Acadgild – Assignment 44

**Task 1**
A Fibonacci series (starting from 1) written in order without any spaces in between, thus producing a sequence of digits.
Write a Scala application to find the Nth digit in the sequence.

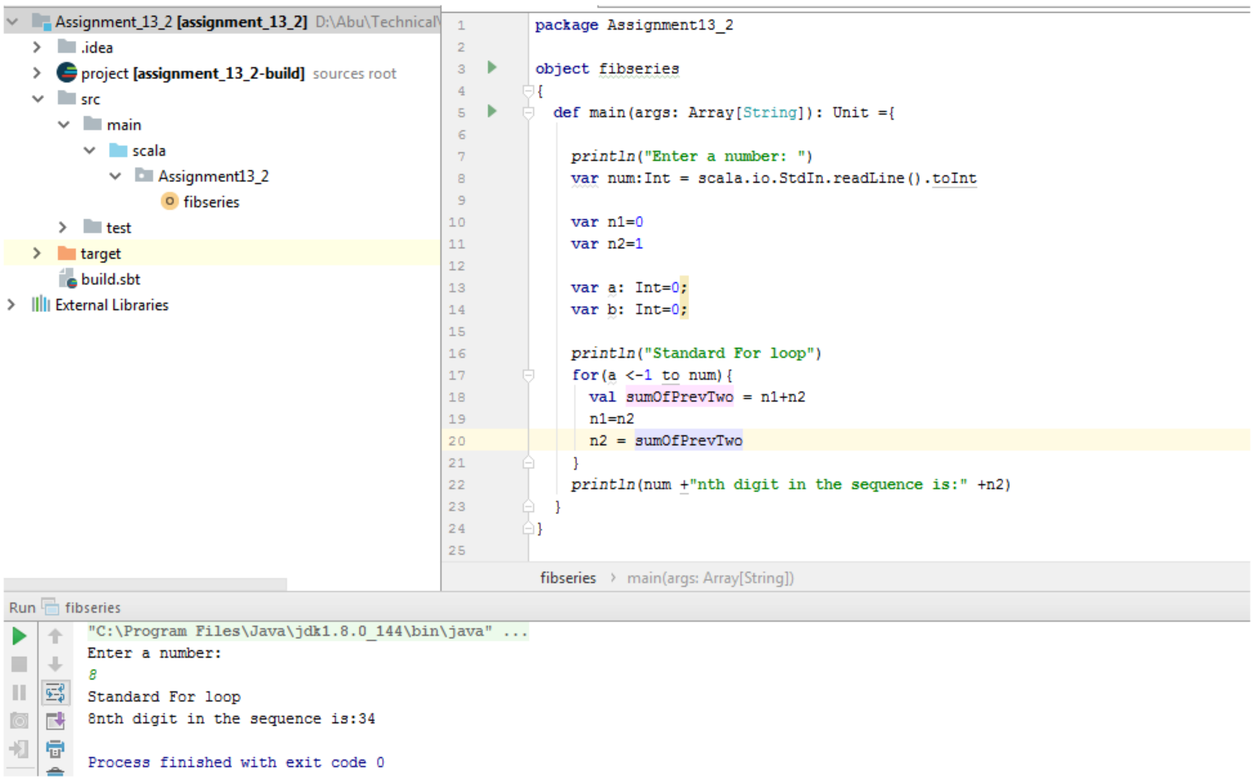
- Write the function using standard for loop
- Write the function using recursion

## write function using standard for loop

### Scala code

```
package Assignment44
object fibseries
{
  def main(args: Array[String]): Unit ={

    println("Enter a number: ")
    var num:Int = scala.io.StdIn.readLine().toInt

    var n1=0
    var n2=1

    var a: Int=0;
    var b: Int=0;

    println("Standard For loop")
    for(a <-1 to num){
      val sumOfPrevTwo = n1+n2
      n1=n2
      n2 = sumOfPrevTwo
    }
    println(num +"nth digit in the sequence is:" +n2)
  }
}
```

Screen Shot:



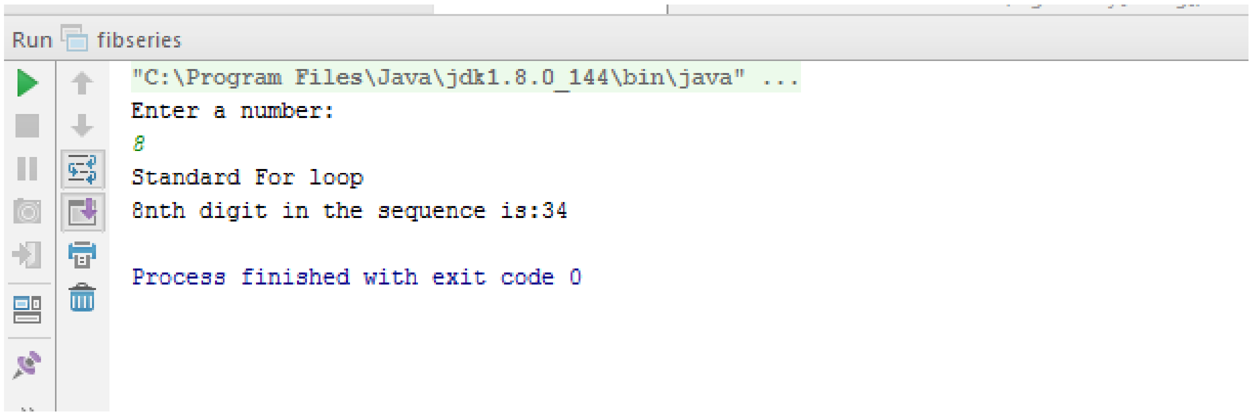
```
Assignment_13_2 [assignment_13_2] D:\Abu\Technical
  .idea
  project [assignment_13_2-build] sources root
  src
    main
      scala
        Assignment13_2
          fibseries
    test
  target
  build.sbt
External Libraries
```

```scala
package Assignment13_2

object fibseries
{
  def main(args: Array[String]): Unit ={

    println("Enter a number: ")
    var num:Int = scala.io.StdIn.readLine().toInt

    var n1=0
    var n2=1

    var a: Int=0;
    var b: Int=0;

    println("Standard For loop")
    for(a <-1 to num){
      val sumOfPrevTwo = n1+n2
      n1=n2
      n2 = sumOfPrevTwo
    }
    println(num +"nth digit in the sequence is:" +n2)
  }
}
```

```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
Enter a number:
8
Standard For loop
8nth digit in the sequence is:34

Process finished with exit code 0
```

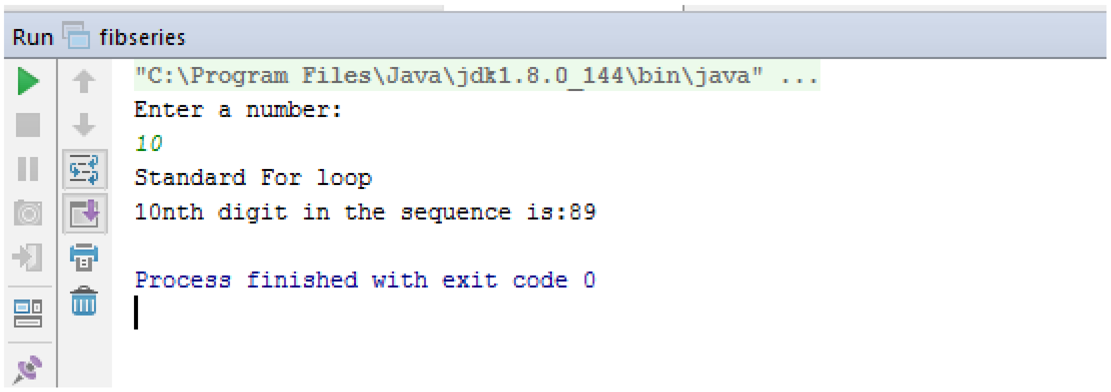
## Output

When we provide number 8 as input, the 8th digit in the Fibonacci sequence is 34.



```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
Enter a number:
8
Standard For loop
8nth digit in the sequence is:34

Process finished with exit code 0
```

If we give the input as 10, the 10th digit of Fibonacci sequence is 89



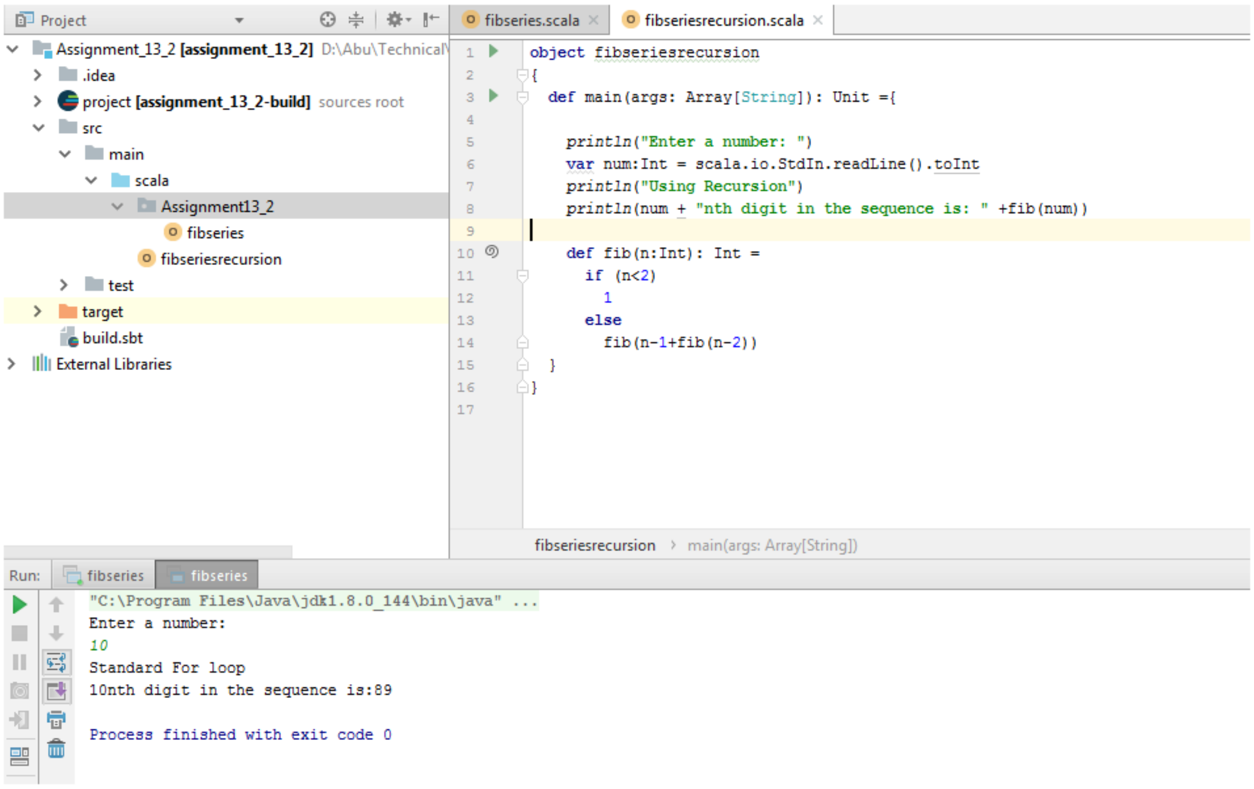
```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
Enter a number:
10
Standard For loop
10nth digit in the sequence is:89

Process finished with exit code 0
```

## Write the function using recursion

### Scala code

```
object fibseriesrecursion
{
  def main(args: Array[String]): Unit ={

    println("Enter a number: ")
    var num:Int = scala.io.StdIn.readLine().toInt
    println("Using Recursion")
    println(num + "nth digit in the sequence is: " +fib(num))

  def fib(n:Int): Int =
    if (n<2)
      1
    else
      fib(n-1+fib(n-2))
  }
}
```

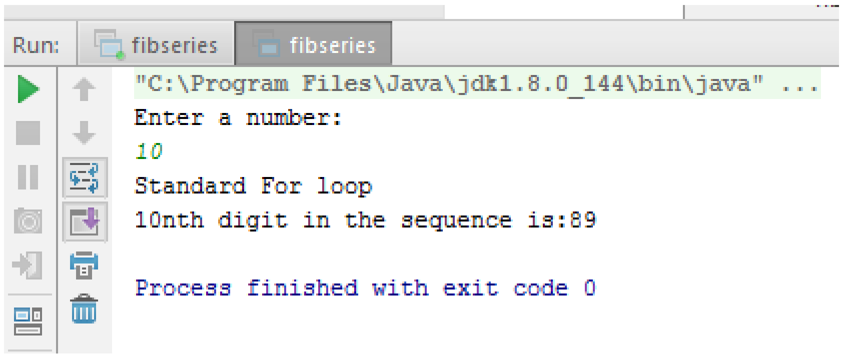
Screen shot:



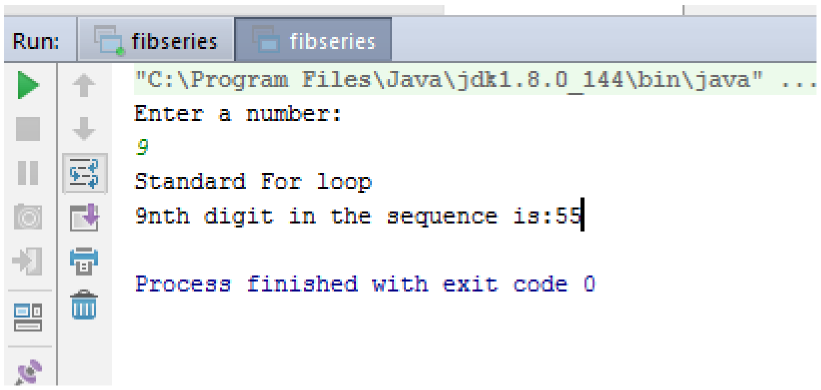
```scala
object fibseriesrecursion
{
  def main(args: Array[String]): Unit ={

    println("Enter a number: ")
    var num:Int = scala.io.StdIn.readLine().toInt
    println("Using Recursion")
    println(num + "nth digit in the sequence is: " +fib(num))

    def fib(n:Int): Int =
      if (n<2)
        1
      else
        fib(n-1+fib(n-2))
  }
}
```

```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
Enter a number:
10
Standard For loop
10nth digit in the sequence is:89

Process finished with exit code 0
```

## Output



```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
Enter a number:
10
Standard For loop
10nth digit in the sequence is:89

Process finished with exit code 0
```



```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
Enter a number:
9
Standard For loop
9nth digit in the sequence is:55

Process finished with exit code 0
```

**Task 2**
Create a calculator to work with rational numbers.
Requirements:
- It should provide capability to add, subtract, divide and multiply rational numbers
- Create a method to compute GCD (this will come in handy during operations on rational)

Add option to work with whole numbers which are also rational numbers i.e. (n/1)
- achieve the above using auxiliary constructors
- enable method overloading to enable each function to work with numbers and rational.

## Create a Scala Class *“Calc”*

Scala Code

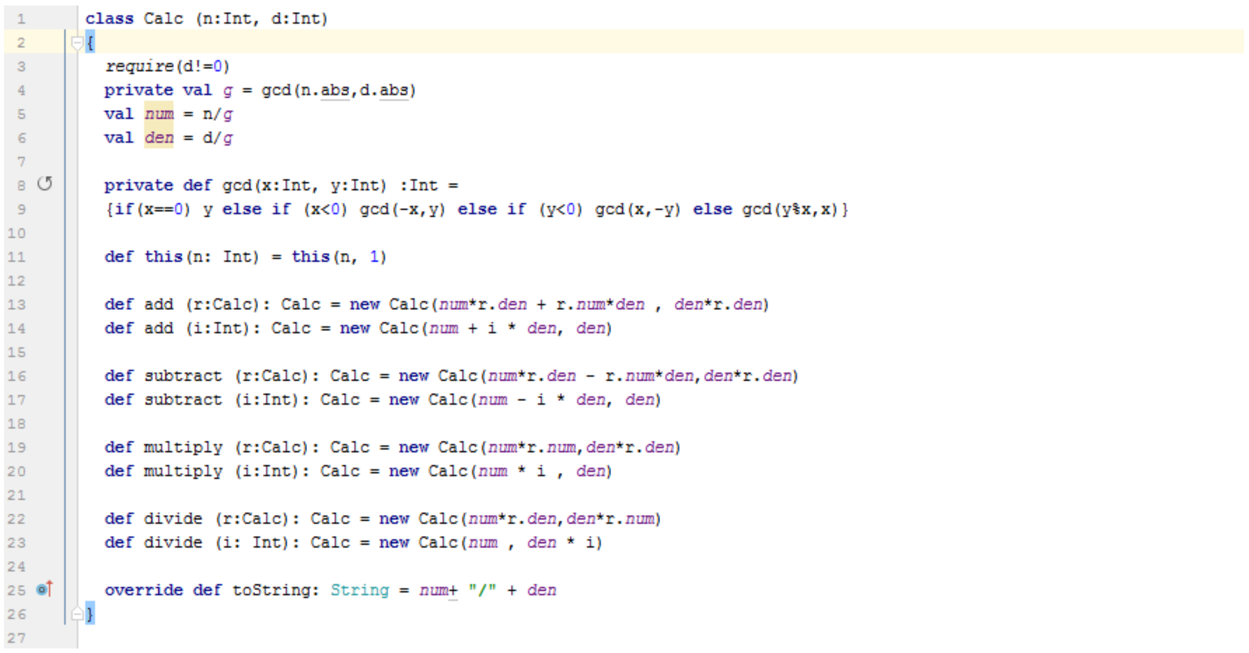
```
class Calc (n:Int, d:Int)
{
  require(d!=0)
  private val g = gcd(n.abs,d.abs)
  val num = n/g
  val den = d/g

  private def gcd(x:Int, y:Int) :Int =
  {if(x==0) y else if (x<0) gcd(-x,y) else if (y<0) gcd(x,-y) else gcd(y%x,x)}

  def this(n: Int) = this(n, 1) // auxiliary constructor

  def add (r:Calc): Calc = new Calc(num*r.den + r.num*den , den*r.den)
  def add (i:Int): Calc = new Calc(num + i * den, den) //method overloading for add

  def subtract (r:Calc): Calc = new Calc(num*r.den - r.num*den,den*r.den)
  def subtract (i:Int): Calc = new Calc(num - i * den, den)//method overloading for
subtract
```

The statement, ***“def this(n: Int) = this(n, 1)*** **“** is an auxiliary constructor, we have created an Object **“CalcObj”** to perform the above functions.

We have Enabled method **overloading** to enable each function (add, sub, multiplication and division) to work with numbers and rational.

We have written the code in such a way that it works with whole numbers as well as with rational numbers (n/1).

IntelliJ console,

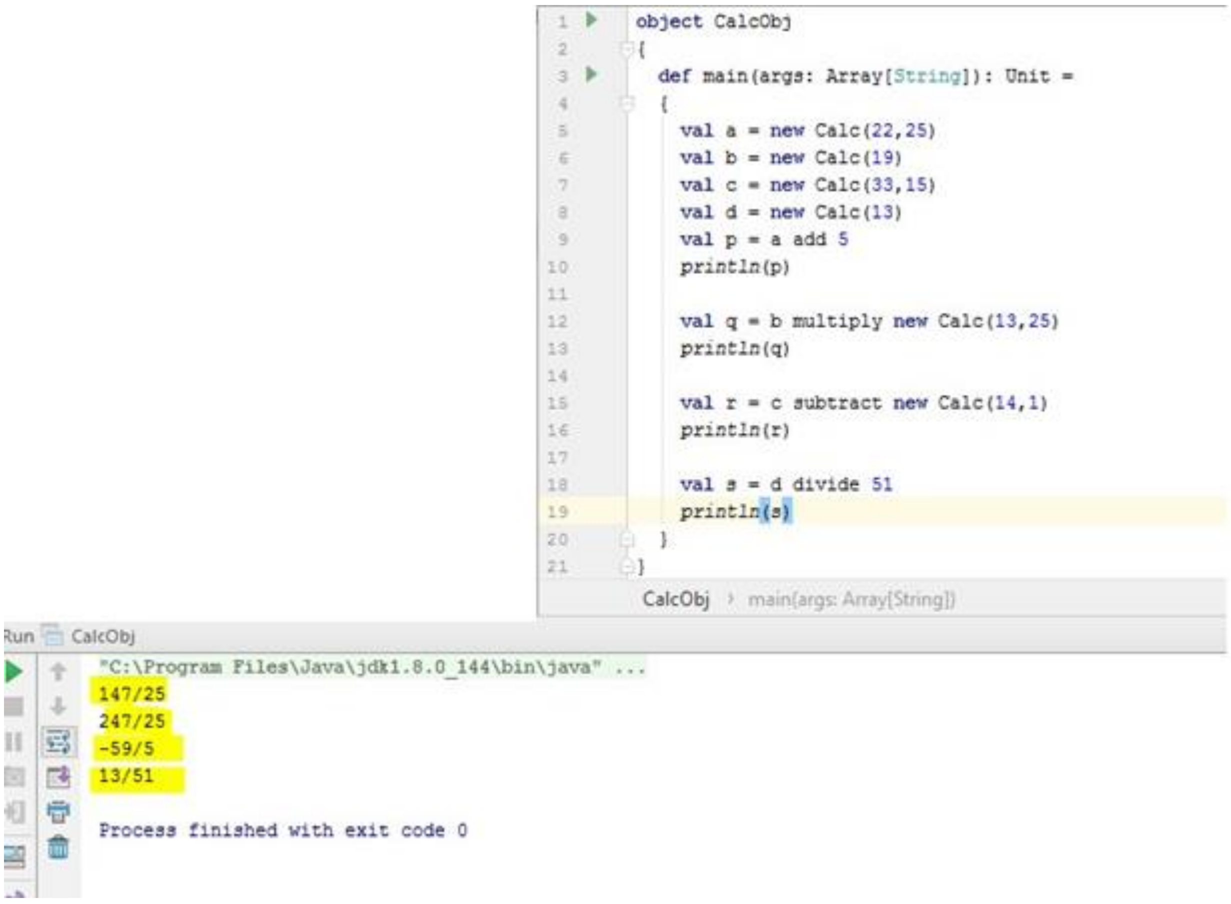


```
class Calc (n:Int, d:Int)
{
  require(d!=0)
  private val g = gcd(n.abs,d.abs)
  val num = n/g
  val den = d/g

  private def gcd(x:Int, y:Int) :Int =
  {if(x==0) y else if (x<0) gcd(-x,y) else if (y<0) gcd(x,-y) else gcd(y%x,x)}

  def this(n: Int) = this(n, 1)

  def add (r:Calc): Calc = new Calc(num*r.den + r.num*den , den*r.den)
  def add (i:Int): Calc = new Calc(num + i * den, den)

  def subtract (r:Calc): Calc = new Calc(num*r.den - r.num*den,den*r.den)
  def subtract (i:Int): Calc = new Calc(num - i * den, den)

  def multiply (r:Calc): Calc = new Calc(num*r.num,den*r.den)
  def multiply (i:Int): Calc = new Calc(num * i , den)

  def divide (r:Calc): Calc = new Calc(num*r.den,den*r.num)
  def divide (i: Int): Calc = new Calc(num , den * i)

  override def toString: String = num+ "/" + den
}
```

## Create a Scala Object “CalObj”

```
object CalcObj

{

  def main(args: Array[String]): Unit =

  {

    val a = new Calc(22,25)
    val b = new Calc(19)
    val c = new Calc(33,15)
    val d = new Calc(13)


    val p = a add 5

    println(p)


    val q = b multiply new Calc(13,25)

    println(q)


    val r = c subtract new Calc(14,1)
```

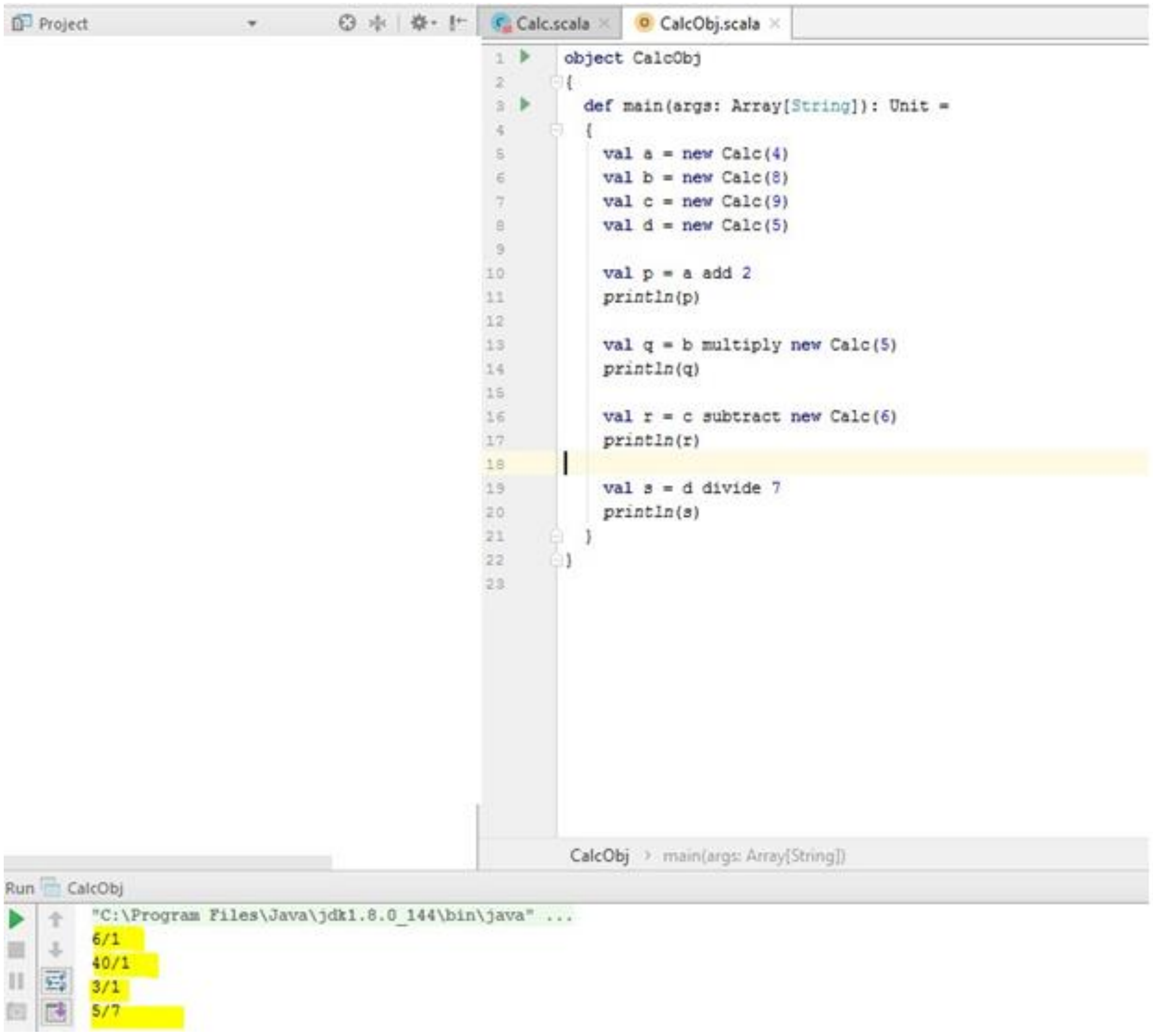
1. Example 1,



```
object CalcObj
{
  def main(args: Array[String]): Unit =
  {
    val a = new Calc(22,25)
    val b = new Calc(19)
    val c = new Calc(33,15)
    val d = new Calc(13)
    val p = a add 5
    println(p)

    val q = b multiply new Calc(13,25)
    println(q)

    val r = c subtract new Calc(14,1)
    println(r)

    val s = d divide 51
    println(s)
  }
}
```

```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
147/25
247/25
-59/5
13/51

Process finished with exit code 0
```

2. Example 2,



```scala
object CalcObj
{
  def main(args: Array[String]): Unit =
  {
    val a = new Calc(4)
    val b = new Calc(8)
    val c = new Calc(9)
    val d = new Calc(5)

    val p = a add 2
    println(p)

    val q = b multiply new Calc(5)
    println(q)

    val r = c subtract new Calc(6)
    println(r)

    val s = d divide 7
    println(s)
  }
}
```

```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
6/1
40/1
3/1
5/7
```

## Task 3

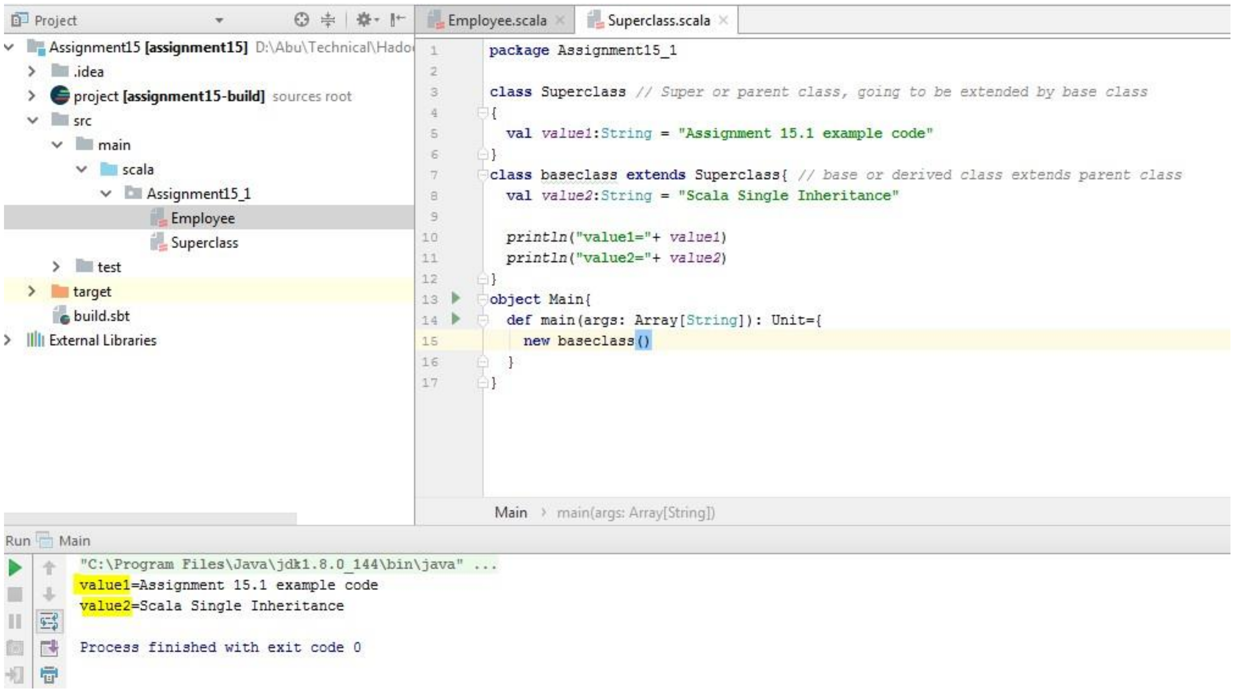
1.Write a simple program to show inheritance in scala.
2.Write a simple program to show multiple inheritance in scala.

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.

Scala Code

```
package Assignment44

class Superclass // Super or parent class, going to be extended by base class

{

  val value1:String = "Assignment 15.1 example code"

}

class baseclass extends Superclass{ // base or derived class extends parent class

  val value2:String = "Scala Single Inheritance"


  println("value1="+ value1)
  println("value2="+ value2)
```

## Output



```scala
package Assignment15_1

class Superclass // Super or parent class, going to be extended by base class
{
  val value1:String = "Assignment 15.1 example code"
}
class baseclass extends Superclass{ // base or derived class extends parent class
  val value2:String = "Scala Single Inheritance"

  println("value1="+ value1)
  println("value2="+ value2)
}
object Main{
  def main(args: Array[String]): Unit={
    new baseclass()
  }
}
```

```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
value1=Assignment 15.1 example code
value2=Scala Single Inheritance

Process finished with exit code 0
```

Multiple inheritance is a feature of some object-oriented computer programming languages in which an object or class can inherit characteristics and features from more than one parent object or parent class. It is distinct from single inheritance, where an object or class may only inherit from one particular object or class.

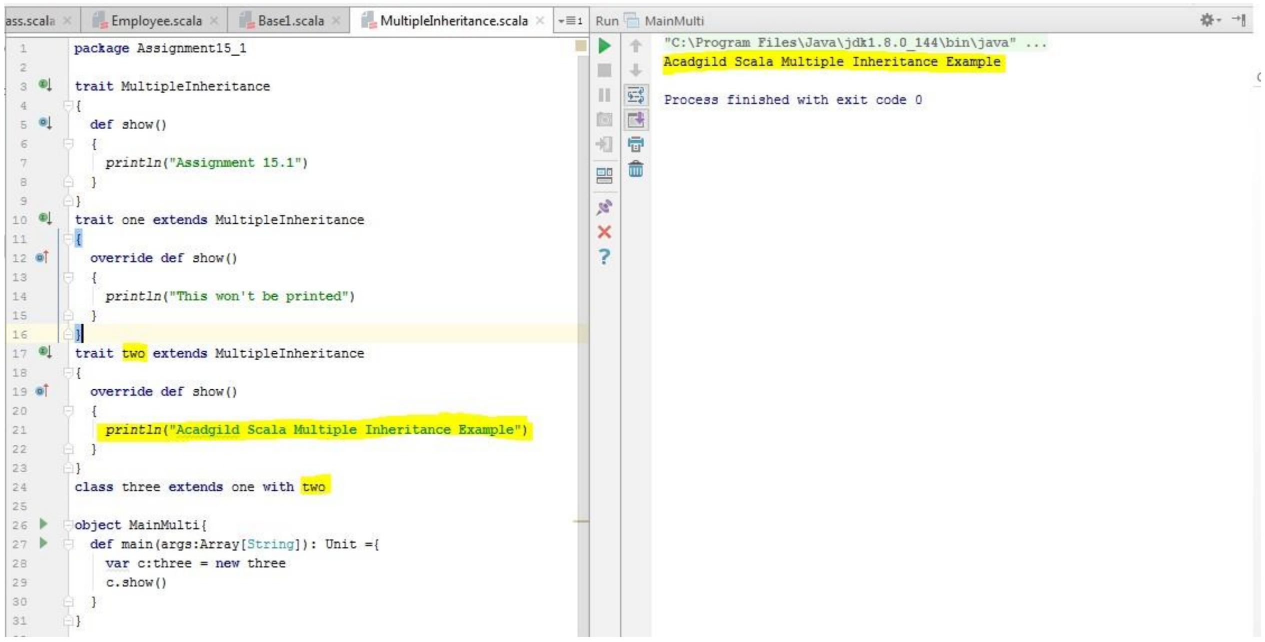
Scala supports various types of inheritance including single, multilevel, **multiple**, and hybrid. You can use single, multilevel and hierarchal in your class. **Multiple** and **hybrid** can only be achieved by using **traits**.

Scala **doesn't allow for multiple inheritance** per se, but allows to extend multiple **traits**.

Traits are used to share interfaces and fields between classes. They are similar to Java 8's interfaces. Classes and objects can extend traits but traits cannot be instantiated and therefore have no parameters. Traits in Scala are best described as "**interfaces that can provide concrete members**."

```scala
package Assignment44

trait MultipleInheritance //parent trait

{

  def show() // defining the function show()

  {

    println("Assignment 44")

  }

}

trait one extends MultipleInheritance // extending the parent trait

{

  override def show()

  {

    println("This won't be printed")

  }

}

trait two extends MultipleInheritance // extending the parent trait

{

  override def show()
```

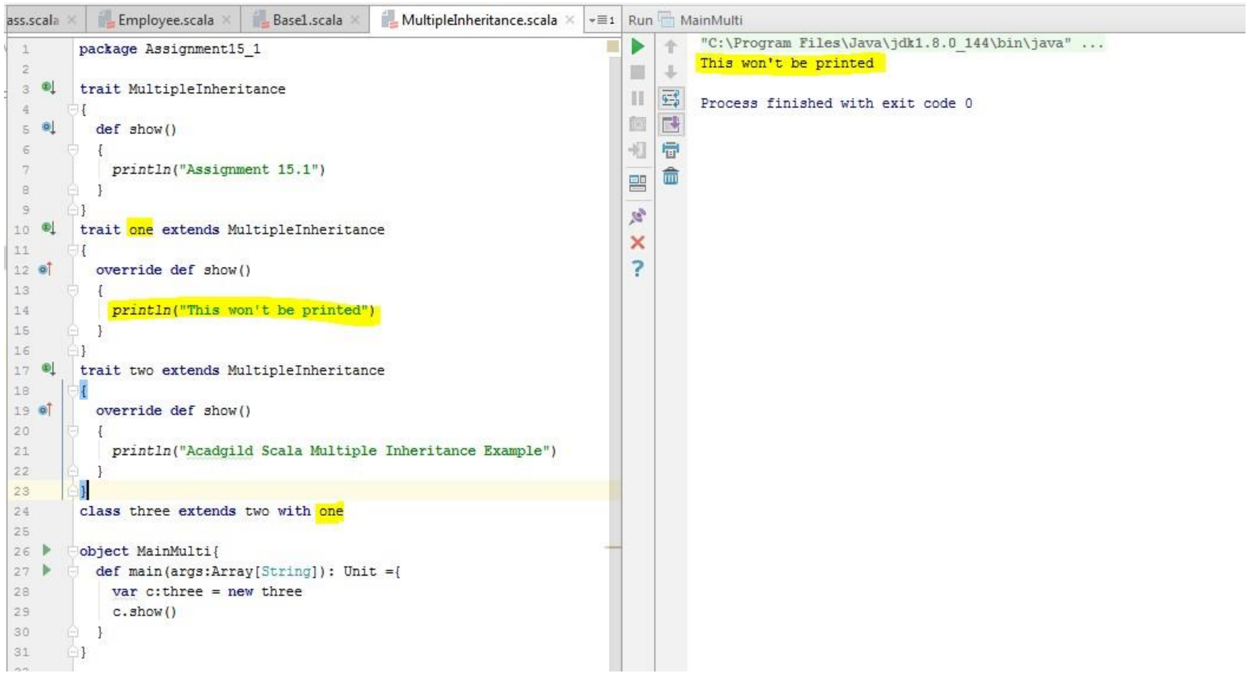
Example 1, here the class ***three*** calling the trait one with ***two***, the ***two*** in the last order and hence the function of ***two*** will be called and output is,



```scala
package Assignment15_1

trait MultipleInheritance
{
  def show()
  {
    println("Assignment 15.1")
  }
}
trait one extends MultipleInheritance
{
  override def show()
  {
    println("This won't be printed")
  }
}
trait two extends MultipleInheritance
{
  override def show()
  {
    println("Acadgild Scala Multiple Inheritance Example")
  }
}
class three extends one with two

object MainMulti{
  def main(args:Array[String]): Unit ={
    var c:three = new three
    c.show()
  }
}
```

```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
Acadgild Scala Multiple Inheritance Example

Process finished with exit code 0
```

Example 2, in this example the object ***MainMulti*** called the trait ***one*** and see the result below,



```scala
package Assignment15_1

trait MultipleInheritance
{
  def show()
  {
    println("Assignment 15.1")
  }
}
trait one extends MultipleInheritance
{
  override def show()
  {
    println("This won't be printed")
  }
}
trait two extends MultipleInheritance
{
  override def show()
  {
    println("Acadgild Scala Multiple Inheritance Example")
  }
}
class three extends two with one

object MainMulti{
  def main(args:Array[String]): Unit ={
    var c:three = new three
    c.show()
  }
}
```

```
"C:\Program Files\Java\jdk1.8.0_144\bin\java" ...
This won't be printed

Process finished with exit code 0
```