

Practical 10 - 22001956

Q1

The screenshot shows the VS Code editor with a file named `q1.scala` open. The code defines a `Rational` class with a constructor that takes numerator `n` and denominator `d`. It includes a `require` statement to ensure the denominator is not zero. The class has private fields for the greatest common divisor `g`, and public fields for `numerator` and `denominator`. It also has a `neg` method to return the negative of the rational number, a `toString` method, and a private `gcd` method. An object `q1` extends `App` and creates a `Rational` instance `x` with numerator 3 and denominator 4, then prints its value and its negative.

```
1 class Rational(n: Int, d: Int) {
2   require(d != 0, "Denominator cannot be zero")
3
4   private val g = gcd(n.abs, d.abs)
5   val numerator: Int = n / g
6   val denominator: Int = d / g
7
8   def this(n: Int) = this(n, 1)
9
10  def neg: Rational = new Rational(-numerator, denominator)
11
12  override def toString: String = s"$numerator/$denominator"
13
14  private def gcd(a: Int, b: Int): Int = if (b == 0) a else gcd(b, a % b)
15 }
16
17 object q1 extends App {
18   val x = new Rational(3, 4)
19   println(s"x: $x")
20   println(s"x.neg: ${x.neg}")
21 }
22
```

The terminal on the right shows the following commands and output:

```
PS D:\UCSC\2Yr 1Sem\FP\practical10> scalac q1.scala
PS D:\UCSC\2Yr 1Sem\FP\practical10> scala q1
x: 3/4
x.neg: -3/4
PS D:\UCSC\2Yr 1Sem\FP\practical10>
```

Q2

The screenshot shows the VS Code editor with a file named `q2.scala` open. The code defines a `Rational` class with a constructor that takes numerator `n` and denominator `d`. It includes a `require` statement to ensure the denominator is not zero. The class has private fields for the greatest common divisor `g`, and public fields for `numerator` and `denominator`. It also has a `neg` method to return the negative of the rational number, a `toString` method, and a private `gcd` method. An object `q2` extends `App` and creates three `Rational` instances `x`, `y`, and `z` with values (3, 4), (5, 8), and (2, 7) respectively. It then calculates the result of `x - (y - z)` and prints it.

```
1 class Rational(n: Int, d: Int) {
2   require(d != 0, "Denominator cannot be zero")
3
4   private val g = gcd(n.abs, d.abs)
5   val numerator: Int = n / g
6   val denominator: Int = d / g
7
8   def this(n: Int) = this(n, 1)
9
10  def sub(that: Rational): Rational = {
11    new Rational(
12      this.numerator * that.denominator - that.numerator * this.denominator,
13      this.denominator * that.denominator
14    )
15  }
16
17  override def toString: String = s"$numerator/$denominator"
18
19  private def gcd(a: Int, b: Int): Int = if (b == 0) a else gcd(b, a % b)
20 }
21
22 object q2 extends App {
23   val x = new Rational(3, 4)
24   val y = new Rational(5, 8)
25   val z = new Rational(2, 7)
26
27   val result = x.sub(y.sub(z))
28
29   println(s"x - (y - z) = $result")
30 }

```

The terminal on the right shows the following commands and output:

```
PS D:\UCSC\2Yr 1Sem\FP\practical10> scalac q2.scala
PS D:\UCSC\2Yr 1Sem\FP\practical10> scala q2
x - (y - z) = 23/56
PS D:\UCSC\2Yr 1Sem\FP\practical10>
```

Q3

```

1 class Account(private var balance: Double) {
2   def deposit(amount: Double): Unit = {
3     if (amount > 0) {
4       balance += amount
5       println(f"Deposited $amount%.2f.\nNew balance: $balance%.2f\n")
6     } else {
7       println("Deposit amount must be positive.\n")
8     }
9   }
10
11   def withdraw(amount: Double): Boolean = {
12     if (amount <= balance) {
13       balance -= amount
14       println(f"Withdraw $amount%.2f.\nNew balance: $balance%.2f\n")
15       true
16     } else {
17       println("Insufficient funds or invalid amount.\n")
18       false
19     }
20   }
21
22   def transfer(amount: Double, toAccount: Account): Boolean = {
23     if (withdraw(amount)) {
24       toAccount.deposit(amount)
25       println(f"Transferred $amount%.2f to the target account.")
26       true
27     } else {
28       println("\nTransfer failed.")
29       false
30     }
31   }
32
33   def getBalance: Double = balance
34
35   override def toString: String = f"Account balance: $balance%.2f"
36 }
37
38 object q3 extends App {
39   val account1 = new Account(1500.00)
40   val account2 = new Account(200.5098237527)
41   account1.transfer(500, account2)
42 }
43
44

```

```

PS D:\UCSC\2Yr 1Sem\FPI\practical10> scalac q3.scala
PS D:\UCSC\2Yr 1Sem\FPI\practical10> scala q3
Withdraw 500.00.
New balance: 1000.00

Deposited 500.00.
New balance: 700.51

Transferred 500.00 to the target account.
PS D:\UCSC\2Yr 1Sem\FPI\practical10>

```

Q4

```

1 class Account(private var balance: Double) {
2   def deposit(amount: Double): Unit = {
3     if (amount > 0) {
4       balance += amount
5       println(f"Deposited $amount%.2f.\nNew balance: $balance%.2f\n")
6     } else {
7       println("Deposit amount must be positive.\n")
8     }
9   }
10
11   def withdraw(amount: Double): Boolean = {
12     if (amount <= balance) {
13       balance -= amount
14       println(f"Withdraw $amount%.2f.\nNew balance: $balance%.2f\n")
15       true
16     } else {
17       println("Insufficient funds or invalid amount.\n")
18       false
19     }
20   }
21
22   def transfer(amount: Double, toAccount: Account): Boolean = {
23     if (withdraw(amount)) {
24       toAccount.deposit(amount)
25       println(f"Transferred $amount%.2f to the target account.")
26       true
27     } else {
28       println("\nTransfer failed.")
29       false
30     }
31   }
32
33   def getBalance: Double = balance
34
35   def applyInterest(): Unit = {
36     if (balance > 0) {
37       balance += balance * 0.05
38     } else {
39       balance += balance * 0.10
40     }
41   }
42
43   override def toString: String = f"Account balance: $balance%.2f"
44 }
45

```

```

PS D:\UCSC\2Yr 1Sem\FPI\practical10> scalac q3.scala
PS D:\UCSC\2Yr 1Sem\FPI\practical10> scala q3
Withdraw 500.00.
New balance: 1000.00

Deposited 500.00.
New balance: 700.51

Transferred 500.00 to the target account.
PS D:\UCSC\2Yr 1Sem\FPI\practical10> scalac q4.scala
PS D:\UCSC\2Yr 1Sem\FPI\practical10> scala q4
Accounts with negative balances:
Account balance: -200.51
Account balance: -1000.20

Total balance of all accounts: 309.79

Balances after applying interest:
Account balance: 1575.00
Account balance: -220.56
Account balance: 11.03
Account balance: -1100.22
PS D:\UCSC\2Yr 1Sem\FPI\practical10>

```

Q4(continued)

```
File Edit Selection View ... practical10
q1.scala q2.scala q3.scala q4.scala x q5.scala
q4.scala > ...
1 class Accountt(private var balance: Double) {
2   }
3   override def toString: String = f"Account balance: $balanceX.2f"
4   }
5   }
6   class Bank(private var accounts: List[Accountt]) {
7     def accountsWithNegativeBalances: List[Accountt] = {
8       accounts.filter(_.getBalance < 0)
9     }
10    }
11    def totalBalance: Double = {
12      accounts.map(_.getBalance).sum
13    }
14    }
15    def applyInterestToAllAccounts(): Unit = {
16      accounts.foreach(_.applyInterest())
17    }
18    }
19    override def toString: String = {
20      accounts.map(_.toString).mkString("\n")
21    }
22    }
23  }
24  run | debug
25  object q4 extends App {
26    val account1 = new Accountt(1500.00)
27    val account2 = new Accountt(-200.5098237527)
28    val account3 = new Accountt(10.50)
29    val account4 = new Accountt(-1000.20)
30    }
31    val bank = new Bank(List(account1, account2, account3, account4))
32    }
33    println("Accounts with negative balances:")
34    bank.accountsWithNegativeBalances.foreach(println)
35    }
36    println(f"\nTotal balance of all accounts: ${bank.totalBalance}X.2f")
37    }
38    bank.applyInterestToAllAccounts()
39    println("\nBalances after applying interest:")
40    println(bank)
41    }
42  }
43  }
44  }
45  }
46  }
47  }
48  }
49  }
50  }
51  }
52  }
53  }
54  }
55  }
56  }
57  }
58  }
59  }
60  }
61  }
62  }
63  }
64  }
65  }
66  }
67  }
68  }
69  }
70  }
71  }
72  }
73  }
74  }
75  }
76  }
77  }
78  }
79  }
80  }
81  }
82  }
83  }
84  }
85  }
86  }
87  }
88  }
89  }
90  }
91  }
92  }
93  }
94  }
95  }
96  }
97  }
98  }
99  }
100 }
```

```
PS D:\UCSC\2Yr 1Sem\FP\practical10> scalac q3.scala
PS D:\UCSC\2Yr 1Sem\FP\practical10> scala q3
Withdrawn 500.00.
New balance: 1000.00
Deposited 500.00.
New balance: 700.51
Transferred 500.00 to the target account.
PS D:\UCSC\2Yr 1Sem\FP\practical10> scalac q4.scala
PS D:\UCSC\2Yr 1Sem\FP\practical10> scala q4
Accounts with negative balances:
Account balance: -200.51
Account balance: -1000.20
Total balance of all accounts: 309.79
Balances after applying interest:
Account balance: 1575.00
Account balance: -220.56
Account balance: 11.03
Account balance: -1100.22
PS D:\UCSC\2Yr 1Sem\FP\practical10>
```

Q5

```
File Edit Selection View Go Run ... practical10
q1.scala q2.scala q3.scala q4.scala q5.scala x
q5.scala > {} q5
run | debug
1 object q5 extends App {
2   }
3   def countLetterOccurrences(words: List[String]): Int = {
4     val wordLengths = words.map(_.length)
5     }
6     val totalLetters = wordLengths.reduce(_ + _)
7     }
8     totalLetters
9   }
10  }
11  val words = List("apple", "banana", "cherry", "date")
12  val totalCount = countLetterOccurrences(words)
13  }
14  println(s"Total count of letter occurrences: $totalCount")
15  }
16  }
```

```
PS D:\UCSC\2Yr 1Sem\FP\practical10> scalac q5.scala
PS D:\UCSC\2Yr 1Sem\FP\practical10> scala q5
Total count of letter occurrences: 21
PS D:\UCSC\2Yr 1Sem\FP\practical10>
```