Task 1

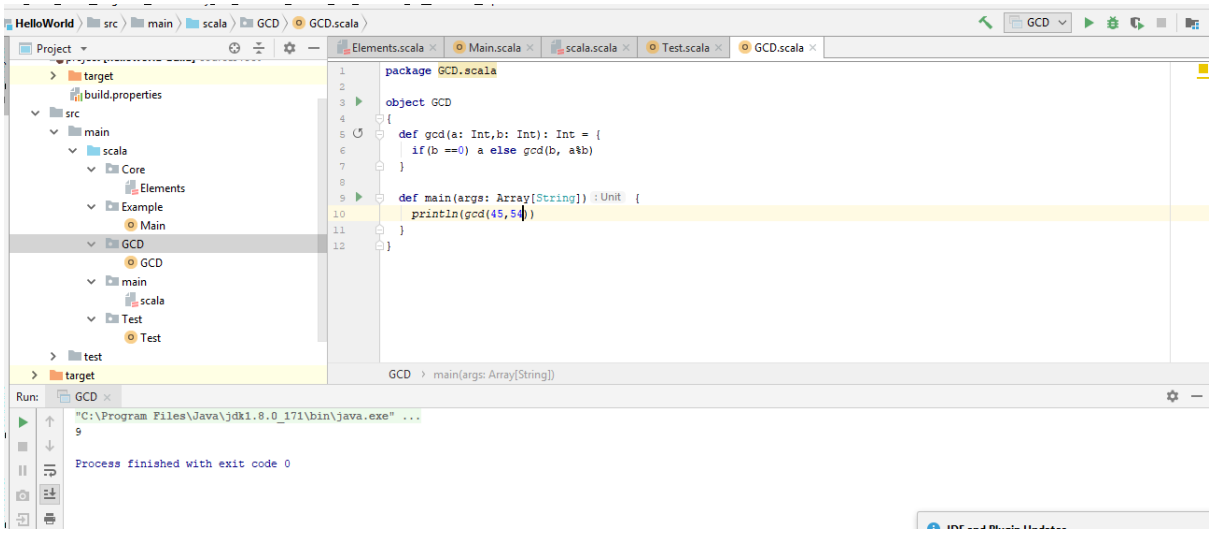
Create a Scala application to find the GCD of two numbers

Program

```
package GCD.scala

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(45,54))
  }
}
```




```
scala> def gcd(a: Int, b: Int): Int = if (b == 0) a else gcd(b, a % b)
gcd: (a: Int, b: Int)Int

scala> gcd(14,21)
res11: Int = 7

scala> gcd(45,54)
res12: Int = 9

scala>
```

Task 2:

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

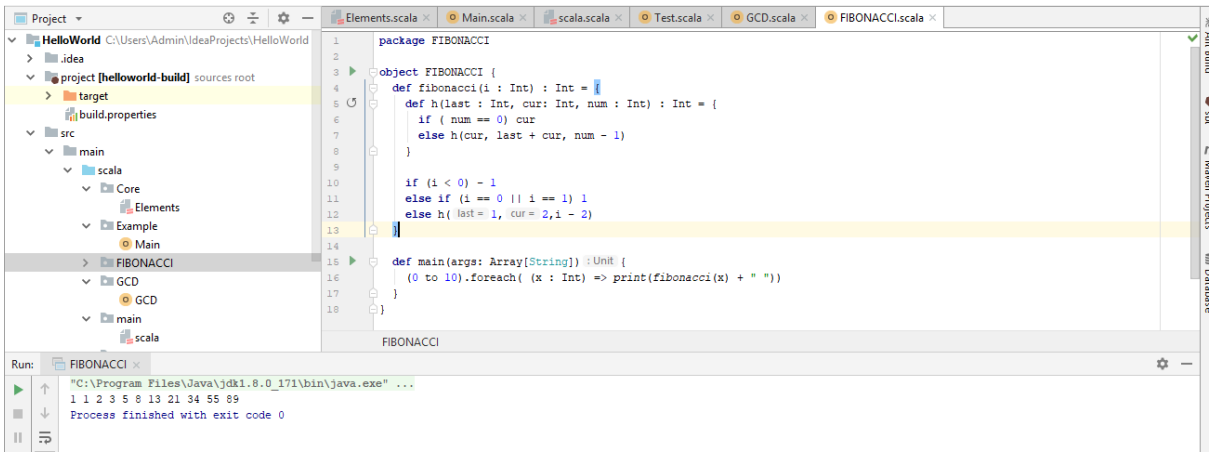
```scala
package FIBONACCI

object FIBONACCI {
  def fibonacci(i : Int) : Int = {
    def h(last : Int, cur: Int, num : Int) : Int = {
      if ( num == 0) cur
      else h(cur, last + cur, num - 1)
    }

    if (i < 0) - 1
    else if (i == 0 || i == 1) 1
    else h(1,2,i - 2)
  }

  def main(args: Array[String]){
    (0 to 10).foreach( (x : Int) => print(fibonacci(x) + " "))
  }
}
```

function using recursion




```
scala> val fibs: Stream[Int] = 1 #:: fibs.scanLeft(1)(_ + _)
fibs: Stream[Int] = Stream(1, ?)

scala>  fibs take 12 toList
<console>:13: warning: postfix operator toList should be enabled
by making the implicit value scala.language.postfixOps visible.
This can be achieved by adding the import clause 'import scala.language.postfixOps'
or by setting the compiler option -language:postfixOps.
See the Scaladoc for value scala.language.postfixOps for a discussion
why the feature should be explicitly enabled.
       fibs take 12 toList
                  ^
res12: List[Int] = List(1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144)

scala>
```

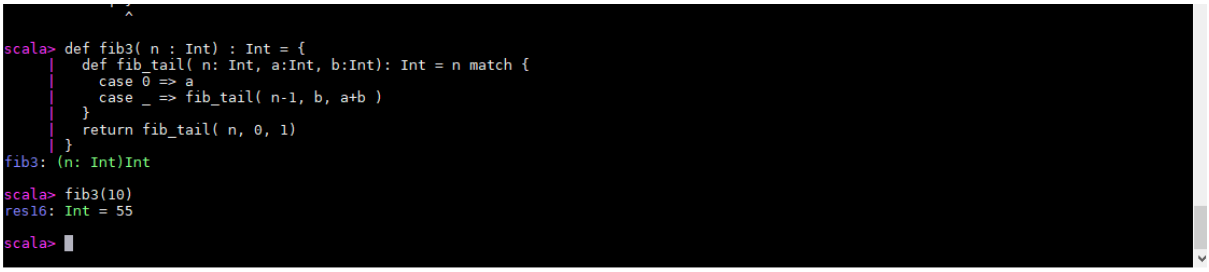
Write a Scala application to find the Nth digit in the sequence.

➢ Write the function using standard for loop

```
scala> def fib2( n : Int ) : Int = {
     |   var a = 0
     |   var b = 1
     |   var i = 0
     |
     |   while( i < n ) {
     |     val c = a + b
     |     a = b
     |     b = c
     |     i = i + 1
     |   }
     |   return a
     | }
fib2: (n: Int)Int

scala> fib2(10)
res13: Int = 55
```

➢ Write the function using recursion



```
scala> def fib3( n : Int) : Int = {
     |   def fib_tail( n: Int, a:Int, b:Int): Int = n match {
     |     case 0 => a
     |     case _ => fib_tail( n-1, b, a+b )
     |   }
     |   return fib_tail( n, 0, 1)
     | }
fib3: (n: Int)Int

scala> fib3(10)
res16: Int = 55

scala>
```

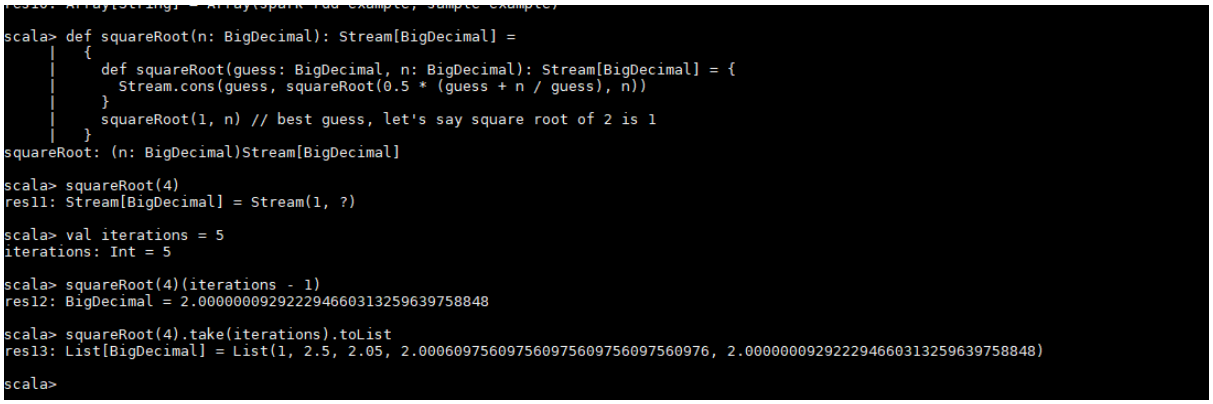
Task 3

Find square root of number 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.

a) Get the next approximation for root using average of x and y

b) Set y = n/x



```
scala> def squareRoot(n: BigDecimal): Stream[BigDecimal] =
     |   {
     |     def squareRoot(guess: BigDecimal, n: BigDecimal): Stream[BigDecimal] = {
     |       Stream.cons(guess, squareRoot(0.5 * (guess + n / guess), n))
     |     }
     |     squareRoot(1, n) // best guess, let's say square root of 2 is 1
     |   }
squareRoot: (n: BigDecimal)Stream[BigDecimal]

scala> squareRoot(4)
res11: Stream[BigDecimal] = Stream(1, ?)

scala> val iterations = 5
iterations: Int = 5

scala> squareRoot(4)(iterations - 1)
res12: BigDecimal = 2.000000092922294660313259639758848

scala> squareRoot(4).take(iterations).toList
res13: List[BigDecimal] = List(1, 2.5, 2.05, 2.000609756097560975609756097560976, 2.000000092922294660313259639758848)

scala>
```