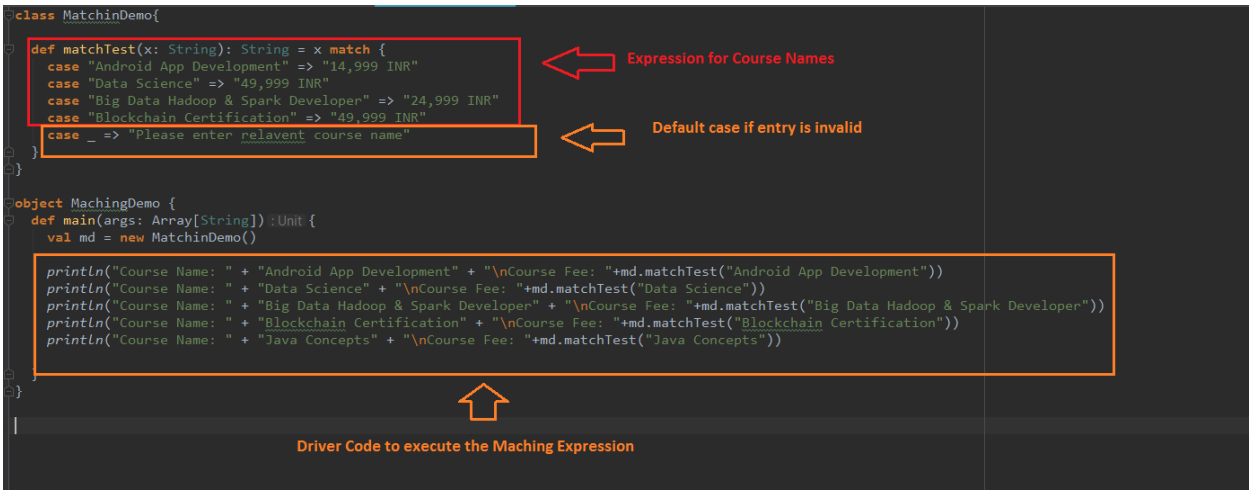
```
i match {
case a if 0 to 9 contains a => println("0-9 range: " + a)
case b if 10 to 19 contains b => println("10-19 range: " + a)
case c if 20 to 29 contains c => println("20-29 range: " + a)
```

```
    case _ => println("Hmmm...")
    }
```

reference class fields in your 'if' statements:

```
    stock match {
    case x if (x.symbol == "XYZ" && x.price < 20) => buy(x)
    case x if (x.symbol == "XYZ" && x.price > 50) => sell(x)
    case x => doNothing(x)
    }
```

Code for getting respective fees from the course name



Output of the above code