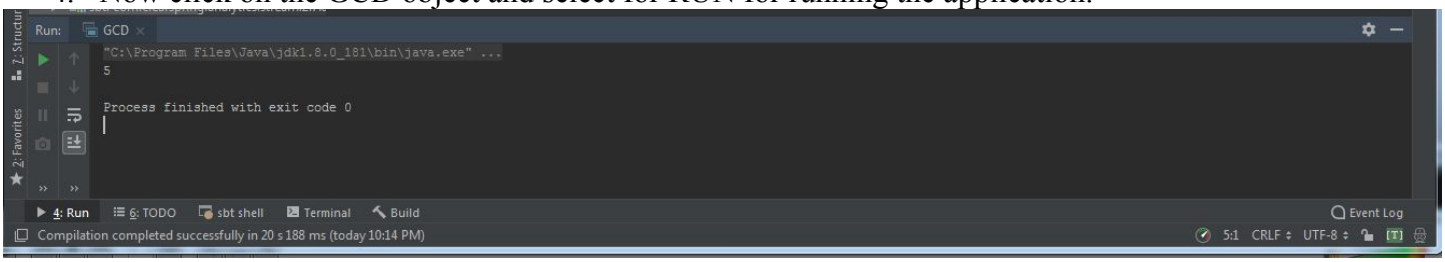
Session 15: SCALA BASICS 2
Assignment 1

Task 1: Create a Scala application to find the GCD of two numbers.

1. Open the intelliJ IDEA create a new scala project.
2. Create a scala object for GCD.
3. Now write the following application into that scala object as follows.

```
object GCD {

  def gcd(a: Int,b: Int): Int = {
    if(b ==0) a else gcd(b, a%b)
  }

  def main(args: Array[String]) {
    println(gcd(25,15))
  }
}
```

4. Now click on the GCD object and select for RUN for running the application.



5 We can observe the output as '5' from the application.

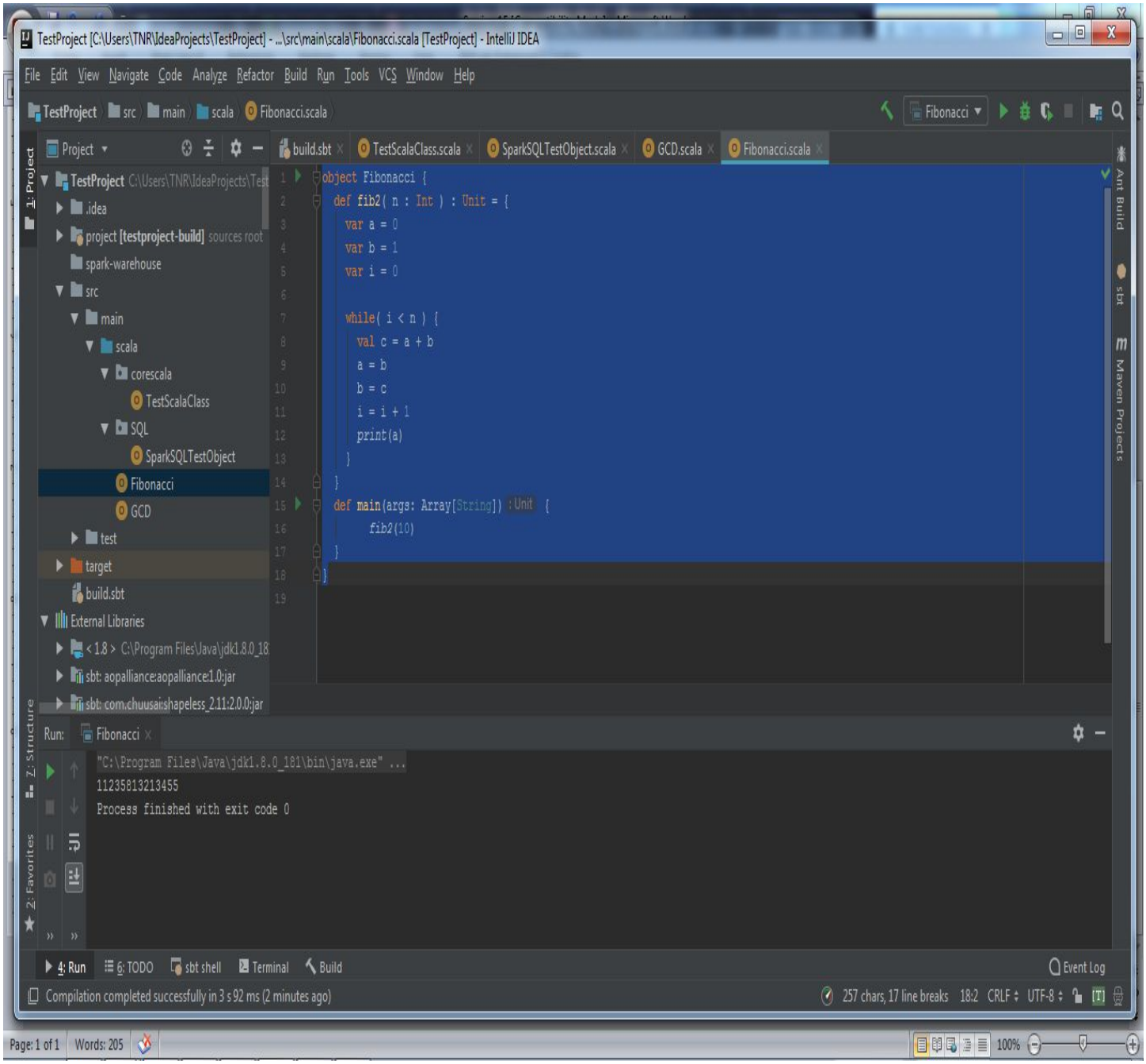
Task 2:

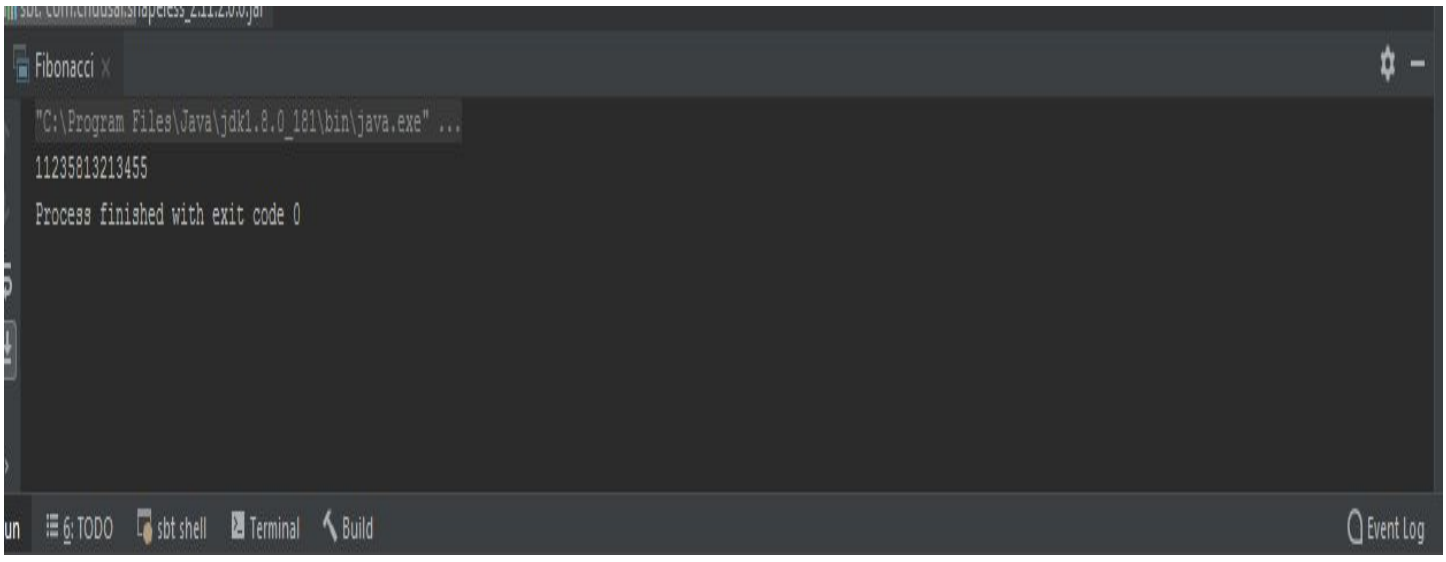
Fibonacci series (starting from 1) written in order without any spaces between, thus producing a sequence of digits.

1. Create a Scala object for Fibonacci and write the following code in to that object:

```
object Fibonacci {
  def fib2( n : Int ) : Unit = {
    var a = 0
    var b = 1
    var i = 0

    while( i < n ) {
      val c = a + b
      a = b
      b = c
      i = i + 1
      print(a)
    }
  }
  def main(args: Array[String]) {
        fib2(10)
  }
}
```

2. Right click on the Fibonacci scala object and select Run option:


TestProject [C:\Users\TNR\IdeaProjects\TestProject] - ...\src\main\scala\Fibonacci.scala [TestProject] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
TestProject src main scala Fibonacci.scala
Fibonacci
Project
build.sbt
TestScalaClass.scala
SparkSQLTestObject.scala
GCD.scala
Fibonacci.scala
TestProject C:\Users\TNR\IdeaProjects\Test
.idea
project [testproject-build] sources root
spark-warehouse
src
main
scala
corescala
TestScalaClass
SQL
SparkSQLTestObject
Fibonacci
GCD
test
target
build.sbt
External Libraries
< 1.8 > C:\Program Files\Java\jdk1.8.0_18
sbt: aopalliance:aopalliance:1.0:jar
sbt: com.chuusai:shapeless_2.11:2.0.0:jar
object Fibonacci {
def fib2(n : Int) : Unit = {
var a = 0
var b = 1
var i = 0
while(i < n) {
val c = a + b
a = b
b = c
i = i + 1
print(a)
}
}
def main(args: Array[String]) :Unit {
fib2(10)
}
}
Run: Fibonacci
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...
11235813213455
Process finished with exit code 0
4: Run
6: TODO
sbt shell
Terminal
Build
Event Log
Compilation completed successfully in 3 s 92 ms (2 minutes ago)
257 chars, 17 line breaks
18:2
CRLF
UTF-8
Page: 1 of 1
Words: 205
100%


Fibonacci
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...
11235813213455
Process finished with exit code 0
6: TODO
sbt shell
Terminal
Build
Event Log

3. We can observe that the output as shown in the above figure, for the Fibonacci series 10.
4. The output we can observe as 11235813213455.

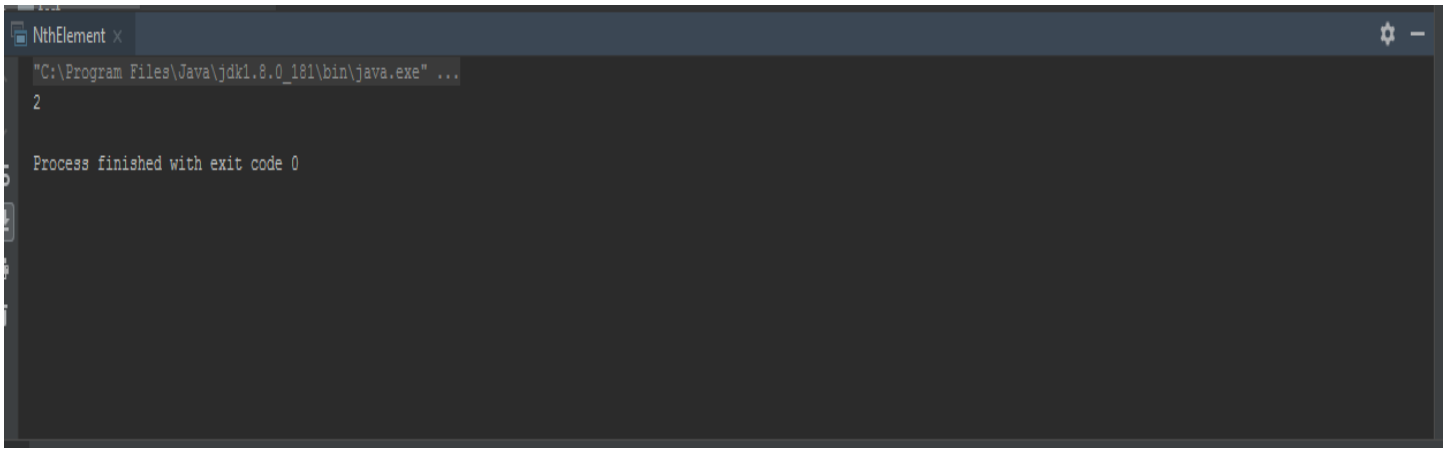
Write a Scala application to find the Nth digit in the sequence.

- **Write the function using standard for loop.**

1. Create the scala object called 'NthNode' and write the following code into the object:

```
2.    object NthElement {
      def nthEle(a: Int): Unit = {
        val list = List(1,2,3,4,5)
        val arr = Array[Int](list:_*)

        for(n <- arr)
          if(a == n)
            println(arr(n))
      }

      def main(args: Array[String]) {
        nthEle(1)
      }
    }
```

2. Right click on the 'NthNode' object and select Run option to run the code.



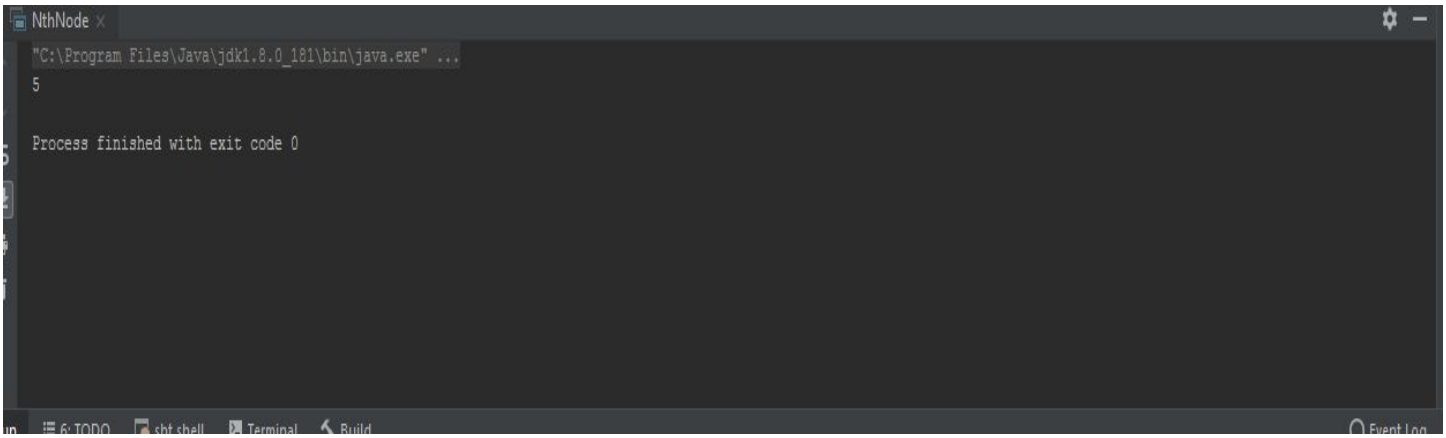
3. We can observe that the position we are searching is 1st postion, the element present in the 1st position is 2.

Write a scala application to find the Nth digit in the sequence using recursion.

1. Create the scala object called 'NthNode' and write the following code into the object.

```
2.    package NthElement

    object NthNode {
      def findKth[A](k:Int, l:List[A]):A = k match {
        case 0 => l.head
        case k if k > 0 => findKth(k - 1, l.tail)
        case _ => throw new NoSuchElementException
      }
      def main(args:Array[String]) :Unit = {
        val myList = List(1,2,3,4,5)
        println(findKth(0, myList))
      }

    }
```

3. Right click on the 'NthNode' select Run option to run the application.



4. We can observe that the position we are looking at position 4 is 5.

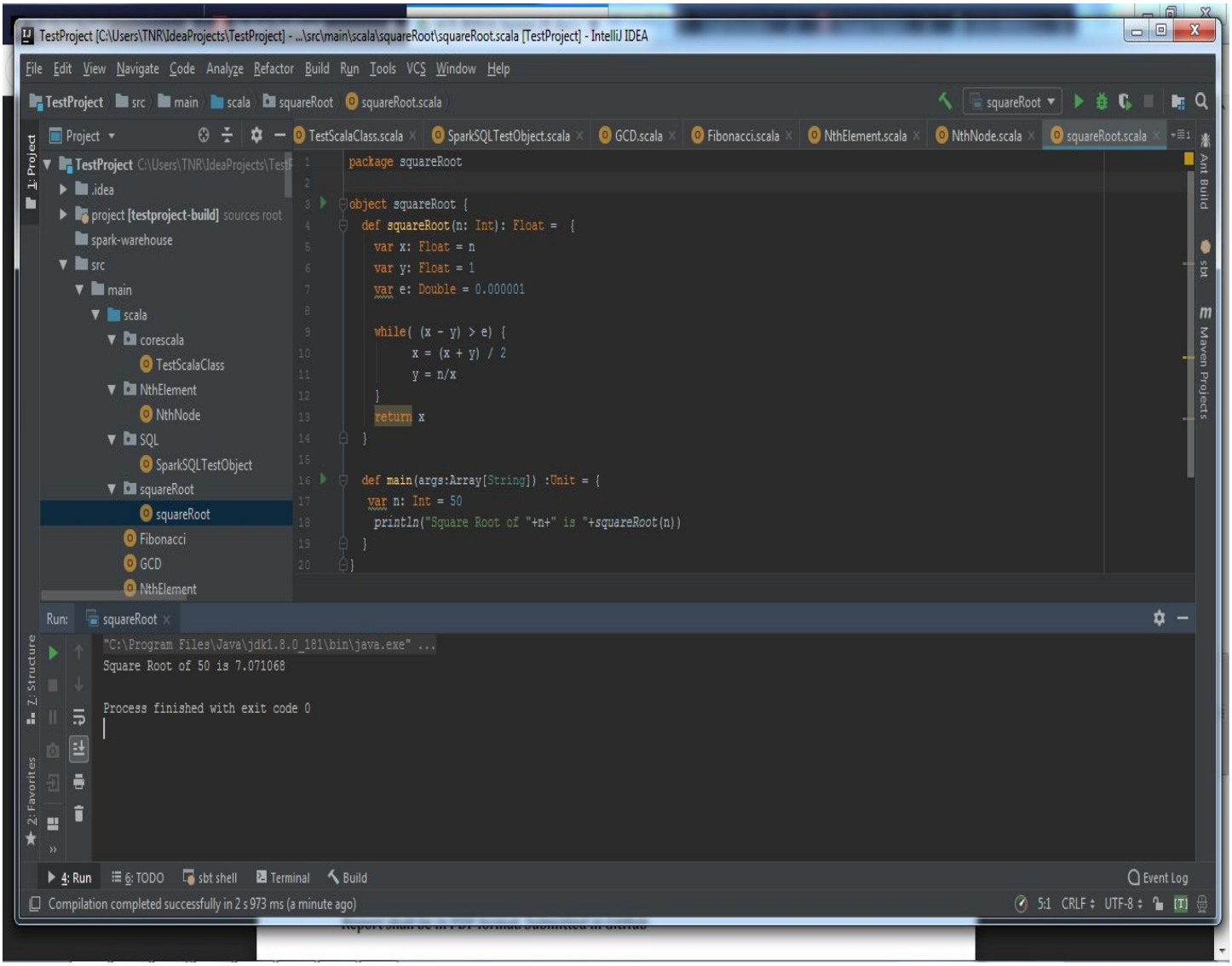
Task 3:
Find square root of numbers using Babylonian method.

1. **Start with an arbitrary positive start value x (the closer to the root, the better).**
2. **Initialize y = 1.**
3. **Do following until desired approximation is achieved.**
4. **Get the next approximation for root using average of x and y.**
5. **Set y = n/x.**

1. Create a scala package with name squareRoot.
2. Create a scala class with object with squareRoot as name, and load the following code in to it.

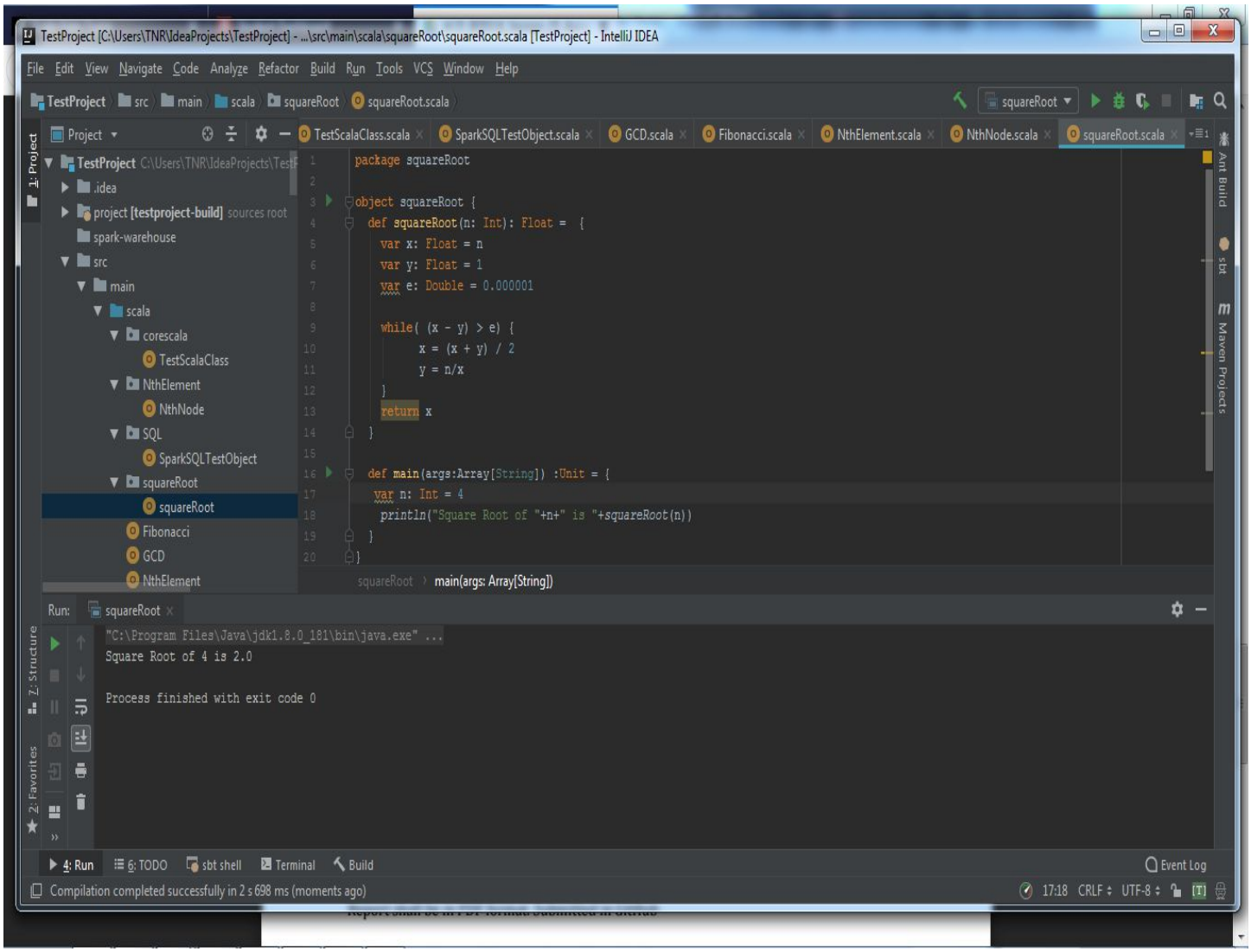
```
3.   package squareRoot

   object squareRoot {
     def squareRoot(n: Int): Float =  {
       var x: Float = n
       var y: Float = 1
       var e: Double = 0.000001

       while( (x - y) > e) {
             x = (x + y) / 2
             y = n/x
       }
       return x
     }

     def main(args:Array[String]) :Unit = {
      var n: Int = 50
       println("Square Root of "+n+" is "+squareRoot(n))
     }
   }
```

4. Now right click on the scala object ‘squareRoot’ and select Run command.



5. We can observe that the output ‘Square Root of 50 is 7.071068’.

1. Now we run the same application with ‘n’ value as 4.



2. Here we can observe that the output of the application 'Square Root of 4 is 2.0'.

Explanation:

n = 4 /* 'n' itself is used for initial approximation. */
Initialize x = 4, y = 1.
Next Approximation x = (x + y) / 2 (= 2.5000000),
Y = n / x (= 1.600000)

Next Approximation x = 2.050000,
Y = 1.951220

Next Approximation x = 2.000610,
Y = 1.999390

Next Approximation x = 2.000000,
Y = 2.000000

Terminate as (x – y) > e now.