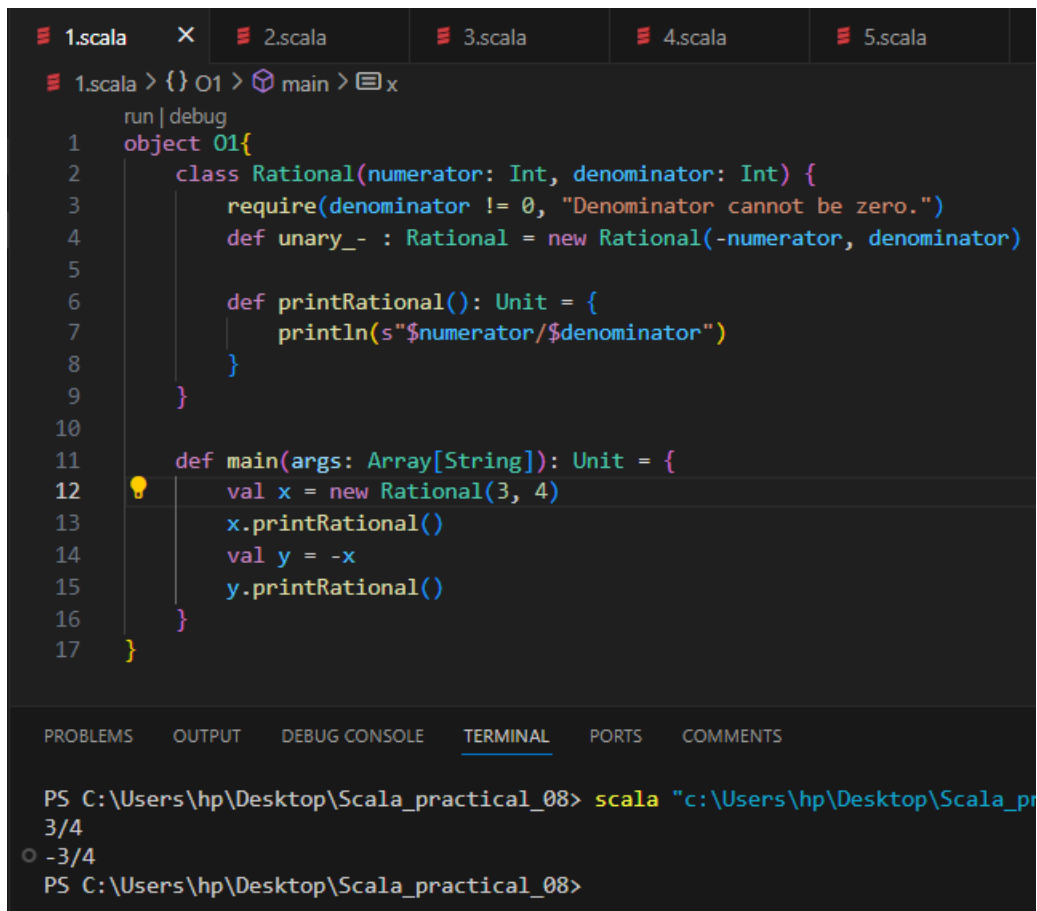


Scala practical 10 – 22001842

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 > {} O1 > main > x
run | debug
1  object O1{
2      class Rational(numerator: Int, denominator: Int) {
3          require(denominator != 0, "Denominator cannot be zero.")
4          def unary_- : Rational = new Rational(-numerator, denominator)
5
6          def printRational(): Unit = {
7              println(s"$numerator/$denominator")
8          }
9      }
10
11     def main(args: Array[String]): Unit = {
12         val x = new Rational(3, 4)
13         x.printRational()
14         val y = -x
15         y.printRational()
16     }
17 }
```

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)

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)
    }

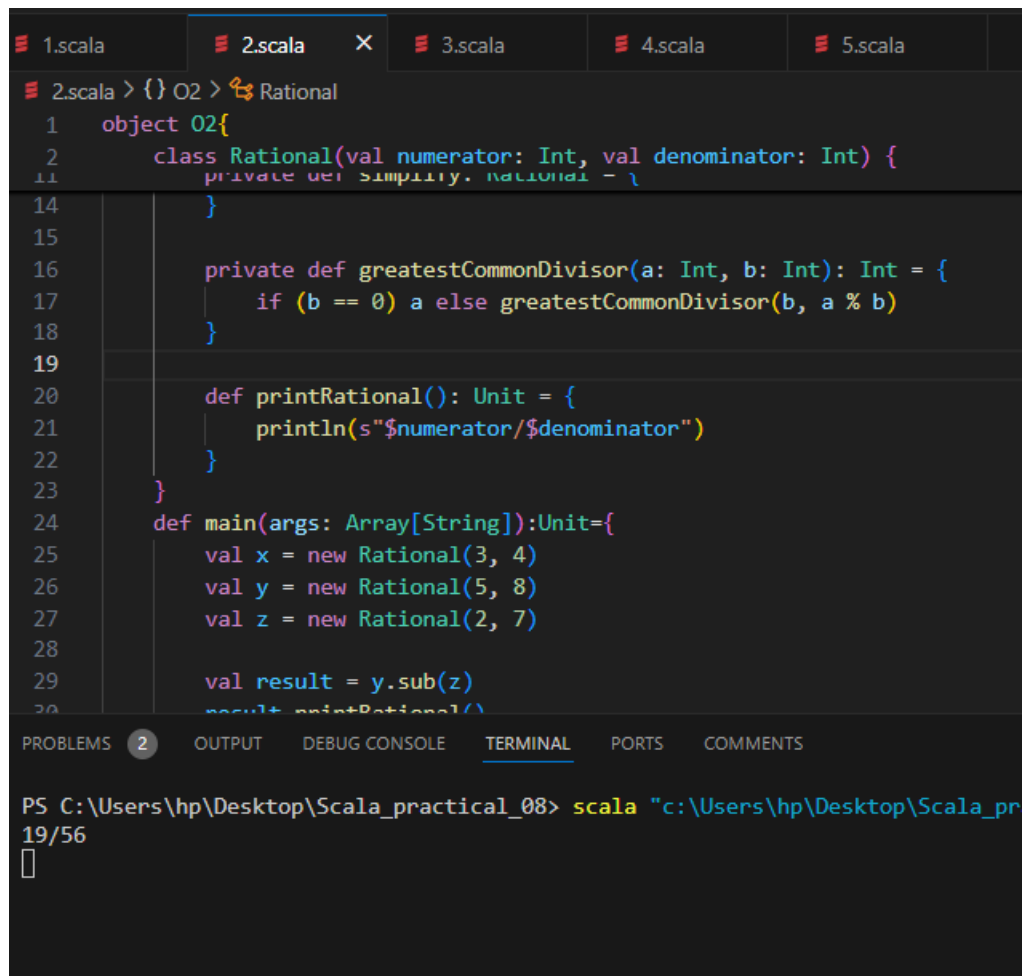
    private def greatestCommonDivisor(a: Int, b: Int): Int = {
        if (b == 0) a else greatestCommonDivisor(b, a % b)
    }

    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 X 3.scala 4.scala 5.scala
2.scala > {} O2 > Rational
1  object O2{
2      class Rational(val numerator: Int, val denominator: Int) {
11         private def simplify: Rational = {
14     }
15
16     private def greatestCommonDivisor(a: Int, b: Int): Int = {
17         if (b == 0) a else greatestCommonDivisor(b, a % b)
18     }
19
20     def printRational(): Unit = {
21         println(s"$numerator/$denominator")
22     }
23 }
24 def main(args: Array[String]):Unit={
25     val x = new Rational(3, 4)
26     val y = new Rational(5, 8)
27     val z = new Rational(2, 7)
28
29     val result = y.sub(z)
30     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)

```
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 than 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.")
    }
}
}

```

```
View Go Run Terminal Help
1.scala 2.scala 3.scala X 4.scala 5.scala
3.scala > {} O3 > Account > Deposit
1  import scala.collection.mutable.Map
2  object O3{
3
4      class Account(var balance:Double, var accountNo:Int){
5
6          def Deposit(amount:Double):Unit={
7              if (amount > 0) {
8                  this.balance = this.balance + amount;
9                  println(s"The new balance is =${this.balance}");
10             }
11             else{
12                 println("Deposit amount can not be less that or equal t
13             }
14         }
15
16         def Withdraw(amount:Double):Unit={
17             if (amount > 0 && amount <= this.balance) {
18                 this.balance = this.balance - amount;
19                 println(s"The new balance is =${this.balance}");
20             }
21         }
22     }
23 }

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
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)

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
}
```

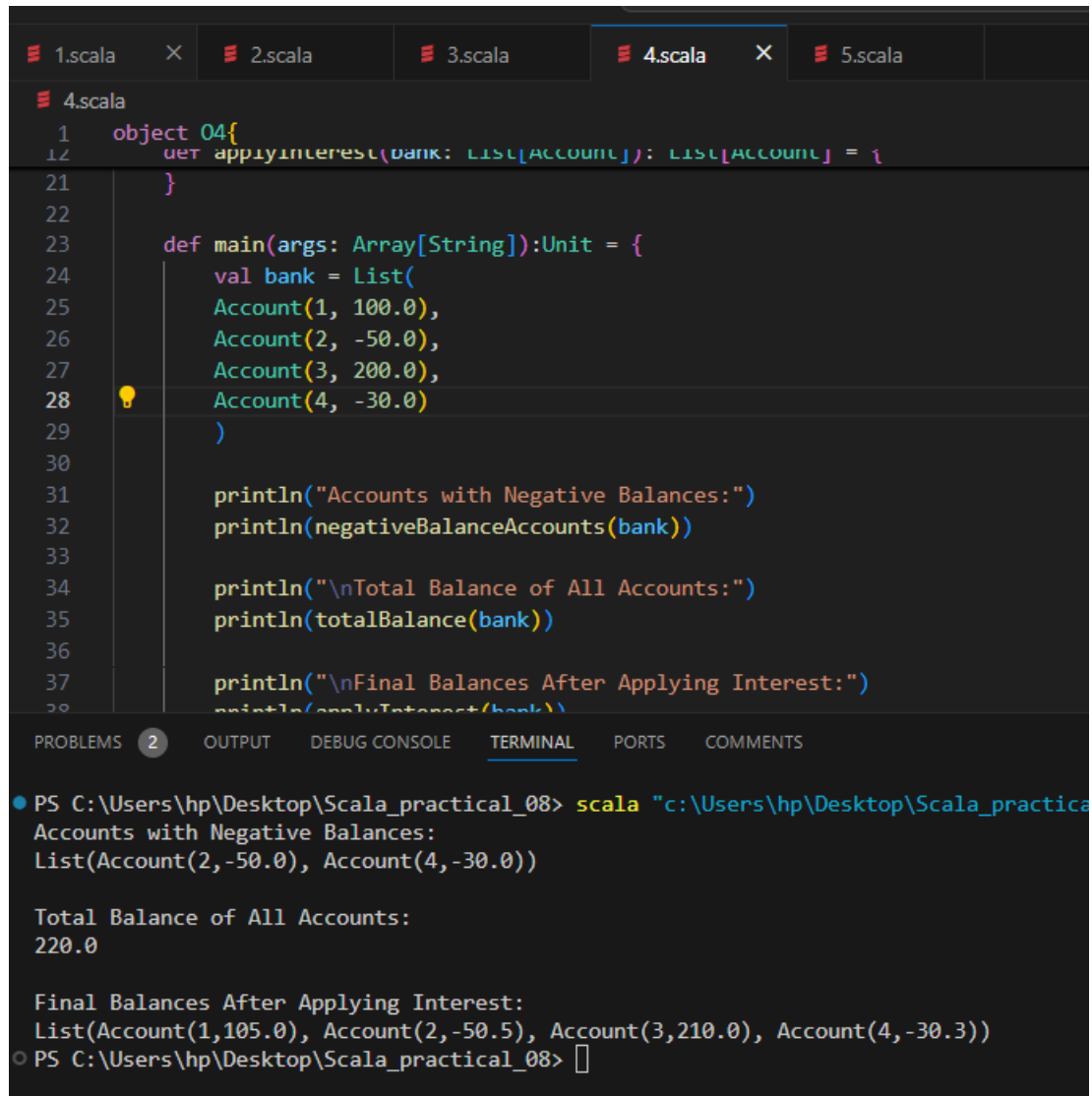
```
}
```

```
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))
```



```
}  
}
```



The screenshot shows an IDE with five tabs: 1.scala, 2.scala, 3.scala, 4.scala (active), and 5.scala. The code in 4.scala defines an object O4 with a method applyInterest and a main function. The main function creates a list of accounts, prints negative balances, calculates the total balance, and prints final balances after interest. The terminal output shows the execution of the main function, displaying the negative balances, the total balance, and the final balances after interest.

```
1  object O4{  
12  def applyInterest(bank: List[Account]): List[Account] = {  
21  }  
22  
23  def main(args: Array[String]):Unit = {  
24      val bank = List(  
25          Account(1, 100.0),  
26          Account(2, -50.0),  
27          Account(3, 200.0),  
28          Account(4, -30.0)  
29      )  
30  
31      println("Accounts with Negative Balances:")  
32      println(negativeBalanceAccounts(bank))  
33  
34      println("\nTotal Balance of All Accounts:")  
35      println(totalBalance(bank))  
36  
37      println("\nFinal Balances After Applying Interest:")  
38      println(applyInterest(bank))  
39  }
```

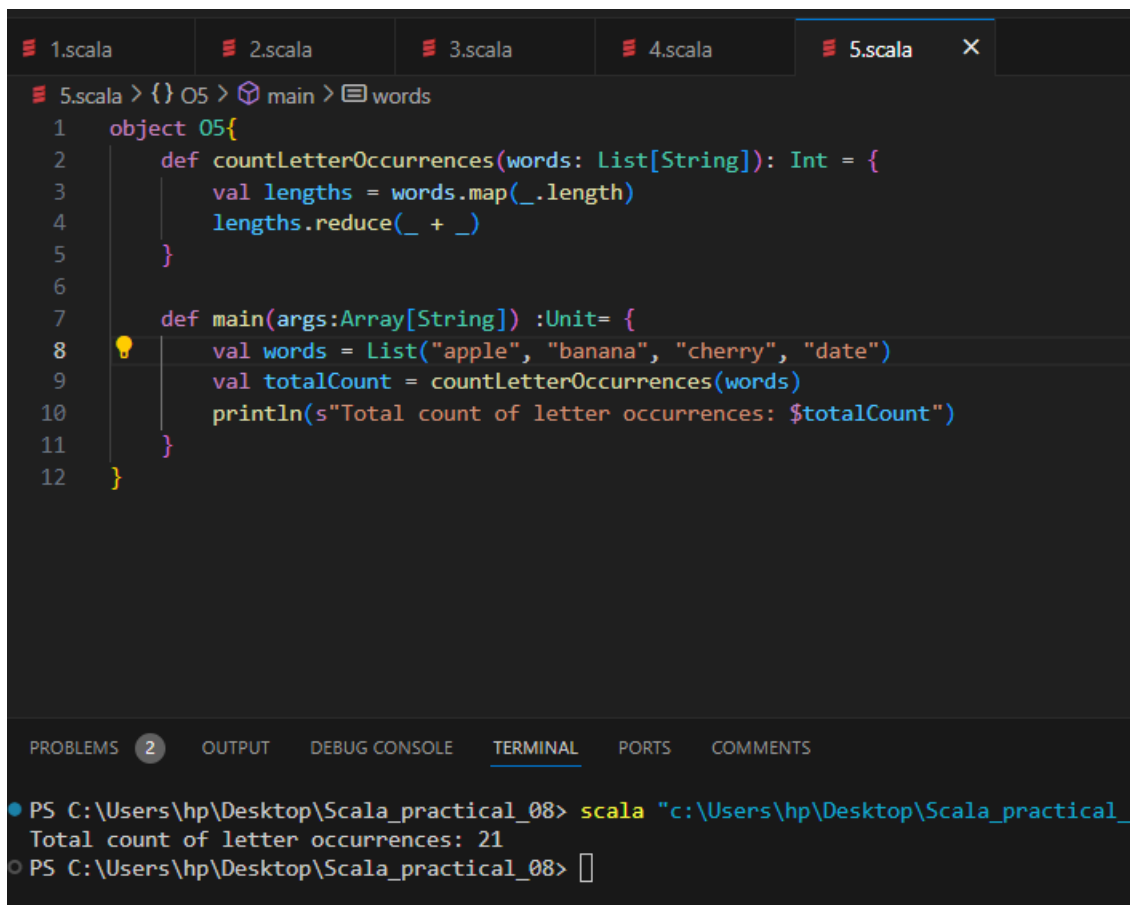
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
● PS C:\Users\hp\Desktop\Scala_practical_08> scala "c:\Users\hp\Desktop\Scala_practical_08\4.scala"  
Accounts with Negative Balances:  
List(Account(2,-50.0), Account(4,-30.0))  
  
Total Balance of All Accounts:  
220.0  
  
Final Balances After Applying Interest:  
List(Account(1,105.0), Account(2,-50.5), Account(3,210.0), Account(4,-30.3))  
○ PS C:\Users\hp\Desktop\Scala_practical_08> █
```

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")  
}  
}
```



The screenshot shows an IDE with a dark theme. At the top, there are tabs for 1.scala, 2.scala, 3.scala, 4.scala, and 5.scala. The 5.scala tab is active. The code editor displays the following Scala code:

```
1  object 05{  
2      def countLetterOccurrences(words: List[String]): Int = {  
3          val lengths = words.map(_.length)  
4          lengths.reduce(_ + _)  
5      }  
6  
7      def main(args:Array[String]) :Unit= {  
8          val words = List("apple", "banana", "cherry", "date")  
9          val totalCount = countLetterOccurrences(words)  
10         println(s"Total count of letter occurrences: $totalCount")  
11     }  
12 }
```

Below the code editor, there is a panel with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and COMMENTS. The TERMINAL tab is selected. It shows the following output:

```
PS C:\Users\hp\Desktop\Scala_practical_08> scala "c:\Users\hp\Desktop\Scala_practical_08\05.scala"  
Total count of letter occurrences: 21  
PS C:\Users\hp\Desktop\Scala_practical_08>
```