22001212 – Practical 10

01)

22001212_q1.scala > { } Rational > Rational

```scala
object Rational{
  class Rational(n:Int, d:Int) {
    require(d != 0, "Denominator cannot be zero");

    val numer: Int = if (d < 0) -n else n;
    val denom: Int = d.abs;

    def neg: Rational = new Rational(-numer, denom);

    override def toString: String = numer+"/"+denom;
  }

  def main(args:Array[String]):Unit={
      print("Enter a value for Numerator :");
      val num=scala.io.StdIn.readInt();
      print("Enter a value for Denominator :");
      val denom=scala.io.StdIn.readInt();
      val x = new Rational(num, denom);
      println(s"x : $x");
      println(s"Negation of x : ${x.neg}");
  }
}
```

```
Enter a value for Numerator :4
Enter a value for Denominator :5
x : 4/5
Negation of x : -4/5
PS C:\Users\User\Desktop\UCSC\2nd Year Sem-01\SCS 2204 - Functional Programming\Practical 10>
```

02)

22001212_q2.scala > {} Rational1

```scala
object Rational1{
    class Rational1(x:Int, y:Int) {
        def numer = x;
        def denom = y;
        def sub(r: Rational1) =new Rational1(numer * r.denom - r.numer * denom,denom * r.denom);
        override def toString = numer + "/" + denom;
    }

    def main(args:Array[String]):Unit={
        print("Enter the Numerator of the first rational number : ");
        val num1=scala.io.StdIn.readInt();
        print("Enter the Denomerator of the first rational number : ");
        val denom1=scala.io.StdIn.readInt();
        print("Enter the Numerator of the second rational number : ");
        val num2=scala.io.StdIn.readInt();
        print("Enter the Denomerator of the second rational number : ");
        val denom2=scala.io.StdIn.readInt();
        val r1=new Rational1(num1,denom1);
        val r2=new Rational1(num2,denom2);
        println(f"r1 = ${r1}");
        println(f"r2 = ${r2}");
        println("r1 - r2 = "+{r1.sub(r2)});
    }
}
```

```
Enter the Numerator of the first rational number : 5
Enter the Numerator of the first rational number : 5
Enter the Denomerator of the first rational number : 8
Enter the Numerator of the second rational number : 2
Enter the Denomerator of the second rational number : 7
r1 = 5/8
r2 = 2/7
r1 - r2 = 19/56
PS C:\Users\User\Desktop\UCSC\2nd Year Sem-01\SCS 2204 - Functional Programming\Practical 10>
```

03)

22001212_q3.scala > {} Bank

```scala
class Account(val accountId: String, private var balance: Double){

  def deposit(amount: Double):Unit={
    if(amount > 0){
      balance += amount;
      println(f"Deposited $amount%.2f to account $accountId");
    }
    else{
      println("Deposit amount must be positive");
    }
  }

  def withdraw(amount: Double):Unit={
    if(amount > 0 && amount <= balance){
      balance -= amount;
      println(f"Withdrew $amount%.2f from account $accountId");
    }
    else if(amount > balance){
      println("Insufficient funds");
    }
    else{
      println("Withdraw amount must be positive");
    }
  }

```

```
  def transfer(toAccount: Account, amount: Double):Unit={
    if(amount > 0 && amount <= balance){
      this.withdraw(amount);
      toAccount.deposit(amount);
      println(f"Transferred $amount%.2f from account $accountId to account ${toAccount.accountId}");
    }
    else{
      println("Transfer failed: Check amount and balance");
    }
  }

  def getBalance: Double = balance;

  override def toString: String = f"Account($accountId, Balance: $balance%.2f)";
}

object Bank extends App{
  val acc1 = new Account("Acc1", 2000.00);
  val acc2 = new Account("Acc2", 4000.00);

  println(acc1);
  println(acc2);

  acc1.deposit(1000.00);
  acc1.withdraw(200.00);
  acc1.transfer(acc2, 300.00);

  println(acc1);
  println(acc2);
}
```

```
Account(Acc1, Balance: 2000.00)
Account(Acc2, Balance: 4000.00)
Deposited 1000.00 to account Acc1
Withdrew 200.00 from account Acc1
Withdrew 300.00 from account Acc1
Deposited 300.00 to account Acc2
Transferred 300.00 from account Acc1 to account Acc2
Account(Acc1, Balance: 2500.00)
Account(Acc2, Balance: 4300.00)
PS C:\Users\User\Desktop\UCSC\2nd Year Sem-01\SCS 2204 -
```

04)

22001212_q4.scala > {} Bank1 > main

```scala
class Account1(val accountId:String, private var balance:Double) {

  def applyInterest():Unit={
    if(balance > 0){
      balance += balance * 0.05;
    }
    else{
      balance += balance * 0.1;
    }
  }

  def getBalance: Double = balance;

  override def toString: String = f"Account($accountId, Balance: $balance%.2f)";
}

object Bank1 {

  def accountsWithNegativeBalance(accounts:List[Account1]):List[Account1]={
    accounts.filter(_.getBalance < 0);
  }

  def totalBalance(accounts:List[Account1]):Double={
    accounts.map(_.getBalance).sum;
  }
```

```
26
27    def applyInterestToAll(accounts:List[Account1]):Unit={
28      accounts.foreach(_.applyInterest());
29    }
30
31    def main(args: Array[String]):Unit={
32
33      val acc1 = new Account1("A123", 1000.00);
34      val acc2 = new Account1("B456", -2000.00);
35      val acc3 = new Account1("C789", 3000.00);
36      val acc4 = new Account1("D012", -1000.00);
37
38      val accounts = List(acc1, acc2, acc3, acc4);
39
40      println("\nBalances before applying interest:");
41      accounts.foreach(println);
42
43      println("\nAccounts with negative balances:");
44      accountsWithNegativeBalance(accounts).foreach(println);
45
46      println(f"\nTotal balance of all accounts: ${totalBalance(accounts)}%.2f");
47
48      applyInterestToAll(accounts);
49      println("\nFinal balances after applying interest:");
50      accounts.foreach(println);
51    }
52  }
```

```
Balances before applying interest:
Account(A123, Balance: 1000.00)
Account(B456, Balance: -2000.00)
Account(C789, Balance: 3000.00)
Account(D012, Balance: -1000.00)

Accounts with negative balances:
Account(B456, Balance: -2000.00)
Account(D012, Balance: -1000.00)

Total balance of all accounts: 1000.00

Final balances after applying interest:
Account(A123, Balance: 1050.00)
Account(B456, Balance: -2200.00)
Account(C789, Balance: 3150.00)
Account(D012, Balance: -1100.00)
PS C:\Users\User\Desktop\UCSC\2nd Year Sem-01\SCS 2204 -
```

05)

22001212_q5.scala > {} LetterCount

```scala
object LetterCount{
  def countLetterOccurrences(words:List[String]):Int={
    val wordLengths = words.map(_.length);

    val totalLetters = wordLengths.reduce(_ + _);

    return totalLetters;
  }

  def main(args:Array[String]):Unit={
    val words = List("apple", "banana", "cherry", "date");
    println("Word list : ");
    words.foreach(println);
    val totalCount = countLetterOccurrences(words);
    println(s"\nTotal count of letter occurrences: $totalCount");
  }
}

```

```
PS C:\Users\User\Desktop\UCSC\2nd Year Sem-01\SCS 220
Word list :
apple
banana
cherry
date

Total count of letter occurrences: 21
PS C:\Users\User\Desktop\UCSC\2nd Year Sem-01\SCS 220
```