Big Data Engineering with Hadoop & Spark

Assignment on Scala Basics

AcadGild | Big Data Assignment Report |

Session 15: Assignment 15.1

This assignment is aimed at consolidating the concepts that was learnt during the Scala Basics session of the course.



***1***

AcadGild | Big Data Assignment Report |

Problem Statement

Task 1:

Create a Scala application to find the GCD of two numbers

**Solution:**

To find the GCD of two numbers I have used the below logic:

− If either 1st or 2nd number is 0, then other number is the Greatest Common Divisor.

− Else call the GCD function again by sending 2nd number as 1st number and difference between 2 numbers as 2nd number.

− This in turn checks for the If clause again.

**Code:**

class appGCD {

/\*\*\*Method to find the GCD of 2 numbers\*\*\*/ def gcd(a: Int, b: Int): Int = {

if(b == 0) a else gcd(b, a%b)

}

/\*\*\*Method to display list of choices to the user\*\*\*/ def OptionsList(): Unit = {

println("\nGCD of 2 numbers")

println("--------------------")

println("\nSelect one of the following:")

println("1. Compute GCD with command line argument")

println("2. Compute GCD with standard input argument")

println("\nEnter your choice (1 or 2): ")

}

}

object appGCD {

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

var wish =""

/\*\*\*Creating the instance of the appGCD class\*\*\*/



***2***

AcadGild | Big Data Assignment Report |

val aGCD = new appGCD()

do {

/\*\*\*Calling the method to display the list of options to the user\*\*\*/ aGCD.OptionsList()

val choice = scala.io.StdIn.readLine()

/\*\*\*Find GCD from CommandLine Input Arguments (Get from the user)\*\*\*/

if (choice.toInt == 1) {

val input1 = args(0).toInt

val input2 = args(1).toInt

println("\nCMD: GCD of ${input1} and ${input2} is : " + aGCD.gcd(input1, input2))

}

/\*\*\*Find GCD from Standard Input Arguments (Get from the user)\*\*\*/ else if (choice.toInt == 2) {

println("Enter the 1st number : ")

val inp1 = scala.io.StdIn.readLine().toInt

println("Enter the 2nd number : ")

val inp2 = scala.io.StdIn.readLine().toInt

println("STDIN: GCD of ${inp1} and ${inp2} is : " + aGCD.gcd(inp1, inp2))

}

else {

println("Invalid choice!")

}

/\*\*\*DoWhile loop conditional variable\*\*\*/ println("\nDo you wish to continue? (Y/N) : ") wish = scala.io.StdIn.readLine().toUpperCase

println("--------------------------------------------------------------------------\n")

}

while (wish.equals("Y"))

}

}



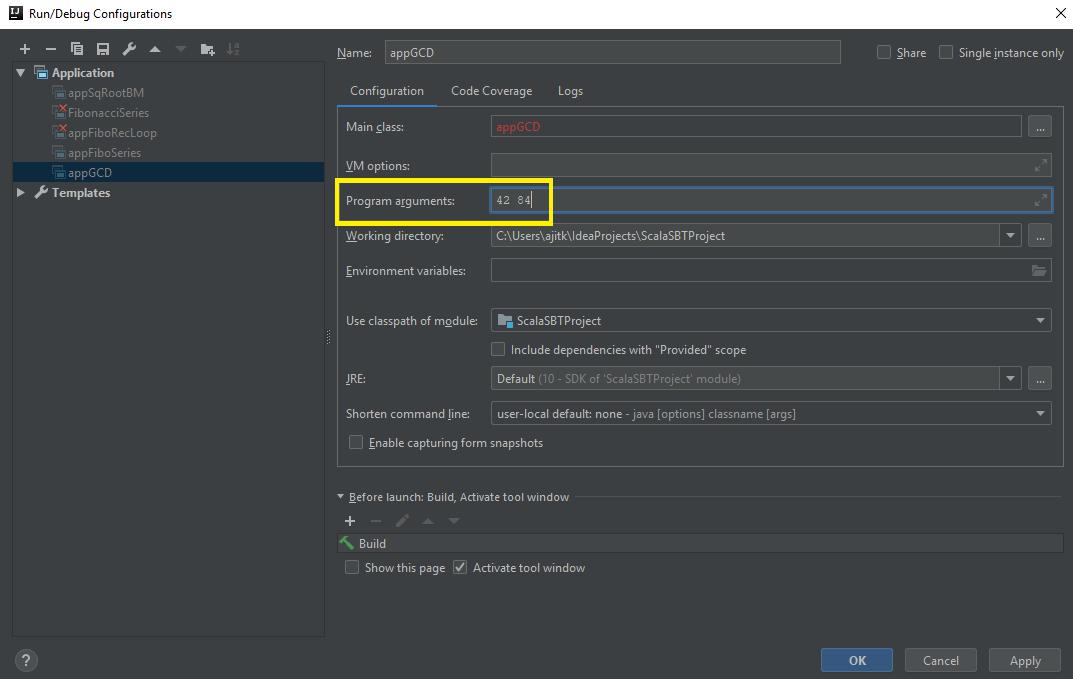
***3***

AcadGild | Big Data Assignment Report |

**Output:**

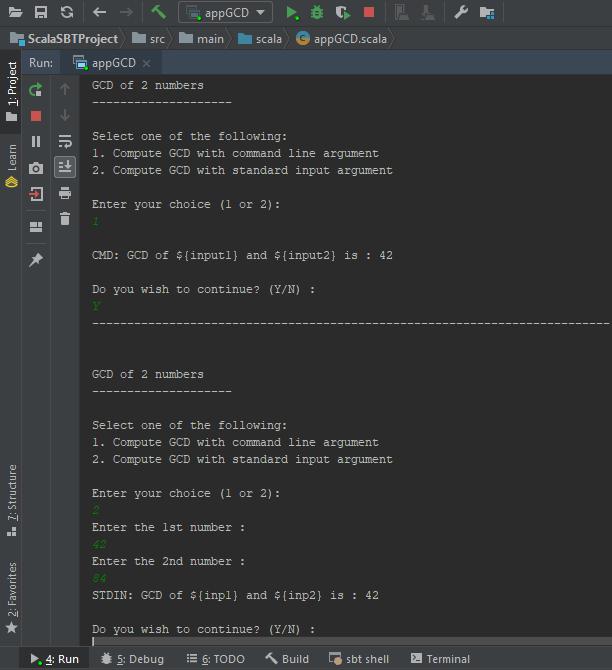
In the above code, I have taken the input for the GCD function in 2 ways:

1. From the Command Line Arguments
   1. For this, use “Edit Configuration” option
   2. Provide values in “Program arguments” section of dialogue box
2. From the User through the Standard Input



***4***

AcadGild | Big Data Assignment Report |



***5***

AcadGild | Big Data Assignment Report |

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. o Write the function using standard for loop

o Write the function using recursion

**Solution:**

To find the Fibonacci Series I have used two methods:

− Using a Standard FOR Loop. This is achieved by the method LoopFibo(digits, nthdigit)

− Using Recursion. This is achieved by the method recFibonacci(digits, nthdigit)

The @tailrec annotation in the code is used to indicate that this is an optimized version of the function to find the Fibonacci series.

**Code:**

import scala.annotation.tailrec

object appFiboSeries {

def recFibonacci(n: Int, nth: Int): Unit = {

var concat\_result = "1"

/\*\*\*Method to find out the Fibonacci Series using Recursion\*\*\*/

@tailrec def fiboRecursive(n: Int, prev: BigInt = 0, next: BigInt = 1): BigInt = n match {

case 0 => prev

case 1 => next

case \_ =>

concat\_result = concat\_result + (prev + next)

fiboRecursive(n - 1, next, next + prev)

}

fiboRecursive(n)

get\_nthchar\_and\_print(n, concat\_result, nth)

}

/\*\*\*Method to find out the Fibonacci Series using For Loop\*\*\*/ def LoopFibo(n: Int, nth: Int): Unit = {

var concat\_result = "1"



***6***

AcadGild | Big Data Assignment Report |

if (n < 2) {

println(n)

}

else {

var result: BigInt = 0

var n1: BigInt = 0

var n2: BigInt = 1

for (i <- 1 until n) {

result = n1 + n2

n1 = n2

n2 = result

concat\_result = concat\_result + result

}

get\_nthchar\_and\_print(n, concat\_result, nth)

result

}

}

/\*\*\*Method to display Nth character in the Fibonacci Sequence\*\*\*/ def get\_nthchar\_and\_print(n: Int, seq: String, nth: Int): Unit = {

println(s"The Fibonacci Series ($n): " + seq)

println(s"The digit at the place $nth of Fibo Sequence ($n): " + seq.charAt(nth -1).toChar)

}

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

var wish = ""

println("Fibonacci Series")

println("------------------------------------------------------------")

do {

println("Select one of the following:")

println("1. Find Nth digit in the Fibonacci Series using For Loop")

println("2. Find Nth digit in the Fibonacci Series using Recursion")

println("Enter your choice (1 or 2): ")



***7***

AcadGild | Big Data Assignment Report |

var choice = scala.io.StdIn.readLine()

println("Enter the number of digits for Fibonacci Sequence: ")

var digits: Int = scala.io.StdIn.readLine().toInt

println("Enter the Nth digit to be found in the Fibonacci Sequence: ")

var nthFind: Int = scala.io.StdIn.readLine().toInt

println("--------------------------------------------------------------------------")

if (choice.toInt == 1) {

/\*\*\*Call to method "LoopFibo" to find out the Fibonacci Series using For Loop\*\*\*/

println(s"Fibonacci Series using For Loop:")

println("----------------------------------")

LoopFibo(digits, nthFind)

}

/\*\*\*Call to method "recFibonacci" to find out the Fibonacci Series using Recursion\*\*\*/

else if (choice.toInt == 2) { println(s"Fibonacci Series using Recursion:") println("-----------------------------------")

recFibonacci(digits, nthFind)

}

else {

println(s"Invalid Choice!")

}

println("Do you wish to continue? (Y/N): ")

/\*\*\*Do-While Loop for condition variable\*\*\*/ wish = scala.io.StdIn.readLine().toUpperCase

println("--------------------------------------------------------------------------\n")

}

while (wish.equals("Y"))

}

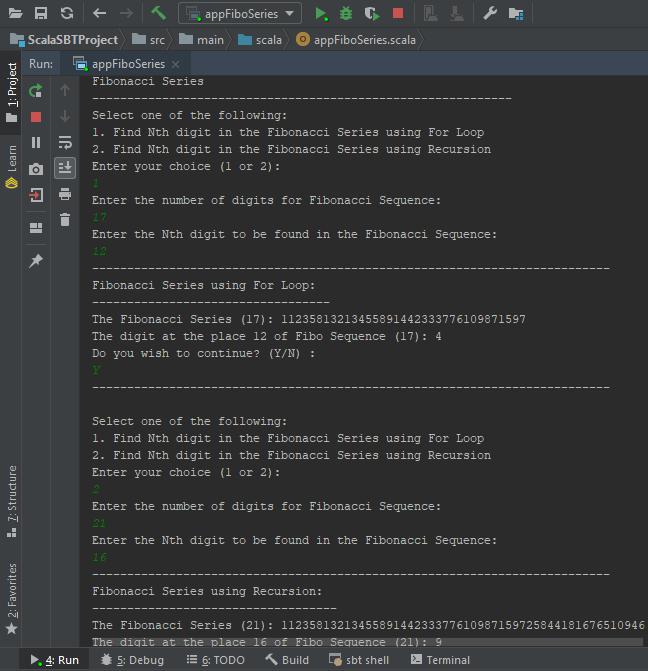
}



***8***

AcadGild | Big Data Assignment Report |

**Output:**



***9***

AcadGild | Big Data Assignment Report |

Task 3:

− Find square root of number using Babylonian method.

− Start with an arbitrary positive start value x (the closer to the root, the better).

− Initialize y = 1.

− Do following until desired approximation is achieved.

1. Get the next approximation for root using average of x and y o Set y = n/x

**Solution:**

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.

**Code:**

/\*\*\*Dividing and Averaging Method to calculate square root of a number\*\*\*/ object appSqRootBM {

/\*\*\*Function to return square root of a number using Babylonian Method\*\*\*/ def squareRootBM(num: Int): Float = {

/\*\*\*Arbitrary positive value x from the user\*\*\*/ var x: Float = num

/\*\*\*Initialize y\*\*\*/

var y: Float = 1

/\*\*\*e decides the accuracy level\*\*\*/

/\*\*\*This is checked when we aren't sure if the number is a perfect square\*\*\*/ val e: Double = 0.000001

/\*\*\*Performs division and averaging until the accuracy level\*\*\*/ while(x - y > e) {

x = (x + y) / 2

y = num / x

}

x /\*\*\*Returns the square root value\*\*\*/

}

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



***10***

AcadGild | Big Data Assignment Report |

var wish = ""

println("\nSquare Root using Babylonian Method")

println("-------------------------------------")

do {

println("\nEnter the number: ")

var input = scala.io.StdIn.readLine().toInt

/\*\*\*Calls the function to calculate Square Root using Babylonian Method\*\*\*/ println("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_") println(s"Square Root of $input is ${squareRootBM(input)}")

println("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

println("\n\n\nDo you wish to continue? (Y/N) : ")

/\*\*\*Do-While Loop for condition variable\*\*\*/ wish = scala.io.StdIn.readLine().toUpperCase

println("--------------------------------------------------------------------------\n")

}

while (wish.equals("Y"))

}

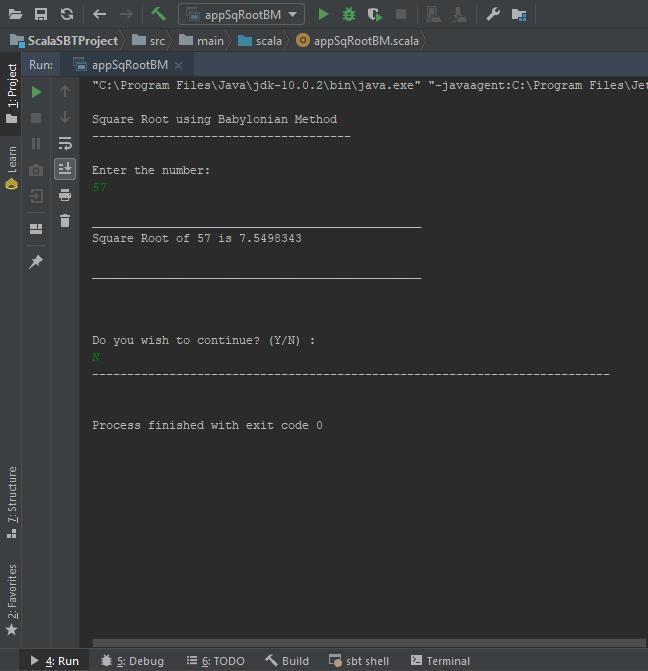
}



***11***

AcadGild | Big Data Assignment Report |

**Output:**



***Note:***

*Scala code files for each application has been provided separately along with this assignment report.*



***12***