Practical 10

22000534

1)

Q1.scala > ...

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

  def neg: Rational = new Rational(-numerator, denominator)

  override def toString: String = s"$numerator/$denominator"
}

run | debug
object RationalMain extends App {
  val x = new Rational(3, 4)
  println(x.neg)
}
```

```
[Running] scala
"c:\Users\ravin_1g5z9nx\Documents\UCSC\2ndyear-Semester1\2204\Practical10\Q1.scala"
-3/4

[Done] exited with code=0 in 4.533 seconds
```

2)

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

  def neg: Fraction = new Fraction(-numerator, denominator)

  def subtract(that: Fraction): Fraction = {
    val newNumerator = this.numerator * that.denominator - that.numerator * this.denominator
    val newDenominator = this.denominator * that.denominator
    new Fraction(newNumerator, newDenominator)
  }

  override def toString: String = s"$numerator/$denominator"
}

run | debug
object FractionSubtractionMain extends App {
  val a = new Fraction(3, 4)
  val b = new Fraction(5, 8)
  val c = new Fraction(2, 7)
  val result = a.subtract(b.subtract(c))
  println(result)
}
```

```
[Running] scala "c:\Users\ravin_1g5z9nx\Documents\UCSC\2ndyear-Semester1\2204\Practical10\Q2
92/224

[Done] exited with code=0 in 4.785 seconds
```

3)

```scala
class Account(private var balance: Double) {
  def deposit(amount: Double): Unit = {
    require(amount > 0, "Deposit amount must be positive")
    balance += amount
  }

  def withdraw(amount: Double): Unit = {
    require(amount > 0, "Withdrawal amount must be positive")
    require(balance >= amount, "Insufficient balance")
    balance -= amount
  }

  def transfer(amount: Double, to: Account): Unit = {
    withdraw(amount)
    to.deposit(amount)
  }

  def getBalance: Double = balance
}

object AccountMain extends App {
  val acc1 = new Account(1000)
  val acc2 = new Account(500)
  acc1.transfer(200, acc2)
  println(s"Account 1 balance: ${acc1.getBalance}")
  println(s"Account 2 balance: ${acc2.getBalance}")
}
```

```
[Running] scala "c:\Users\ravin_1g5z9nx\Documents\UCSC\2ndyear-Semester1\2204\Practical10\Q3.scala"
Account 1 balance: 800.0
Account 2 balance: 700.0

[Done] exited with code=0 in 4.885 seconds
```

4)

```scala
class UserAccount(private var balance: Double) {
  def deposit(amount: Double): Unit = {
    require(amount > 0, "Deposit amount must be positive")
    balance += amount
  }

  def withdraw(amount: Double): Unit = {
    require(amount > 0, "Withdrawal amount must be positive")
    require(balance >= amount, "Insufficient balance")
    balance -= amount
  }

  def transfer(amount: Double, to: UserAccount): Unit = {
    withdraw(amount)
    to.deposit(amount)
  }

  def getBalance: Double = balance

  override def toString: String = s"Account(balance: $balance)"
}

object FinancialInstitution {
  def accountsWithNegativeBalances(accounts: List[(String, UserAccount)]): List[(String, Double)] = {
    accounts.filter(_._2.getBalance < 0).map { case (name, acc) => (name, acc.getBalance) }
  }

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

  def adjustedBalances(accounts: List[(String, UserAccount)]): List[(String, Double)] = {
    accounts.map { case (name, acc) =>
      val balance = acc.getBalance
      val adjustedBalance = if (balance > 0) {
        balance * 1.05
      } else {
        balance * 0.90
      }
      (name, adjustedBalance)
    }
  }
}
```

```scala
run | debug
object FinancialInstitutionMain extends App {
  val userAcc1 = new UserAccount(1000)
  val userAcc2 = new UserAccount(-200)
  val userAcc3 = new UserAccount(300)

  val allAccounts = List(
    ("Account1", userAcc1),
    ("Account2", userAcc2),
    ("Account3", userAcc3)
  )

  println("Accounts with negative balances:")
  FinancialInstitution.accountsWithNegativeBalances(allAccounts).foreach { case (name, balance) =>
    println(s"$name: $balance")
  }

  println(s"\nTotal balance of all accounts: ${FinancialInstitution.totalBalance(allAccounts.map(_._2))}")

  println("\nAdjusted balances after interest:")
  FinancialInstitution.adjustedBalances(allAccounts).foreach { case (name, adjustedBalance) =>
    println(s"$name: $adjustedBalance")
  }
}
```

```
[Running] scala "c:\Users\ravin_1g5z9nx\Documents\UCSC\2ndyear-Semester1\2204\Practical10\Q4.scala"
Accounts with negative balances:
Account2: -200.0

Total balance of all accounts: 1100.0

Adjusted balances after interest:
Account1: 1050.0
Account2: -180.0
Account3: 315.0
```

5)

```scala
run | debug
object LetterCounter {
  def countLetterOccurrences(words: List[String]): Int = {
    val lengths = words.map(_.length)

    val totalLetters = lengths.reduce(_ + _)

    totalLetters
  }

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

```
[Running] scala "c:\Users\ravin_1g5z9nx\Documents\UCSC\2ndyear-Semester1\2204\Practical1
Total count of letter occurrences: 21
```