# ASSIGNMENT 15.1

## Problem Statement:

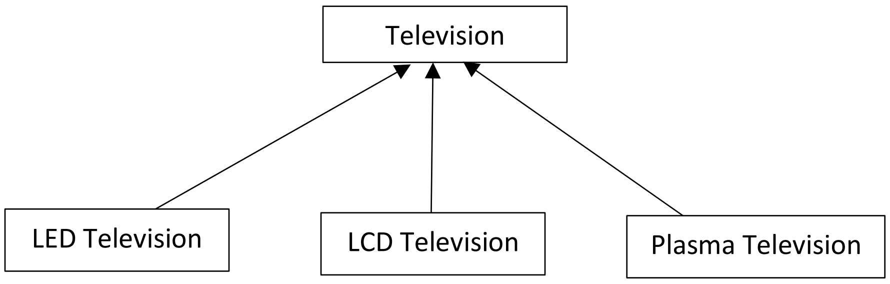
1. Write a simple program to show inheritance in Scala.
2. Write a simple program to show multiple inheritance in Scala.

## Solution:

1. Write a simple program to show inheritance in Scala.

**Inheritance** is a concept of object oriented programming that allows a class to inherit properties and methods of another existing class. The class that inherits from other class is known as **sub class** or **derived class** whereas another class is known as **super class** or **base class**. The main advantage of inheritance is that it provides **code reusability**.

Here is an example code I have written in Scala to understand the concept better.



**Television.scala**

```
abstract class Television(modelName: String) {                    // base class with two strings variables
        def modelWithType = modelName + " is a " + tvType + " TV"
        def tvType: String
}
class LEDTelevision(modelName: String) extends Television(modelName) {
        override def tvType = "LED"                        // subclass LEDTelevison overriding value of
  }                                                        // tvType variable
```

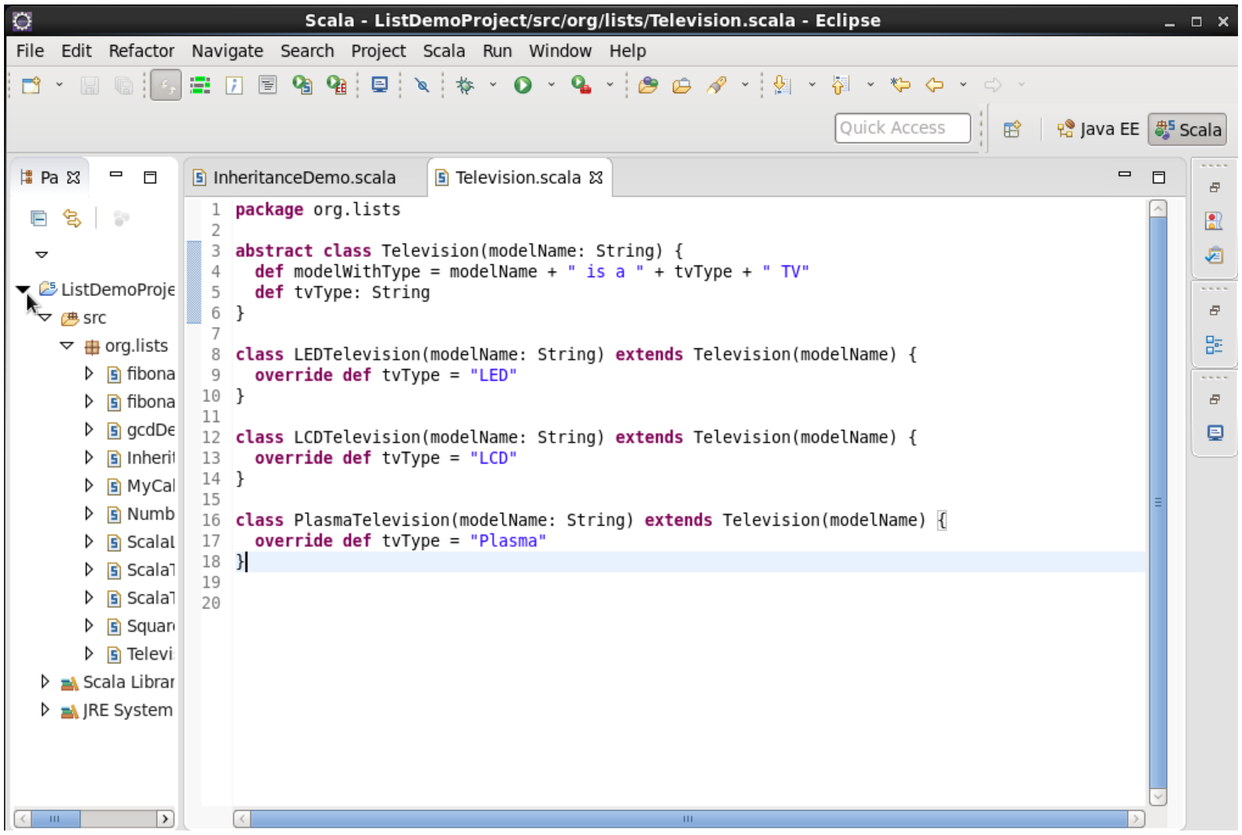
```
class LCDTelevision(modelName: String) extends Television(modelName) {
        override def tvType = "LCD"                    // subclass LCDTelevison overriding value of
  }                                                    // tvType variable
class PlasmaTelevision(modelName: String) extends Television(modelName) {
        override def tvType = "Plasma"                 // subclass PlasmaTelevison overriding value
  }                                                    // of tvType variable
```

Here we have a super class named ‘Television’ which has two properties, tvType and modelWithType that forms a string consisting of type of the television along with television model name taken as input via constructor.
The subclasses: LEDTelevsion, LCDTelevision as well as PlasmaTelevision inherits the members of base class, Television. Subclasses can override the members of super class. In the above example, the property ‘tvType’ has been overridden based on respective class names.

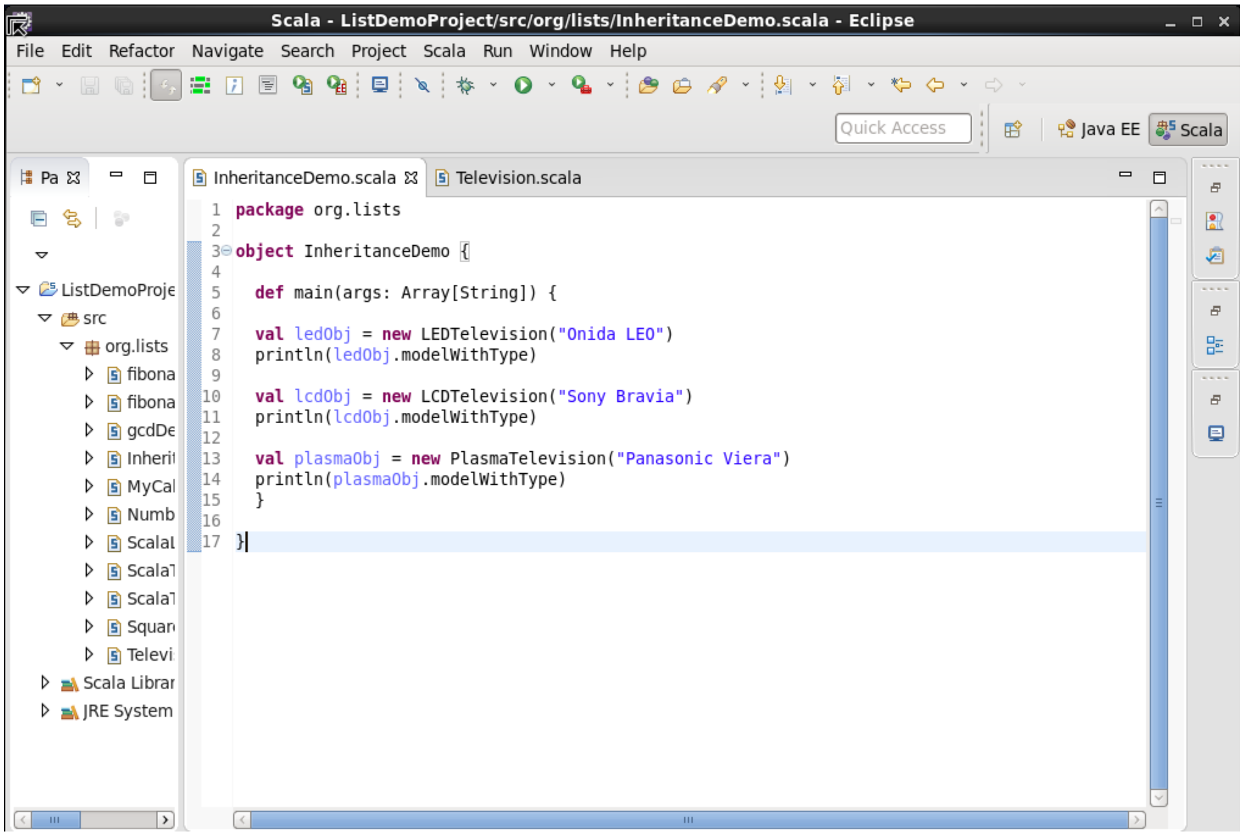
Let’s create objects for each of these sub classes and try to print the model name along with type of television:

**InheritanceDemo.scala**

```
object InheritanceDemo {
  def main(args: Array[String]) {
        val ledObj = new LEDTelevision("Onida LEO")       // create object of LEDTelevision
        println(ledObj. modelWithType)                    // print model and type of television

        val lcdObj = new LCDTelevision("Sony Bravia")     // create object of LCDTelevision
        println(lcdObj. modelWithType)                    // print model and type of television

        val plasmaObj = new PlasmaTelevision("Panasonic Viera") // create object: PlasmaTelevision
        println(plasmaObj. modelWithType)                 // print model and type of television
  }
}
```

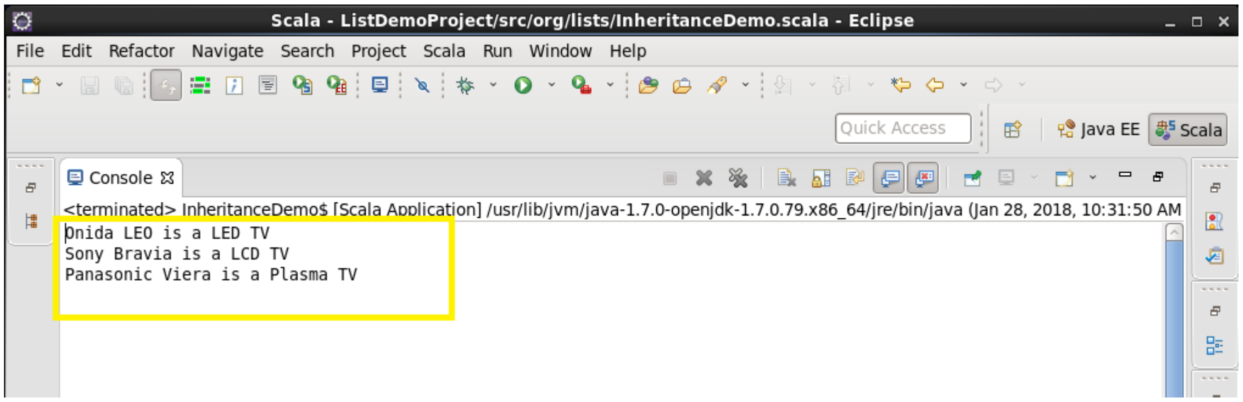

Scala - ListDemoProject/src/org/lists/Television.scala - Eclipse

```
package org.lists

abstract class Television(modelName: String) {
  def modelWithType = modelName + " is a " + tvType + " TV"
  def tvType: String
}

class LEDTelevision(modelName: String) extends Television(modelName) {
  override def tvType = "LED"
}

class LCDTelevision(modelName: String) extends Television(modelName) {
  override def tvType = "LCD"
}

class PlasmaTelevision(modelName: String) extends Television(modelName) {
  override def tvType = "Plasma"
}
```


Scala - ListDemoProject/src/org/lists/InheritanceDemo.scala - Eclipse

```
package org.lists

object InheritanceDemo {

  def main(args: Array[String]) {

  val ledObj  = new LEDTelevision("Onida LEO")
  println(ledObj.modelWithType)

  val lcdObj  = new LCDTelevision("Sony Bravia")
  println(lcdObj.modelWithType)

  val plasmaObj  = new PlasmaTelevision("Panasonic Viera")
  println(plasmaObj.modelWithType)
  }

}
```

**Output:**


Scala - ListDemoProject/src/org/lists/InheritanceDemo.scala - Eclipse
File Edit Refactor Navigate Search Project Scala Run Window Help
Quick Access
Java EE
Scala
Console

```
<terminated> InheritanceDemo$ [Scala Application] /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.79.x86_64/jre/bin/java (Jan 28, 2018, 10:31:50 AM
Onida LEO is a LED TV
Sony Bravia is a LCD TV
Panasonic Viera is a Plasma TV
```

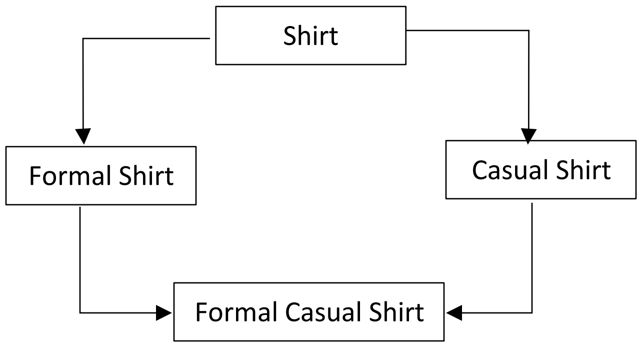
2. Write a simple program to show multiple inheritance in Scala.

**Multiple inheritance:**

In Scala, a sub class cannot inherit members of multiple super classes as it leads to an ambiguity when same members (fields and/or methods) are defined in two or more super classes.

As a solution to this, Scala introduces a new concept called **'traits'**. A trait is a group of abstract and non-abstract methods. That means a trait may or may not implement methods. In case it doesn't provide implementation, it's the responsibility of a class that refers to the trait to do the same.

Let's see an example in Scala to understand it better:



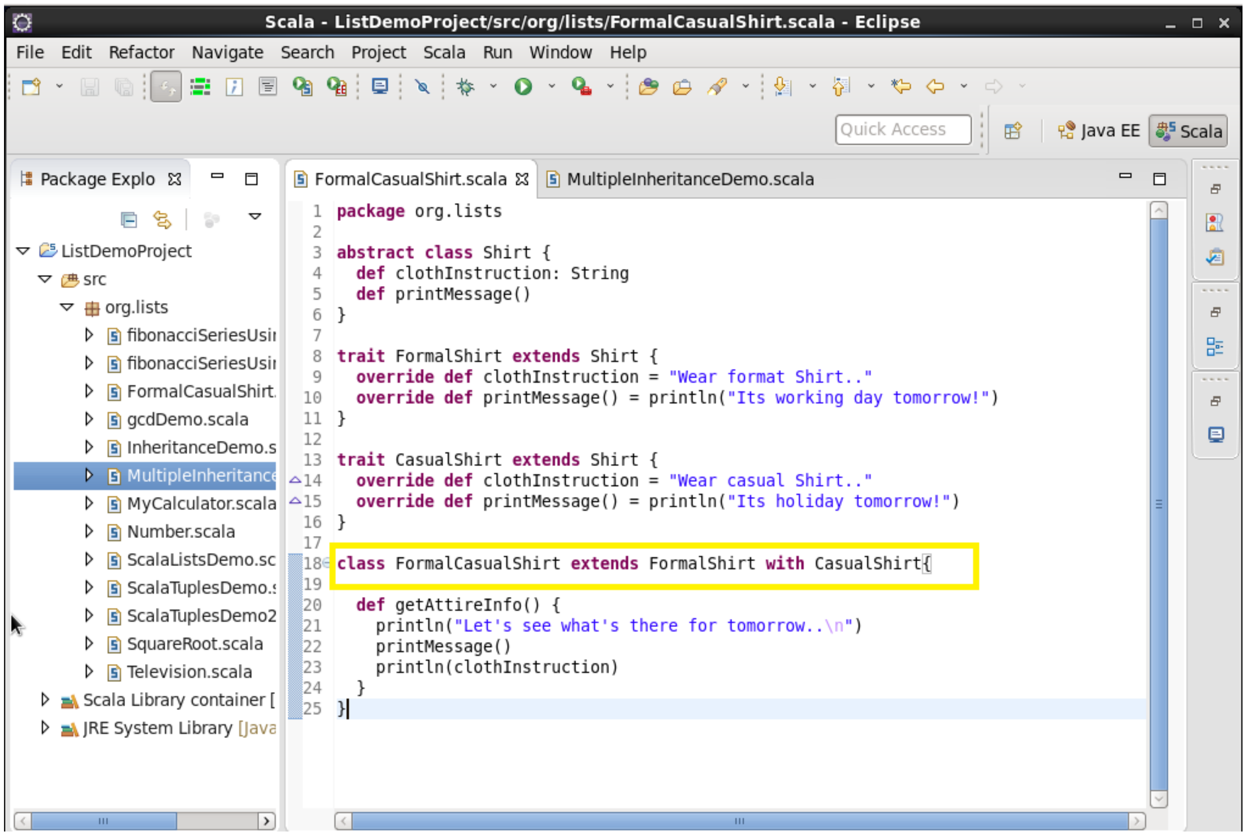
**FormalCasualShirt.scala**

```
abstract class Shirt {                          // base class with a data field and a method
  def clothInstruction: String
  def printMessage()
}
```

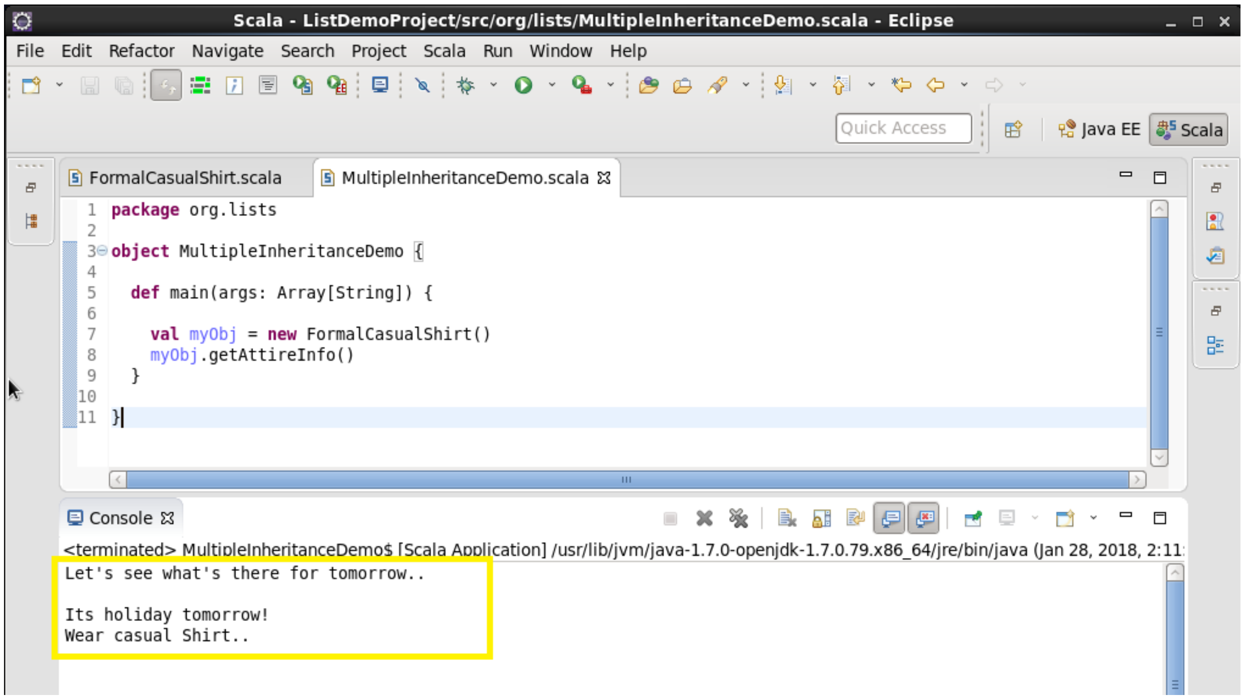
```
trait FormalShirt extends Shirt {                    // first trait that extends base class and
                                                     overrides the base class members
  override def clothInstruction = "Wear format Shirt.."
  override def printMessage() = println("Its working day tomorrow!")
}
trait CasualShirt extends Shirt {                    // second trait that extends base class
                                                     and overrides the base class members
  override def clothInstruction = "Wear casual Shirt.."
  override def printMessage() = println("Its holiday tomorrow!")
}
class FormalCasualShirt extends FormalShirt with CasualShirt {
                                                     // finally, the sub class that uses
  def getAttireInfo() {                              members of either of above two traits
    println("Let's see what's there for tomorrow..\n")
    printMessage()
    println(clothInstruction)
  }
}
```

An interesting part in the above code is the creation of class, ‘FormalCasualShirt’ which extends trait ‘FormaShirt’ along with another trait ‘CasualShirt’. Since the trait ‘CasualShirt’ is associated with **‘with’** keyword, **members of this trait will be called** whenever this class tries to access them. Let’s create an object of this class and call the method, getAttireInfo() to see the output:

```
object MultipleInheritanceDemo {
  def main(args: Array[String]) {
    val myObj = new FormalCasualShirt()          //create an object of class: FormalCasualShirt
    myObj.getAttireInfo()                        // invoke method getAttireInfo()
  }
}
```



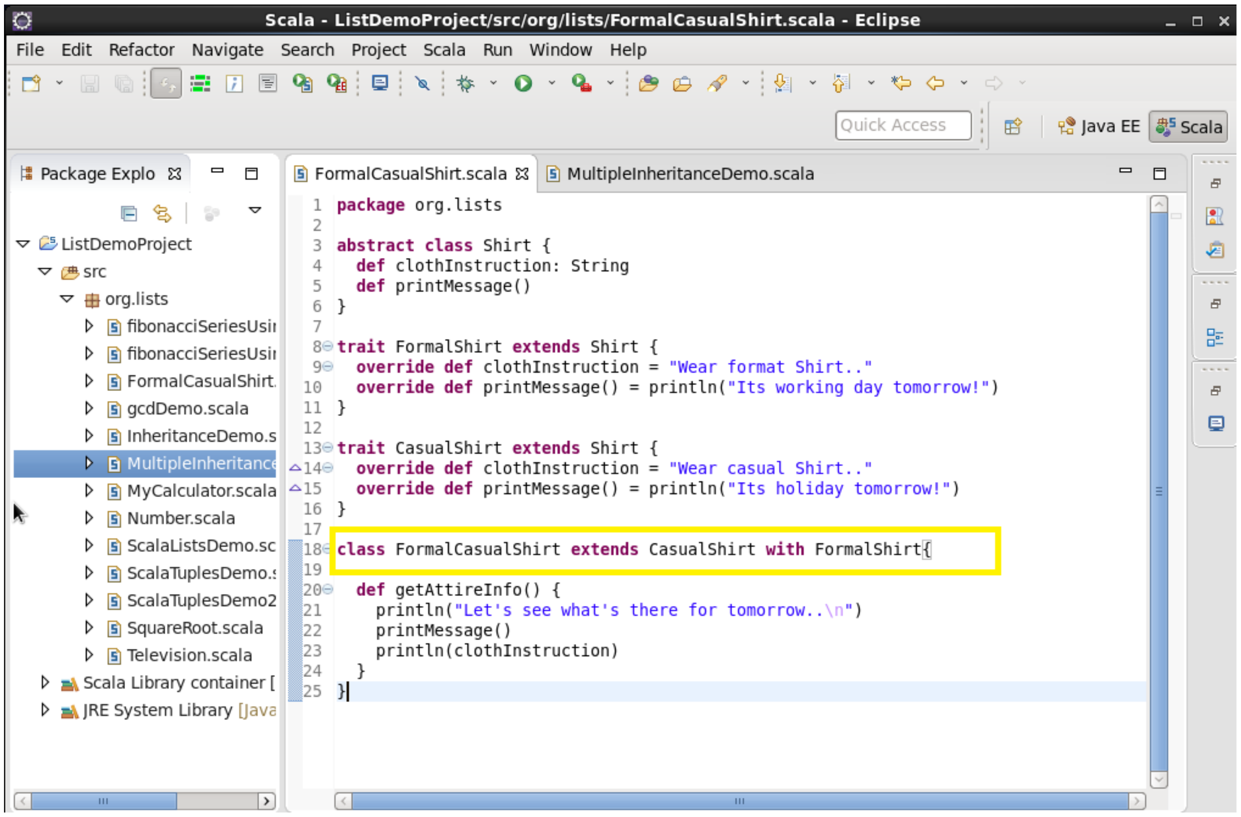
```
package org.lists

abstract class Shirt {
  def clothInstruction: String
  def printMessage()
}

trait FormalShirt extends Shirt {
  override def clothInstruction = "Wear format Shirt.."
  override def printMessage() = println("Its working day tomorrow!")
}

trait CasualShirt extends Shirt {
  override def clothInstruction = "Wear casual Shirt.."
  override def printMessage() = println("Its holiday tomorrow!")
}

class FormalCasualShirt extends FormalShirt with CasualShirt{

  def getAttireInfo() {
    println("Let's see what's there for tomorrow..\n")
    printMessage()
    println(clothInstruction)
  }
}
```



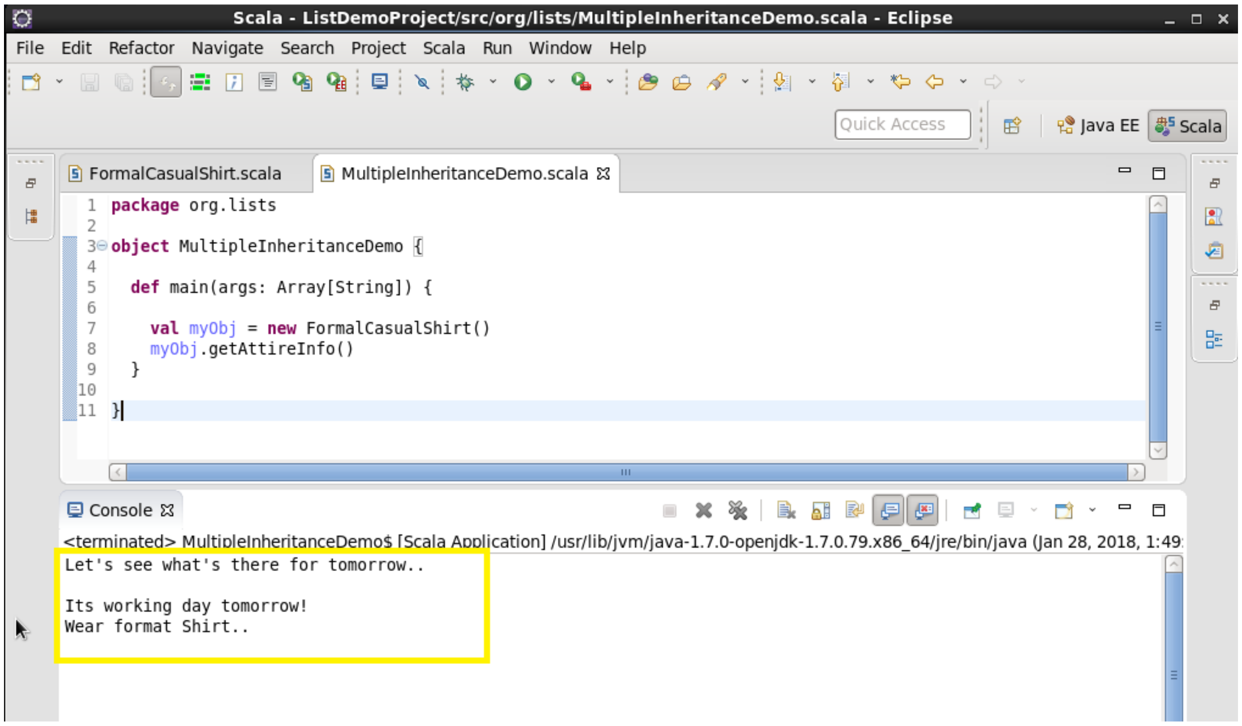
```
package org.lists

object MultipleInheritanceDemo {

  def main(args: Array[String]) {

    val myObj = new FormalCasualShirt()
    myObj.getAttireInfo()
  }

}
```

```
<terminated> MultipleInheritanceDemo$ [Scala Application] /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.79.x86_64/jre/bin/java (Jan 28, 2018, 2:11:
Let's see what's there for tomorrow..

Its holiday tomorrow!
Wear casual Shirt..
```

As we can see in the console of above screen shot, it has picked up members of trait ‘CasualShirt’ and has printed the associated values. Let’s interchange these traits while creating the ‘FormalCasualShirt’ and see what it prints:



```
package org.lists

abstract class Shirt {
  def clothInstruction: String
  def printMessage()
}

trait FormalShirt extends Shirt {
  override def clothInstruction = "Wear format Shirt.."
  override def printMessage() = println("Its working day tomorrow!")
}

trait CasualShirt extends Shirt {
  override def clothInstruction = "Wear casual Shirt.."
  override def printMessage() = println("Its holiday tomorrow!")
}

class FormalCasualShirt extends CasualShirt with FormalShirt{

  def getAttireInfo() {
    println("Let's see what's there for tomorrow..\n")
    printMessage()
    println(clothInstruction)
  }
}
```


Scala - ListDemoProject/src/org/lists/MultipleInheritanceDemo.scala - Eclipse
File Edit Refactor Navigate Search Project Scala Run Window Help
Quick Access
Java EE
Scala
FormalCasualShirt.scala
MultipleInheritanceDemo.scala
Console

```
package org.lists

object MultipleInheritanceDemo {

  def main(args: Array[String]) {

    val myObj = new FormalCasualShirt()
    myObj.getAttireInfo()
  }

}
```

```
<terminated> MultipleInheritanceDemo$ [Scala Application] /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.79.x86_64/jre/bin/java (Jan 28, 2018, 1:49:
Let's see what's there for tomorrow..

Its working day tomorrow!
Wear format Shirt..
```

As we have anticipated, this time it has picked up members of the trait ‘FormalShirt’ since we are referring that alongside ‘with’ keyword while defining the sub class. This is how we can achieve **multiple inheritance in Scala with the use of traits**.