***Task 1***

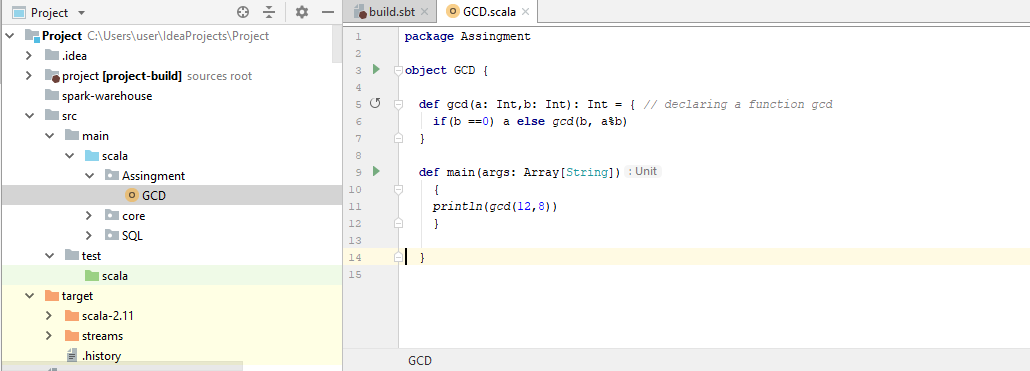
***Create a Scala application to find the GCD of two numbers***

Greatest Common Divisor (GCD) of two or more integers, which are not all zero, is the largest positive integer that divides each of the integers.

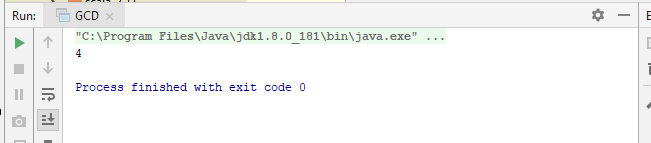
For example, the **gcd** of **8 and 12 is 4.**

## Scala Application using IntelliJ

In the below scala code, we are going to find the gcd of the two numbers 12 and 8.

****

Result



***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***

A Fibonacci series (starting from 1) written in order without any spaces in between, thusproducing 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

Before going in to the tasks, we will just see an over view that what is he Fibonacci number,

# The Fibonacci sequence is the series of numbers,

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

The next number is found by adding up the two numbers before it.

The 2 is found by adding the two numbers before it (1+1)

The 3 is found by adding the two numbers before it (1+2),

And the 5 is (2+3),

And so on!

Example: the next number in the sequence above is 21+34 = 55

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| xn = | 0 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | 144 | 233 | 377 | 610 | 987 |

Formula,

**xn = xn-1 + xn-2**

Example,

The 8th term is the 7th term plus the 6th term: X8 = X7+X6

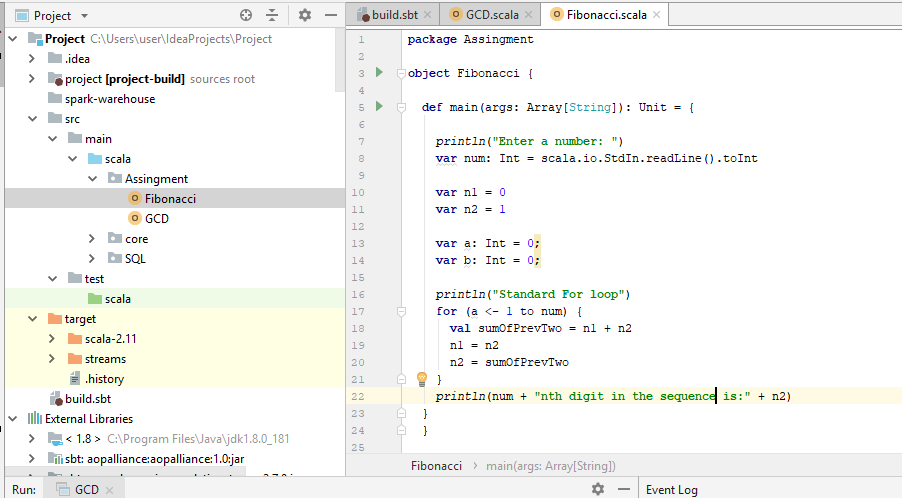
From the above table,

The 8th term is 21, hence the 7th term 21+the 6th term 13 = 34.

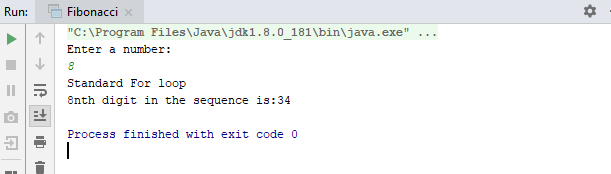
Code:

**package** Assingment  
  
**object** Fibonacci {  
  
 **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)  
 }  
 }

Screeshot:



Result

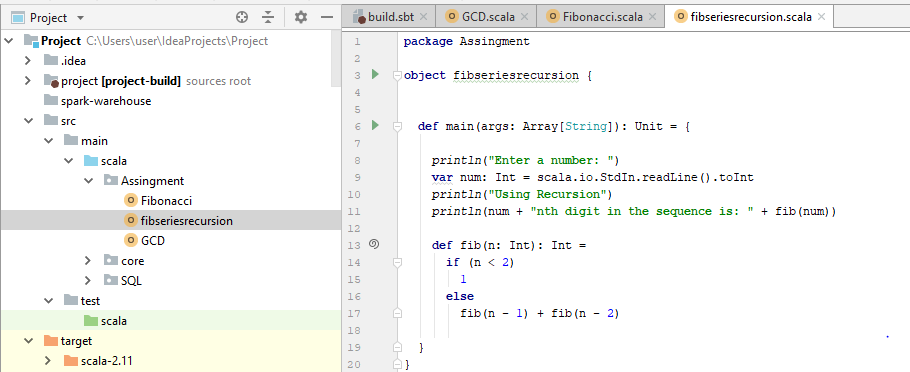


***Write the function using recursion***

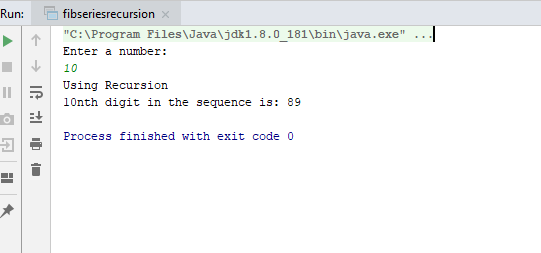
Code logic

**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)

Screenshot



Result



***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.***

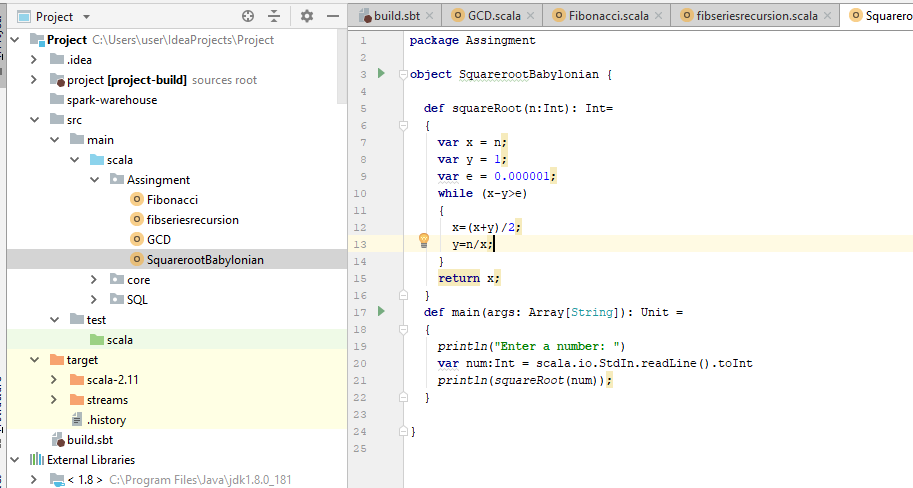
***a) Get the next approximation for root using average of x and y***

***b) Set y = n/x***

The Babylonian method for finding square roots involves dividing and averaging, over and over, to obtain a more accurate solution with each repeat of the process. Step 2: Divide your original number by your guess. Step 3: Find the average of these numbers. Step 4: Use this average as your next guess.

**package** Assingment  
  
**object** SquarerootBabylonian {  
  
 **def** squareRoot(n:Int): Int=  
 {  
 **var** x = n;  
 **var** y = 1;  
 **var** e = 0.000001;  
 **while** (x-y>e)  
 {  
 x=(x+y)/2;  
 y=n/x;  
 }  
 **return** x;  
 }  
 **def** main(args: Array[String]): Unit =  
 {  
 *println*(**"Enter a number: "**)  
 **var** num:Int = scala.io.StdIn.readLine().toInt  
 *println*(*squareRoot*(num));  
 }  
  
}

Screenshot



Result

