Muhammad Shas K T
UID: 281775

Analysis of Airline Delay using Spark

Problem Statement 1:

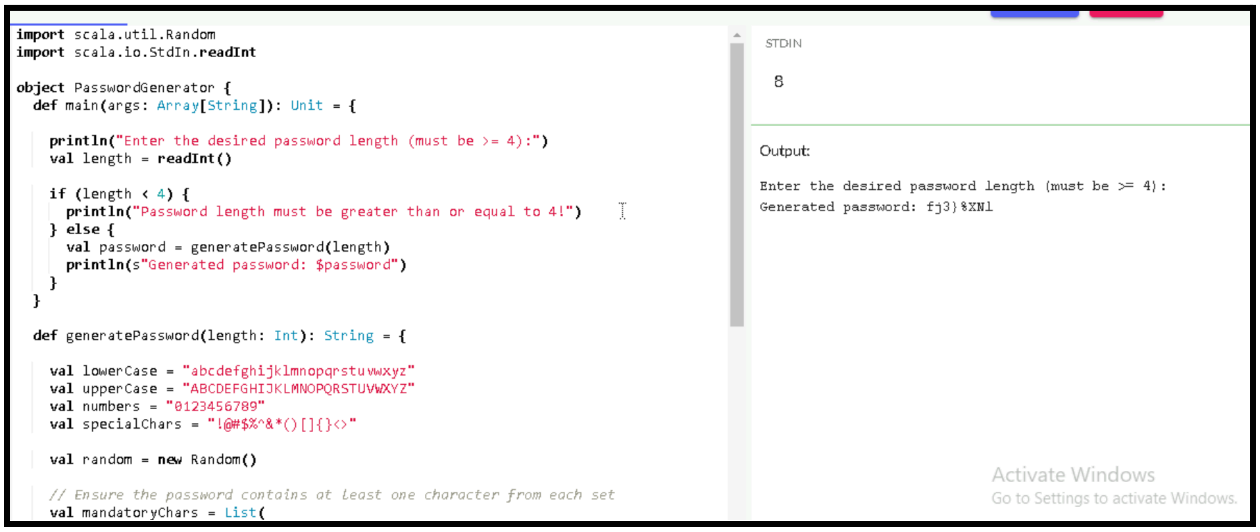
Generate solution for you are tasked with creating a random password generator in Scala. The generator will take user input for password length and generate a random password that includes a mix of lowercase letters, uppercase letters, numbers, and special characters.

Code:-

```
import scala.util.Random
import scala.io.StdIn.readInt

object PasswordGenerator {
 def main(args: Array[String]): Unit = {

   println("Enter the desired password length (must be >= 4):")
   val length = readInt()

   if (length < 4) {
     println("Password length must be greater than or equal to 4!")
   } else {
     val password = generatePassword(length)
     println(s"Generated password: $password")
   }
 }

 def generatePassword(length: Int): String = {

   val lowerCase = "abcdefghijklmnopqrstuvwxyz"
   val upperCase = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
   val numbers = "0123456789"
   val specialChars = "!@#$%^&*()[]{}<>"

   val random = new Random()

   // Ensure the password contains at least one character from each set
   val mandatoryChars = List(
     lowerCase(random.nextInt(lowerCase.length)),
     upperCase(random.nextInt(upperCase.length)),
     numbers(random.nextInt(numbers.length)),
     specialChars(random.nextInt(specialChars.length))
   )

   val allChars = lowerCase + upperCase + numbers + specialChars
   val remainingChars = (1 to (length - mandatoryChars.length)).map { _ =>
     allChars(random.nextInt(allChars.length))
   }
```

```
  // Combine mandatory and remaining characters
  val passwordChars = (mandatoryChars ++ remainingChars).toArray

  // Shuffle the characters
  val password = random.shuffle(passwordChars.toList).mkString

  password
 }
}
```

Output:-



```
import scala.util.Random
import scala.io.StdIn.readInt

object PasswordGenerator {
  def main(args: Array[String]): Unit = {

    println("Enter the desired password length (must be >= 4):")
    val length = readInt()

    if (length < 4) {
      println("Password length must be greater than or equal to 4!")
    } else {
      val password = generatePassword(length)
      println(s"Generated password: $password")
    }
  }

  def generatePassword(length: Int): String = {

    val lowerCase = "abcdefghijklmnopqrstuvwxyz"
    val upperCase = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    val numbers = "0123456789"
    val specialChars = "!@#$%^&*()[]{}<>"

    val random = new Random()

    // Ensure the password contains at least one character from each set
    val mandatoryChars = List(
```

STDIN

```
8
```

Output:

```
Enter the desired password length (must be >= 4):
Generated password: fj3}%XNl
```

Activate Windows
Go to Settings to activate Windows.

Problem Statement 2: UST Shopping Cart Application

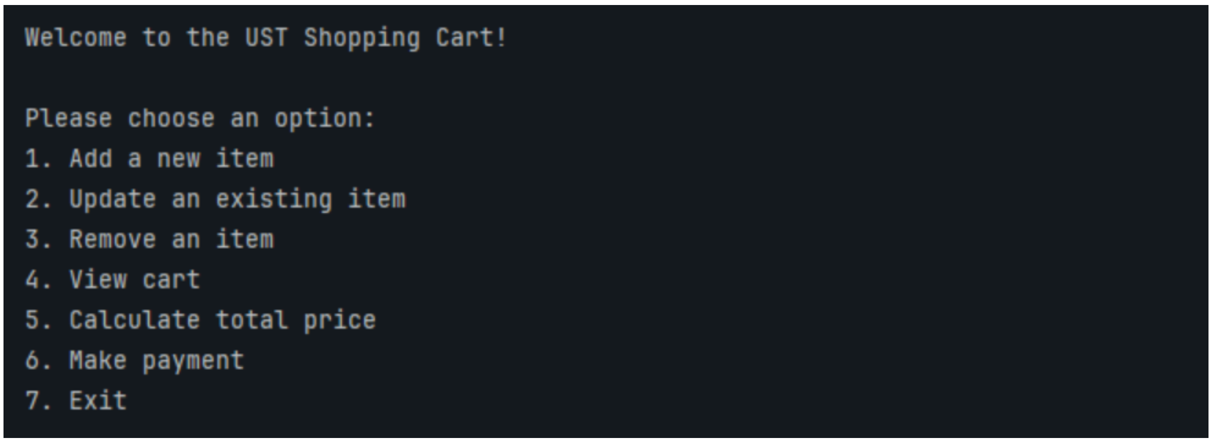
You are tasked with developing a Shopping Cart application in Scala. The application will manage a shopping cart, allowing customers to add, remove, update, view items in their cart, and proceed to payment. Each item will have details such as name, quantity, price, and category. Additionally, users will be able to make payments through a simulated payment gateway – Credit Car, Debit Card, UPI. The application will also calculate the total price including GST (Goods and Services Tax) and will add delivery charges below than Rs.200 cart value.

Code:-

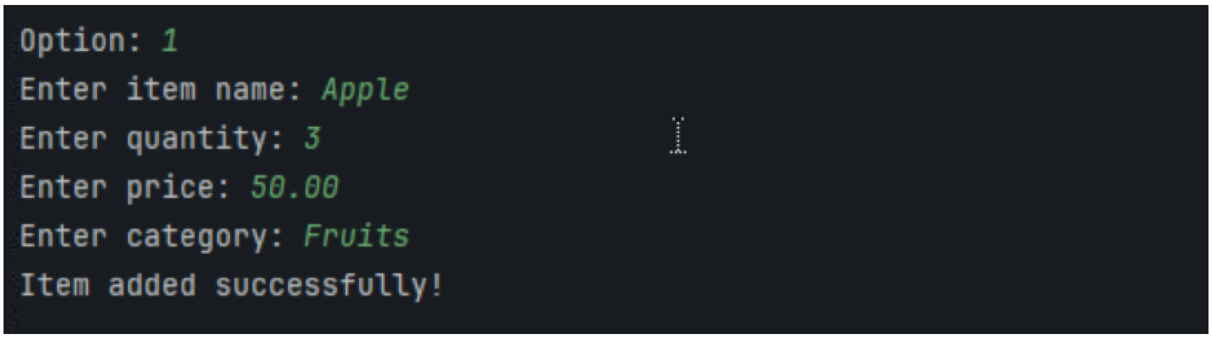
```
case class Item(id: Int, name: String, quantity: Int, price: Double, category: String)

class ShoppingCart {
 private var items: Map[Int, Item] = Map()
 private var currentId: Int = 0

 def addItem(item: Item): Unit = {
  currentId += 1
  items += (currentId -> item.copy(id = currentId))
  println("Item added successfully!")
 }

 def updateItem(id: Int, updatedItem: Item): Unit = {
  if (items.contains(id)) {
    items += (id -> updatedItem.copy(id = id))
    println("Item updated successfully!")
  } else {
    println("Item not found.")
  }
 }

 def removeItem(id: Int): Unit = {
  if (items.contains(id)) {
    items -= id
    println("Item removed successfully!")
  } else {
    println("Item not found.")
  }
 }

 def viewCart(): Unit = {
  if (items.isEmpty) {
    println("Your cart is empty.")
  } else {
    println("Viewing cart:")
    items.values.foreach(item => println(s"${item.id}. ${item.name} - Quantity: ${item.quantity}, Price: 
${item.price}, Category: ${item.category}"))
  }
 }

 def totalPrice(withGST: Boolean = true): Double = {
  val total = items.values.map(item => item.quantity * item.price).sum
  val gst = if (withGST) total * 0.05 else 0.0
  val deliveryCharge = if (total < 200) 30 else 0

  println(f"Cart Value: \t\t₹${total}%.2f")
  println(f"Delivery Charge: \t₹${deliveryCharge}%.2f")
  println(f"GST: \t\t\t₹${gst}%.2f")
  val amountPayable = total + gst + deliveryCharge
  println(f"Amount Payable: \t₹${amountPayable}%.2f")

  amountPayable
 }
}


class PaymentGateway {
 def processPayment(amount: Double, paymentMethod: String): String = {
  val confirmationNumber = "UST" + (100000000 + scala.util.Random.nextInt(900000000))
  println(s"Processing payment of ₹$amount using $paymentMethod...")
  println(s"Payment successful! Confirmation number: $confirmationNumber")
  confirmationNumber
 }
}

object ShoppingCartApp extends App {
```

```scala
val cart = new ShoppingCart
val paymentGateway = new PaymentGateway

def showMenu(): Unit = {
  println(
    """
      Welcome to the UST Shopping Cart!

      Please choose an option:
       1. Add a new item
       2. Update an existing item
       3. Remove an item
       4. View cart
       5. Calculate total price
       6. Make payment
       7. Exit
    """.stripMargin)
}

var continue = true
while (continue) {
  showMenu()
  print("Option: ")
  val option = scala.io.StdIn.readInt()
  option match {
    case 1 =>
      print("Enter item name: ")
      val name = scala.io.StdIn.readLine()
      print("Enter quantity: ")
      val quantity = scala.io.StdIn.readInt()
      print("Enter price: ")
      val price = scala.io.StdIn.readDouble()
      print("Enter category: ")
      val category = scala.io.StdIn.readLine()
      cart.addItem(Item(0, name, quantity, price, category))

    case 2 =>
      print("Enter item ID to update: ")
      val id = scala.io.StdIn.readInt()
      print("Enter new item name: ")
      val name = scala.io.StdIn.readLine()
      print("Enter new quantity: ")
      val quantity = scala.io.StdIn.readInt()
      print("Enter new price: ")
      val price = scala.io.StdIn.readDouble()
      print("Enter new category: ")
      val category = scala.io.StdIn.readLine()
      cart.updateItem(id, Item(id, name, quantity, price, category))

    case 3 =>
      print("Enter item ID to remove: ")
      val id = scala.io.StdIn.readInt()
      cart.removeItem(id)
    case 4 =>
      cart.viewCart()

    case 5 =>
      cart.totalPrice()

    case 6 =>
      val totalAmount = cart.totalPrice()
      print("Choose a payment method (Credit Card/Debit Card/UPI): ")
      val paymentMethod = scala.io.StdIn.readLine()
      paymentGateway.processPayment(totalAmount, paymentMethod)

    case 7 =>
      continue = false
      println("Exiting the application. Goodbye!")

    case _ =>
      println("Invalid option. Please try again.")
   }
  }
}
```

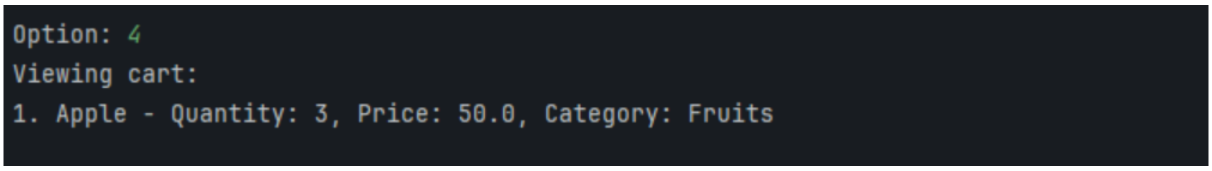
Output:-



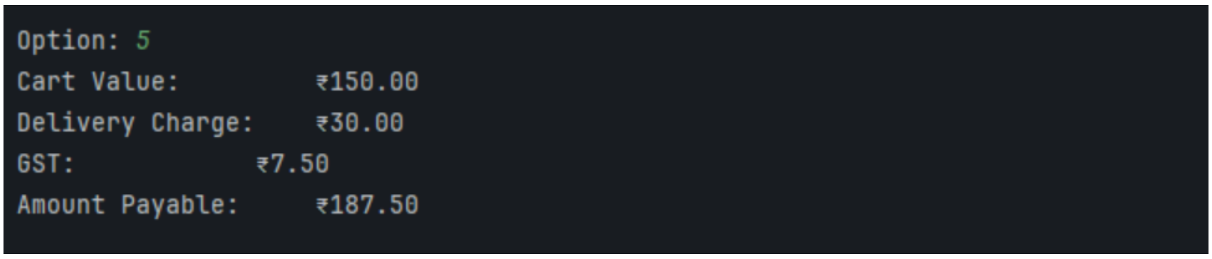
```
Welcome to the UST Shopping Cart!

Please choose an option:
1. Add a new item
2. Update an existing item
3. Remove an item
4. View cart
5. Calculate total price
6. Make payment
7. Exit
```



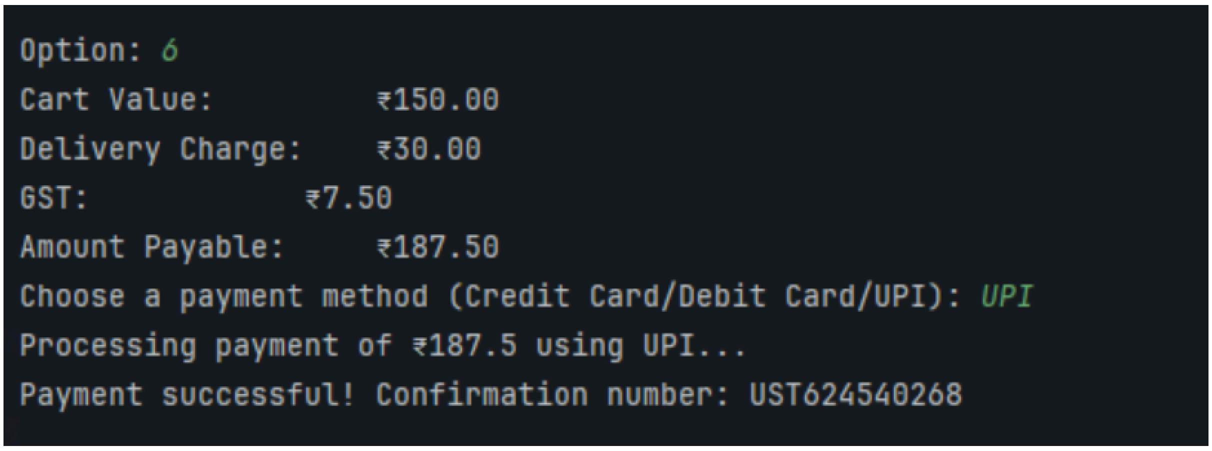
```
Option: 1
Enter item name: Apple
Enter quantity: 3
Enter price: 50.00
Enter category: Fruits
Item added successfully!
```



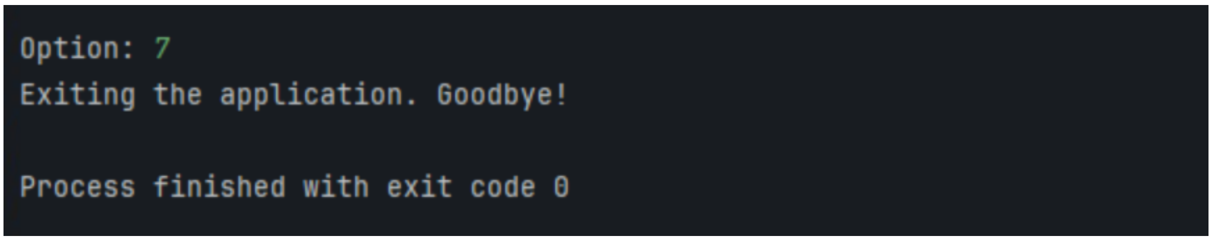
```
Option: 4
Viewing cart:
1. Apple - Quantity: 3, Price: 50.0, Category: Fruits
```



```
Option: 5
Cart Value:         ₹150.00
Delivery Charge:    ₹30.00
GST:            ₹7.50
Amount Payable:     ₹187.50
```



```
Option: 6
Cart Value:         ₹150.00
Delivery Charge:    ₹30.00
GST:            ₹7.50
Amount Payable:     ₹187.50
Choose a payment method (Credit Card/Debit Card/UPI): UPI
Processing payment of ₹187.5 using UPI...
Payment successful! Confirmation number: UST624540268
```



```
Option: 7
Exiting the application. Goodbye!

Process finished with exit code 0
```

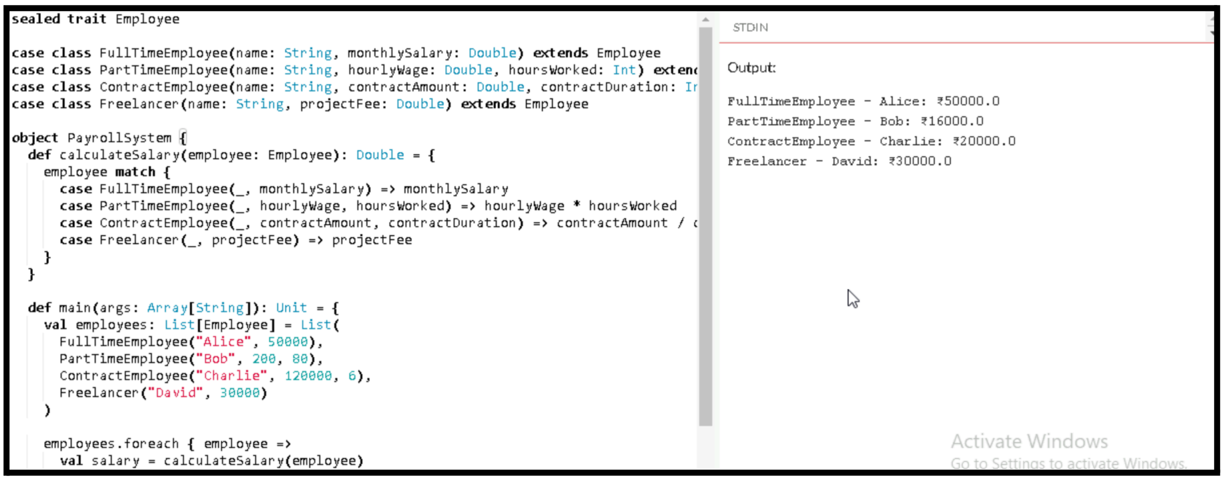
<u>Problem Statement 3: Case Classes and Pattern Matching</u>

Create a Scala application that uses case classes to model a simple payroll system. Implement pattern matching to calculate the salary of different types of employee – FullTimeEmployee, PartTimeEmployee, ContractType, Freelancers.

<u>Code:-</u>

```
sealed trait Employee

case class FullTimeEmployee(name: String, monthlySalary: Double) extends Employee
case class PartTimeEmployee(name: String, hourlyWage: Double, hoursWorked: Int) extends
Employee
case class ContractEmployee(name: String, contractAmount: Double, contractDuration: Int) extends
Employee // contractDuration in months
case class Freelancer(name: String, projectFee: Double) extends Employee

object PayrollSystem {
  def calculateSalary(employee: Employee): Double = {
    employee match {
      case FullTimeEmployee(_, monthlySalary) => monthlySalary
      case PartTimeEmployee(_, hourlyWage, hoursWorked) => hourlyWage * hoursWorked
      case ContractEmployee(_, contractAmount, contractDuration) => contractAmount /
contractDuration
      case Freelancer(_, projectFee) => projectFee
    }
  }

  def main(args: Array[String]): Unit = {
    val employees: List[Employee] = List(
      FullTimeEmployee("Alice", 50000),
      PartTimeEmployee("Bob", 200, 80),
      ContractEmployee("Charlie", 120000, 6),
      Freelancer("David", 30000)
    )

    employees.foreach { employee =>
      val salary = calculateSalary(employee)
      println(s"${employee.getClass.getSimpleName} - ${employee.asInstanceOf[{ def name: String
}].name}: ₹$salary")
    }
  }
}
```

Output:-



```
sealed trait Employee

case class FullTimeEmployee(name: String, monthlySalary: Double) extends Employee
case class PartTimeEmployee(name: String, hourlyWage: Double, hoursWorked: Int) extenc
case class ContractEmployee(name: String, contractAmount: Double, contractDuration: Ir
case class Freelancer(name: String, projectFee: Double) extends Employee

object PayrollSystem {
  def calculateSalary(employee: Employee): Double = {
    employee match {
      case FullTimeEmployee(_, monthlySalary) => monthlySalary
      case PartTimeEmployee(_, hourlyWage, hoursWorked) => hourlyWage * hoursWorked
      case ContractEmployee(_, contractAmount, contractDuration) => contractAmount / c
      case Freelancer(_, projectFee) => projectFee
    }
  }

  def main(args: Array[String]): Unit = {
    val employees: List[Employee] = List(
      FullTimeEmployee("Alice", 50000),
      PartTimeEmployee("Bob", 200, 80),
      ContractEmployee("Charlie", 120000, 6),
      Freelancer("David", 30000)
    )

    employees.foreach { employee =>
      val salary = calculateSalary(employee)
```

```
Output:

FullTimeEmployee - Alice: ₹50000.0
PartTimeEmployee - Bob: ₹16000.0
ContractEmployee - Charlie: ₹20000.0
Freelancer - David: ₹30000.0
```

Problem Statement 4: File Processing

Write a Scala program to read a text file, count the occurrences of each word, and display the top N most frequent words.

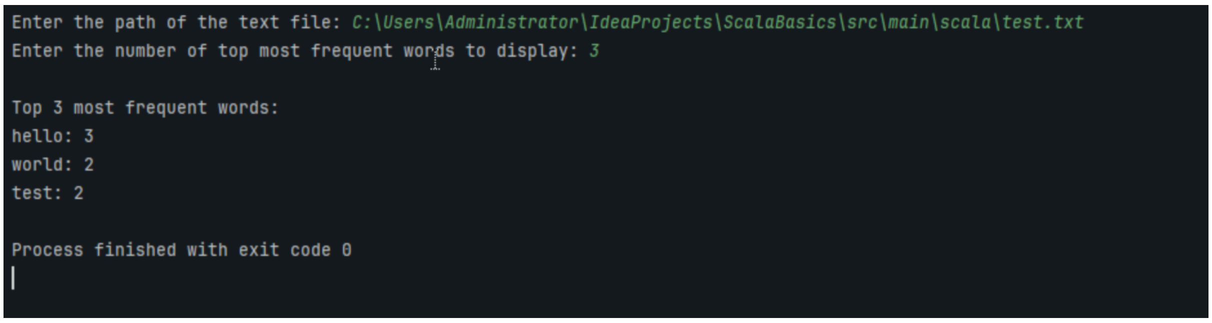
- Create a method wordCount(filePath: String, topN: Int): List[(String, Int)] that reads a text file and returns a list of tuples containing the top N most frequent words and their counts.
- Program ask user to enter N top most frequent words and show N most frequent words as output.

Code:-

```
import scala.io.Source
import scala.collection.mutable
```

```scala
object FileProcessor {
 def main(args: Array[String]): Unit = {

   print("Enter the path of the text file: ")
   val filePath = scala.io.StdIn.readLine()

   print("Enter the number of top most frequent words to display: ")
   val topN = scala.io.StdIn.readInt()

   val topWords = wordCount(filePath, topN)

   println(s"\nTop $topN most frequent words:")
   topWords.foreach { case (word, count) =>
     println(s"$word: $count")
   }
 }

 def wordCount(filePath: String, topN: Int): List[(String, Int)] = {

   val source = Source.fromFile(filePath)
   val lines = try source.getLines().mkString(" ") finally source.close()

   // Split the content into words, remove punctuation, and convert to lower case
   val words = lines.split("\\W+").filter(_.nonEmpty).map(_.toLowerCase)

   val wordCounts = mutable.Map[String, Int]()
   words.foreach { word =>
     wordCounts(word) = wordCounts.getOrElse(word, 0) + 1
   }

   // Sort the words by their counts in descending order and take the top N
   wordCounts.toList.sortBy { case (_, count) => -count }.take(topN)
 }
}
```

Output:-



```
Enter the path of the text file: C:\Users\Administrator\IdeaProjects\ScalaBasics\src\main\scala\test.txt
Enter the number of top most frequent words to display: 3

Top 3 most frequent words:
hello: 3
world: 2
test: 2

Process finished with exit code 0
```

Problem Statement 5: File Analysis Application in Scala

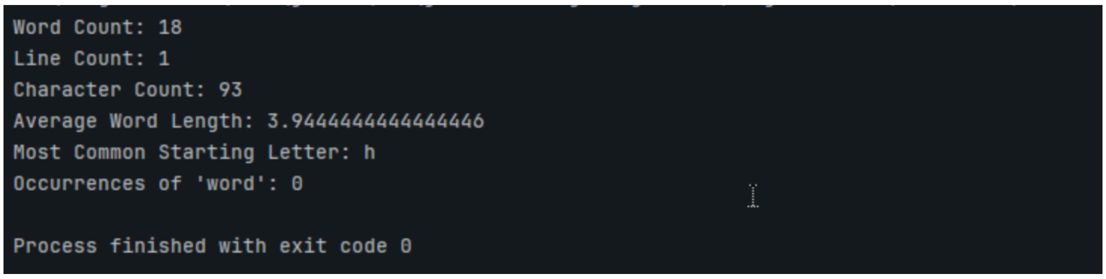
The application will process a text file and provide various analytical insights about its content. The insights will include word count, line count, character count, frequency of each word, and the top N most frequent words.

a. FileAnalyzer Class: Create a FileAnalyzer class with the following methods:
b. loadFile(filePath: String): Load and Read a text file.
c. wordCount(): Returns the total number of words in the file.
d. lineCount(): Returns the total number of lines in the file.
e. characterCount(): Returns the total number of characters in the file.
f. averageWordLength(): Double: Returns the average word length in the file.
g. mostCommonStartingLetter(): Option[Char]: Returns the most common starting alphabet of words in the input files.
h. wordOccurrences(word: String): Int: Returns the number of occurrences of a specific word in file.

Code:-

```
import scala.io.Source
import scala.collection.mutable

class FileAnalyzer {
  private var lines: Seq[String] = Seq()
  private var words: Seq[String] = Seq()

  def loadFile(filePath: String): Unit = {
    lines = Source.fromFile(filePath).getLines().toSeq
    words = lines.flatMap(_.split("\\W+")).filter(_.nonEmpty)
  }

  def wordCount(): Int = words.length
  def lineCount(): Int = lines.length
  def characterCount(): Int = lines.map(_.length).sum

  def averageWordLength(): Double = {
    if (words.isEmpty) 0.0
    else words.map(_.length).sum.toDouble / words.length
  }

  def mostCommonStartingLetter(): Option[Char] = {
    if (words.isEmpty) None
    else {
      val startingLetters = words.map(_.toLowerCase.head)
      val mostCommon = startingLetters.groupBy(identity).mapValues(_.size).maxBy(_._2)._1
      Some(mostCommon)
    }
  }

  def wordOccurrences(word: String): Int = {
    words.count(_.equalsIgnoreCase(word))
  }
}

object FileAnalyzerApp extends App {
  val analyzer = new FileAnalyzer
  analyzer.loadFile("C:\\Users\\Administrator\\IdeaProjects\\ScalaBasics\\src\\main\\scala\\test.txt")

  println(s"Word Count: ${analyzer.wordCount()}")
  println(s"Line Count: ${analyzer.lineCount()}")
  println(s"Character Count: ${analyzer.characterCount()}")
  println(s"Average Word Length: ${analyzer.averageWordLength()}")
  println(s"Most Common Starting Letter:
${analyzer.mostCommonStartingLetter().getOrElse("None")}")
  println(s"Occurrences of 'word': ${analyzer.wordOccurrences("word")}")
}
```

Output:-



```
Word Count: 18
Line Count: 1
Character Count: 93
Average Word Length: 3.9444444444444446
Most Common Starting Letter: h
Occurrences of 'word': 0

Process finished with exit code 0
```