Scala practical 10 – 22001842

01.

```
object O1{
  class Rational(numerator: Int, denominator: Int) {
    require(denominator != 0, "Denominator cannot be zero.")
    def unary_- : Rational = new Rational(-numerator, denominator)

    def printRational(): Unit = {
      println(s"$numerator/$denominator")
    }
  }

  def main(args: Array[String]): Unit = {
    val x = new Rational(3, 4)
    x.printRational()
    val y = -x
    y.printRational()
  }
}
```

1.scala 2.scala 3.scala 4.scala 5.scala

1.scala > {} O1 > main > x

```scala
run | debug
object O1{
    class Rational(numerator: Int, denominator: Int) {
        require(denominator != 0, "Denominator cannot be zero.")
        def unary_- : Rational = new Rational(-numerator, denominator)

        def printRational(): Unit = {
            println(s"$numerator/$denominator")
        }
    }

    def main(args: Array[String]): Unit = {
        val x = new Rational(3, 4)
        x.printRational()
        val y = -x
        y.printRational()
    }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
PS C:\Users\hp\Desktop\Scala_practical_08> scala "c:\Users\hp\Desktop\Scala_pr
3/4
-3/4
PS C:\Users\hp\Desktop\Scala_practical_08>
```

02)

```scala
object O2{
  class Rational(val numerator: Int, val denominator: Int) {
    require(denominator != 0, "Denominator cannot be zero.")


    def sub(that: Rational): Rational = {
      val newNumerator = (this.numerator * that.denominator) - (that.numerator * this.denominator)
      val newDenominator = this.denominator * that.denominator
      new Rational(newNumerator, newDenominator).simplify
    }


    private def simplify: Rational = {
```

```
        val gcd = greatestCommonDivisor(numerator, denominator)
        new Rational(numerator / gcd, denominator / gcd)
    }

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

    def printRational(): Unit = {
        println(s"$numerator/$denominator")
    }
  }
  def main(args: Array[String]):Unit={
    val x = new Rational(3, 4)
    val y = new Rational(5, 8)
    val z = new Rational(2, 7)

    val result = y.sub(z)
    result.printRational()
  }
}
```

1.scala 2.scala 3.scala 4.scala 5.scala

2.scala > {} O2 > Rational

```scala
object O2{
    class Rational(val numerator: Int, val denominator: Int) {
        private def simplify: Rational = {
        }

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

        def printRational(): Unit = {
            println(s"$numerator/$denominator")
        }
    }
    def main(args: Array[String]):Unit={
        val x = new Rational(3, 4)
        val y = new Rational(5, 8)
        val z = new Rational(2, 7)

        val result = y.sub(z)
        result.printRational()
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
PS C:\Users\hp\Desktop\Scala_practical_08> scala "c:\Users\hp\Desktop\Scala_pr
19/56
```

03)

```scala
import scala.collection.mutable.Map

object O3{


  class Account(var balance:Double, var accountNo:Int){


    def Deposit(amount:Double):Unit={

      if (amount > 0) {

        this.balance = this.balance + amount;

        println(s"The new balance is =${this.balance}");

      }
```

```
    else{
        println("Deposit amount can not be less that or equal to 0")
    }
}

def Withdraw(amount:Double):Unit={
    if (amount > 0 && amount <= this.balance) {
        this.balance = this.balance - amount;
        println(s"The new balance is =${this.balance}");
    }
    else if (amount > this.balance){
        println("Insufficient funds for this withdrawal")
    }
    else{
        println("Withdrawal amount must be greater than zero")
    }
}

def Transfer(account:Account, amount:Double):Unit={
    if (amount > 0 && amount <= balance) {
        account.balance = account.balance + amount;
        println(s"Transfer to ${account.accountNo} successful")
    }
    else if (amount > this.balance){
        println("Insufficient funds for this transfer")
    }
    else{
        println("transfer amount must be greater than zero")
    }
```

```
    }
  }

  def main(args: Array[String]):Unit = {
    val accounts: Map[Int, Account] = Map()

    val account1 = new Account(1000, 123)
    val account2 = new Account(500, 456)

    accounts += (123 -> account1)
    accounts += (456 -> account2)

    val fromAccountNo = 123
    val toAccountNo = 456
    val transferAmount = 300

    if (accounts.contains(fromAccountNo) && accounts.contains(toAccountNo)) {
      val fromAccount = accounts(fromAccountNo)
      val toAccount = accounts(toAccountNo)
      fromAccount.Transfer(toAccount, transferAmount)
    }
    else {
      println("One or both of the accounts were not found.")
    }
  }
}
```

```scala
import scala.collection.mutable.Map
object O3{

    class Account(var balance:Double, var accountNo:Int){

        def Deposit(amount:Double):Unit={
            if (amount > 0) {
                this.balance = this.balance + amount;
                println(s"The new balance is =${this.balance}");
            }
            else{
                println("Deposit amount can not be less that or equal t
            }
        }

        def Withdraw(amount:Double):Unit={
            if (amount > 0 && amount <= this.balance) {
                this.balance = this.balance - amount;
                println(s"The new balance is =${this.balance}");
```

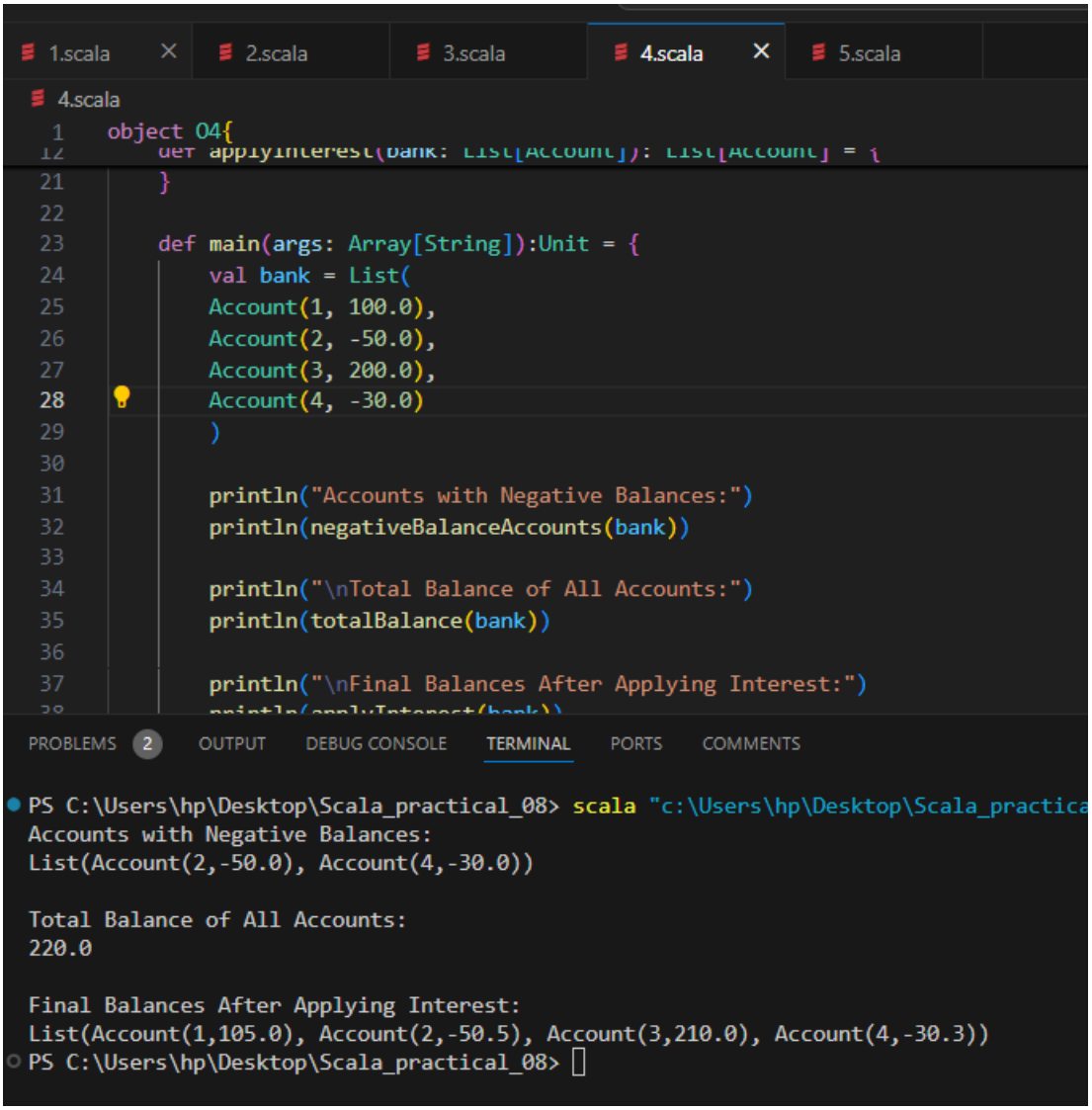
```
PS C:\Users\hp\Desktop\Scala_practical_08> scala "c:\Users\hp\Desktop\Scala_p
Transfer to 456 successful
PS C:\Users\hp\Desktop\Scala_practical_08>
```

04)

```scala
object O4{

   case class Account(accountNumber: Int, balance: Double)


   def negativeBalanceAccounts(bank: List[Account]): List[Account] = {

   bank.filter(_.balance < 0)

   }


   def totalBalance(bank: List[Account]): Double = {

   bank.map(_.balance).sum
```

```
}

def applyInterest(bank: List[Account]): List[Account] = {
bank.map { account =>
   val newBalance = if (account.balance > 0) {
   account.balance * 1.05
   } else {
   account.balance * 1.01
   }
   account.copy(balance = newBalance)
}
}

def main(args: Array[String]):Unit = {
   val bank = List(
   Account(1, 100.0),
   Account(2, -50.0),
   Account(3, 200.0),
   Account(4, -30.0)
   )

   println("Accounts with Negative Balances:")
   println(negativeBalanceAccounts(bank))

   println("\nTotal Balance of All Accounts:")
   println(totalBalance(bank))

   println("\nFinal Balances After Applying Interest:")
   println(applyInterest(bank))
```

```
    }
}
```



05)

```
object O5{
    def countLetterOccurrences(words: List[String]): Int = {
        val lengths = words.map(_.length)
        lengths.reduce(_ + _)
    }
```

```
  def main(args:Array[String]) :Unit= {
    val words = List("apple", "banana", "cherry", "date")
    val totalCount = countLetterOccurrences(words)
    println(s"Total count of letter occurrences: $totalCount")
  }
}
```

1.scala 2.scala 3.scala 4.scala 5.scala

5.scala > {} O5 > main > words

```scala
object O5{
    def countLetterOccurrences(words: List[String]): Int = {
        val lengths = words.map(_.length)
        lengths.reduce(_ + _)
    }

    def main(args:Array[String]) :Unit= {
        val words = List("apple", "banana", "cherry", "date")
        val totalCount = countLetterOccurrences(words)
        println(s"Total count of letter occurrences: $totalCount")
    }
}
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
PS C:\Users\hp\Desktop\Scala_practical_08> scala "c:\Users\hp\Desktop\Scala_practical_
Total count of letter occurrences: 21
PS C:\Users\hp\Desktop\Scala_practical_08>
```