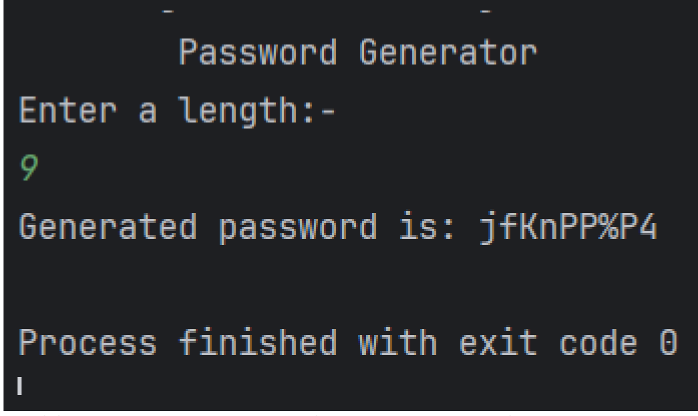
ASSIGNMENT - SCALA

1. **Problem Statement 1:**
   Generate solutions 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.

```
object password
{
 import scala.io.StdIn
 import scala.util.Random
 def main (args: Array[String]): Unit=
 {
   println("\t\tPassword Generator")
   println("Enter a length:- ")
   val ln =StdIn.readInt()
   val password = makePass(ln)
   println(s"Generated password is: $password")
 }
 def makePass(leng: Int): String = {
   val allCharacters =
"!@#$%^&*()|;:?`~ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789a
bcdefghijklmnopqrstuvwxyz"
   val random = new Random
   val password = new StringBuilder(leng)
   for (_ <- 1 to leng) {
     val randomChar =
allCharacters(random.nextInt(allCharacters.length))
     password.append(randomChar)
   }
   password.toString()
 }
}
```

OUTPUT



```
        Password Generator
Enter a length:-
9
Generated password is: jfKnPP%P4

Process finished with exit code 0
```

2. **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.

Item Class:

Create an **Item () class** with the following attributes:

- id: Unique identifier for the item
- name: Name of the item.
- quantity: Quantity of the item.
- price: Price of the item.
- category: Category of the item.

Shopping Cart:

Create a **ShoppingCart()** class that manages a collection of Item objects. Implement the following methods:

- addItem(item: Item): Adds a new item to the cart.
- updateItem(id: Int, updatedItem: Item): Updates an existing item in the cart.
- removeItem(id: Int): Removes an item from the cart.
- viewCart(): Displays all items in the cart.
- totalPrice(withGST: Boolean = true): Calculates and displays the total price of all items in the cart with GST charges.

Payment Gateway:

- Create a PaymentGateway class to simulate payment processing. Implement the following methods:
  processPayment(amount: Double, paymentMethod: String): Simulates processing a payment and returns a confirmation message.
- Payment methods can include "Credit Card", "Debit Card", and "UPI".

GST Calculation:

- Assume a GST rate of 5%. Implement the GST calculation within the totalPrice method.

Cart Operations:

Implement the following functions:

- Add Item: Allow users to input details for a new item and add it to the cart.
- Update Item: Allow users to update the details of an existing item.
- Remove Item: Allow users to remove an item from the cart.
- View Cart: Display all items in the cart.
- Calculate Total: Calculate and display the total price of items in the cart, optionally including GST.
- Make Payment: Allow users to proceed to payment and select a payment method. Use the PaymentGateway class to process the payment.

Error Handling:

Ensure appropriate error handling for scenarios such as trying to update or remove an item that doesn't exist, and handling payment errors.

Sample Outcome:

```
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 & Vegetables
 Item added successfully!

 Option: 4
 Viewing cart:
 1. [1] Apple - Quantity: 3, Price: 50.00, Category: Fruits & Vegetables
 ...

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

 Option: 6
 Choose a payment method (Credit Card/Debit Card/UPI): Credit Card
 Processing payment of 187.50 using Credit Card...
 Payment successful! Confirmation number: UST123456789

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

Code:

```scala
import scala.collection.mutable.Map


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


class PaymentGateway {
 def processPayment(amount: Double, paymentMethod:
String): String = {
   paymentMethod match {
     case "Credit Card" | "Debit Card" | "UPI" =>
       s"Payment of ₹${amount} processed successfully
using $paymentMethod."
     case _ =>
       s"Payment method $paymentMethod not supported."
   }
 }
}

class ShoppingCart() {
 var items = Map[String, Item]()
 val GST_RATE = 0.05
 val Delivery_Charge_Threshold = 200.0
 val Delivery_charge = 30.0

 def add(item: Item): Unit = {
   if (item.quantity <= 0 || item.price < 0) {
     println("Quantity must be greater than zero and price
cannot be negative.")
   } else if (items.contains(item.name)) {
     println("Item is already in your cart... Use update
to change Quantity")
   } else {
     items += (item.name -> item)
     println("Added Successfully")
   }
 }
```

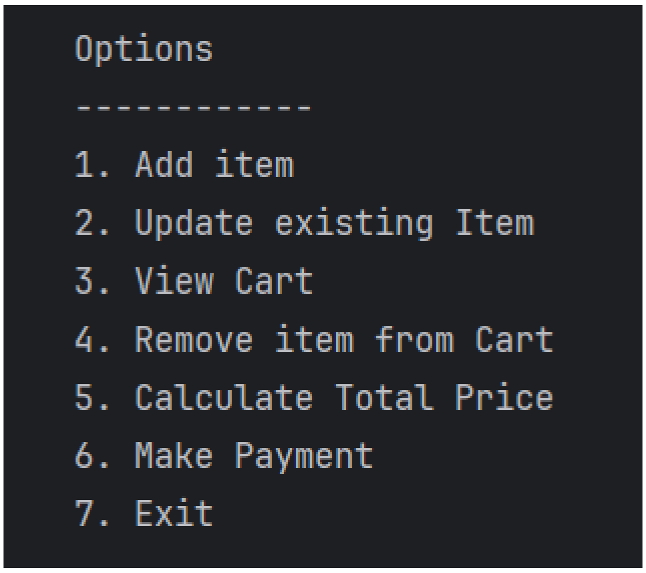
```scala
def updateItem(name: String, quantity: Int): Unit = {
  if (quantity < 0) {
    println("Quantity cannot be negative.")
  } else if (items.contains(name)) {
    if (quantity == 0) {
      items.remove(name)
      println("Item removed from the cart as quantity is set to zero.")
    } else {
      items(name).quantity = quantity
      println("Quantity updated successfully.")
    }
  } else {
    println("Item is not present in the cart.")
  }
}


def viewCart(): Unit = {
  if (items.isEmpty) {
    println("Empty Cart")
  } else {
    items.values.foreach { item =>
      println(s"${item.name} => quantity: ${item.quantity} => total: ${item.quantity * item.price} (without GST)")
    }
  }
}
```

```scala
def removeItem(name: String): Unit = {
  if (items.contains(name)) {
    items.remove(name)
    println(s"Removed $name from the cart.")
  } else {
    println("Cart doesn't contain this item.")
  }
}

def totPrice(): Double = {
  if (items.isEmpty) {
    println("Cart is Empty")
    0.0
  } else {
    val total_price = items.map(item => item._2.quantity
* item._2.price).sum
    val gst = total_price * GST_RATE
    val totalWithGST = total_price + gst
    val finalTotal = if (totalWithGST <
Delivery_Charge_Threshold) totalWithGST + Delivery_charge
else totalWithGST
    println(f"Cart Value:\t\t₹${total_price}%.2f")
    if (totalWithGST < Delivery_Charge_Threshold) {
      println(f"Delivery
Charge:\t₹${Delivery_charge}%.2f")
    }
    println(f"GST:\t\t\t₹${gst}%.2f")
    println(f"Amount Payable:\t\t₹${finalTotal}%.2f")
    finalTotal
  }
```

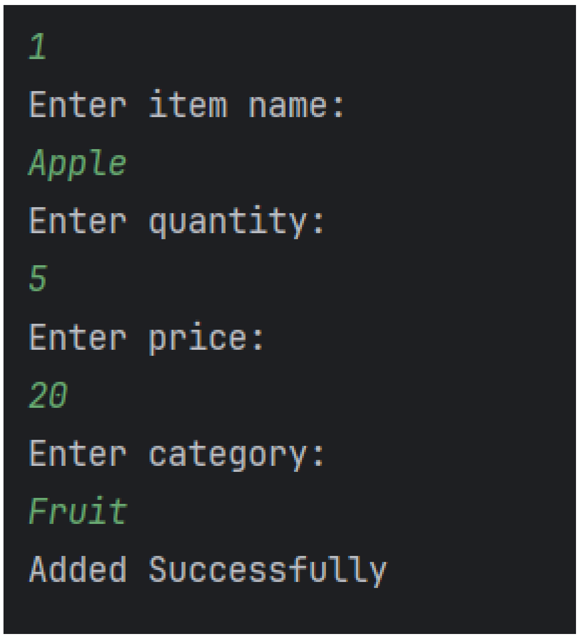
```scala
 }
}

object ShoppingCartApp extends App {
 val cart = new ShoppingCart()
 val paymentGateway = new PaymentGateway()

 var go = true

 while (go) {
   println(
     """
       Options
       ------------
       1. Add item
       2. Update existing Item
       3. View Cart
       4. Remove item from Cart
       5. Calculate Total Price
       6. Make Payment
       7. Exit
     """
   )
   val option = scala.io.StdIn.readInt()

   option match {
     case 1 =>
       println("Enter item name: ")
       val name = scala.io.StdIn.readLine()
```

```scala
        println("Enter quantity: ")
        val quantity = scala.io.StdIn.readInt()

        println("Enter price: ")
        val price = scala.io.StdIn.readDouble()

        println("Enter category: ")
        val category = scala.io.StdIn.readLine()

        cart.add(Item(name.toLowerCase(), quantity, price,
category.toLowerCase()))

      case 2 =>
        println("Enter item name to update: ")
        val name = scala.io.StdIn.readLine()

        println("Enter new quantity: ")
        val quantity = scala.io.StdIn.readInt()

        cart.updateItem(name.toLowerCase(), quantity)

      case 3 =>
        cart.viewCart()

      case 4 =>
        println("Enter item name to remove: ")
        val name = scala.io.StdIn.readLine()
        cart.removeItem(name.toLowerCase())
      case 5 =>
        cart.totPrice()

      case 6 =>
        println("Choose a payment method (Credit Card/Debit
Card/UPI): ")
        val paymentMethod = scala.io.StdIn.readLine()
        val amount = cart.totPrice()
        if (amount > 0) {
          println(paymentGateway.processPayment(amount,
paymentMethod))
        } else {
          println("Cannot proceed to payment. The cart is
empty or an error occurred.")
        }

      case 7 =>
        go = false
        println("Exiting...")

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

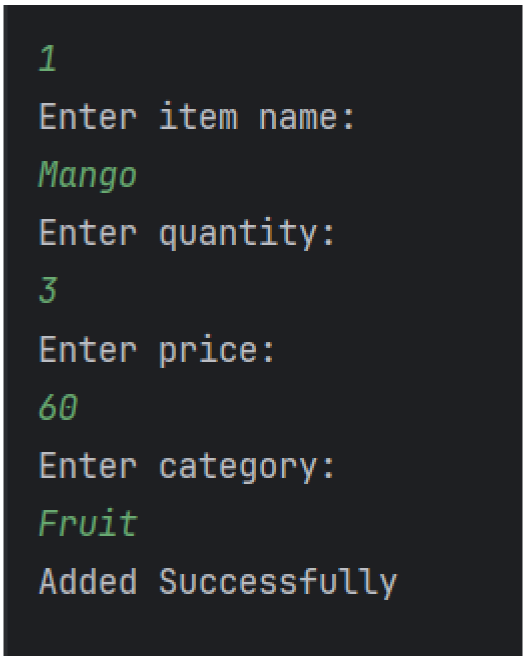
Outputs:



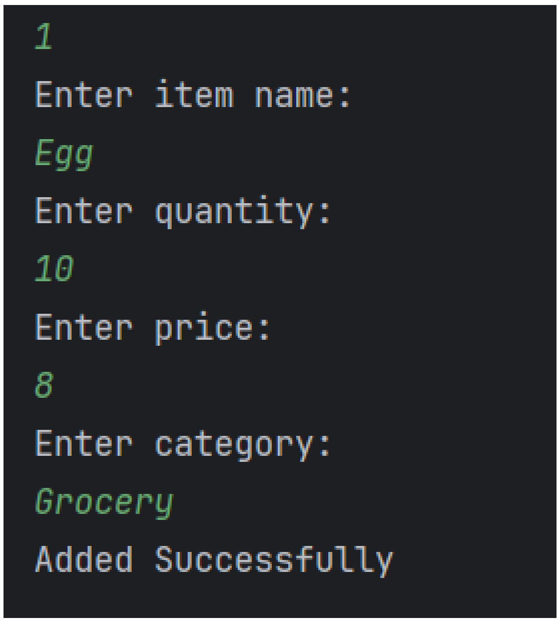
```
   Options
   ------------
   1. Add item
   2. Update existing Item
   3. View Cart
   4. Remove item from Cart
   5. Calculate Total Price
   6. Make Payment
   7. Exit
```



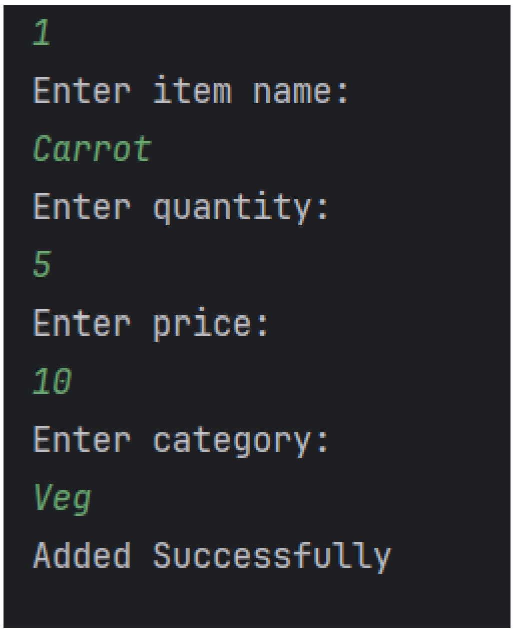
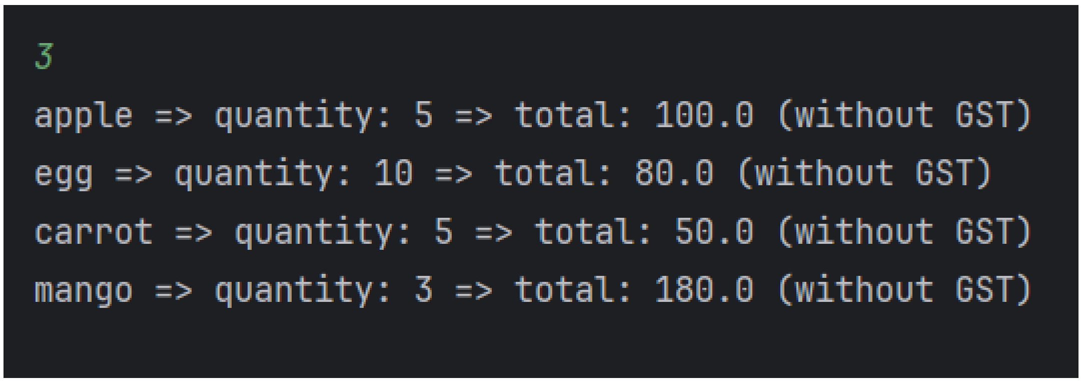
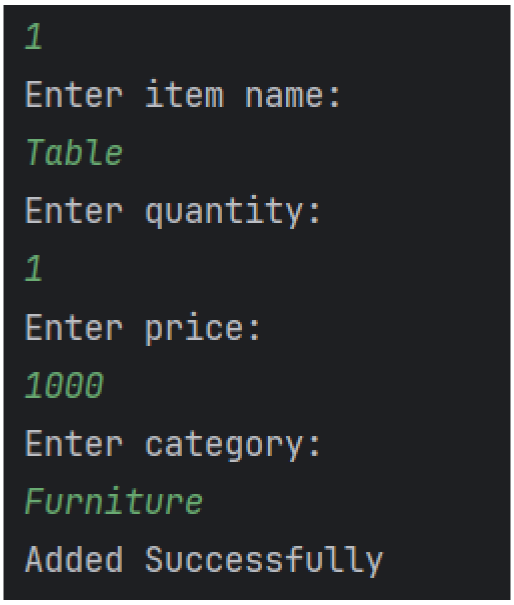
```
1
Enter item name:
Apple
Enter quantity:
5
Enter price:
20
Enter category:
Fruit
Added Successfully
```

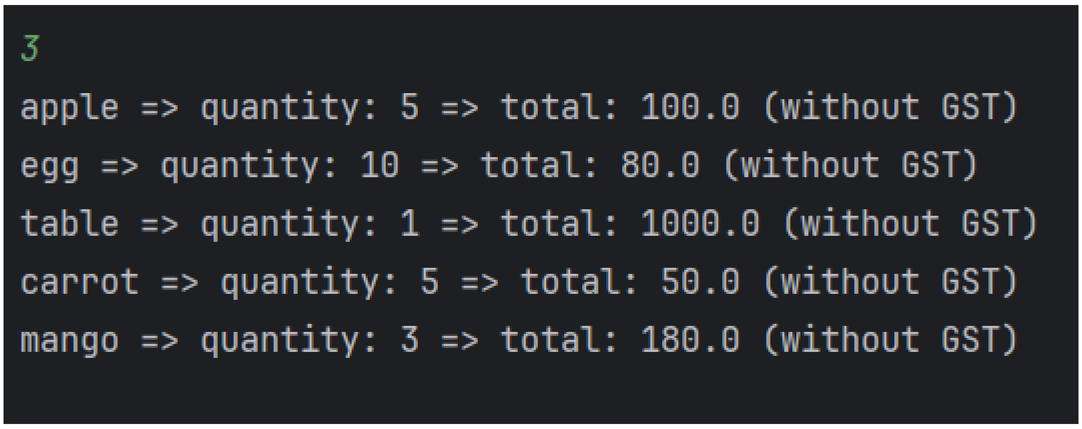


```
1
Enter item name:
Mango
Enter quantity:
3
Enter price:
60
Enter category:
Fruit
Added Successfully
```

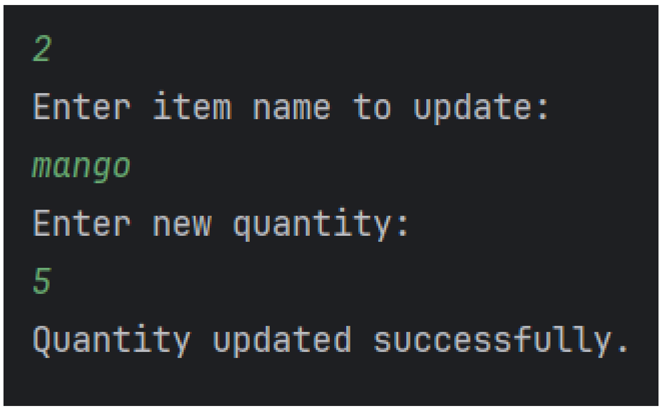


```
1
Enter item name:
Egg
Enter quantity:
10
Enter price:
8
Enter category:
Grocery
Added Successfully
```

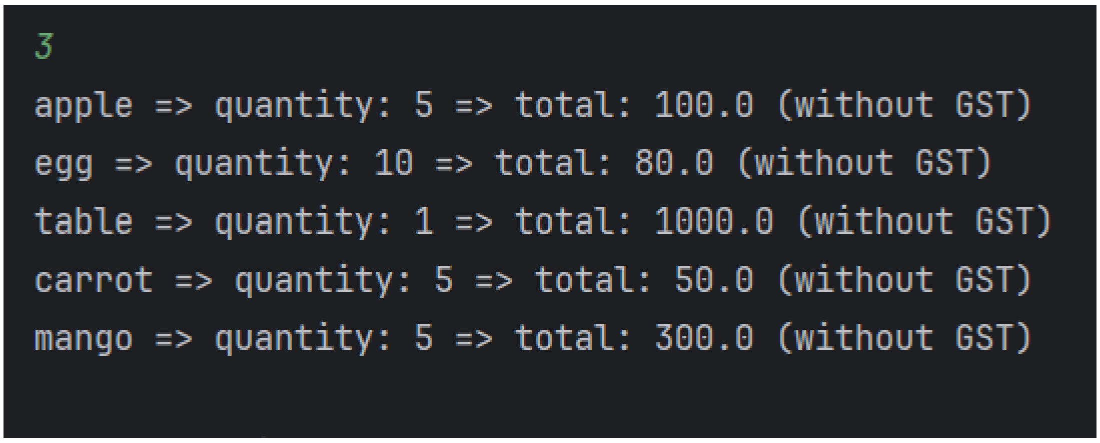

```
1
Enter item name:
Carrot
Enter quantity:
5
Enter price:
10
Enter category:
Veg
Added Successfully
```

```
3
apple => quantity: 5 => total: 100.0 (without GST)
egg => quantity: 10 => total: 80.0 (without GST)
carrot => quantity: 5 => total: 50.0 (without GST)
mango => quantity: 3 => total: 180.0 (without GST)
```

```
1
Enter item name:
Table
Enter quantity:
1
Enter price:
1000
Enter category:
Furniture
Added Successfully
```



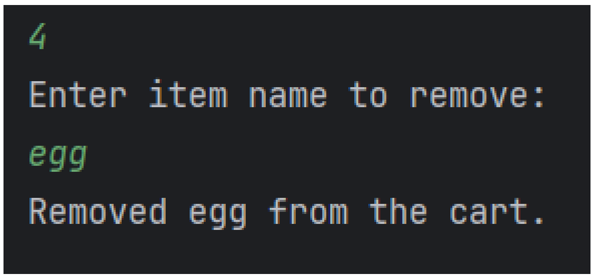
```
3
apple => quantity: 5 => total: 100.0 (without GST)
egg => quantity: 10 => total: 80.0 (without GST)
table => quantity: 1 => total: 1000.0 (without GST)
carrot => quantity: 5 => total: 50.0 (without GST)
mango => quantity: 3 => total: 180.0 (without GST)
```



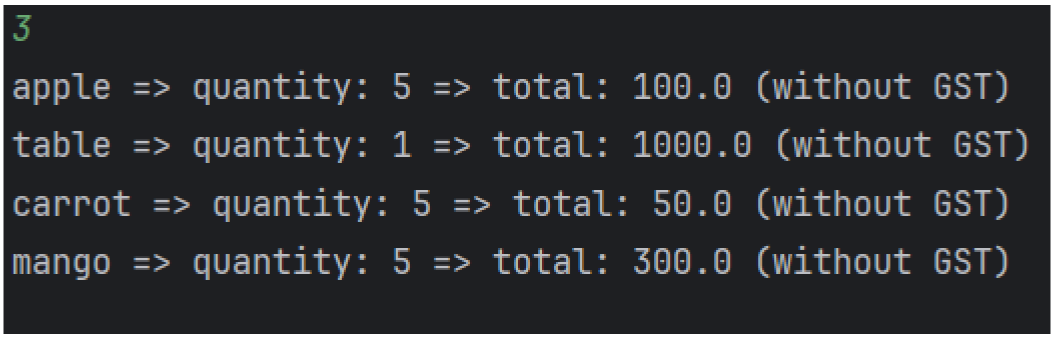
```
2
Enter item name to update:
mango
Enter new quantity:
5
Quantity updated successfully.
```

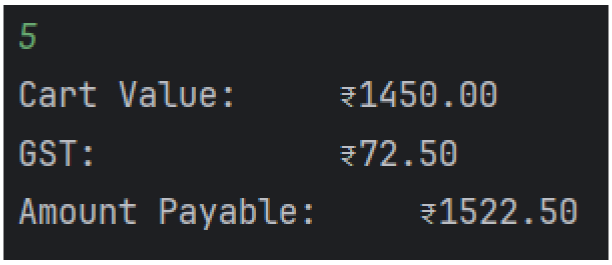


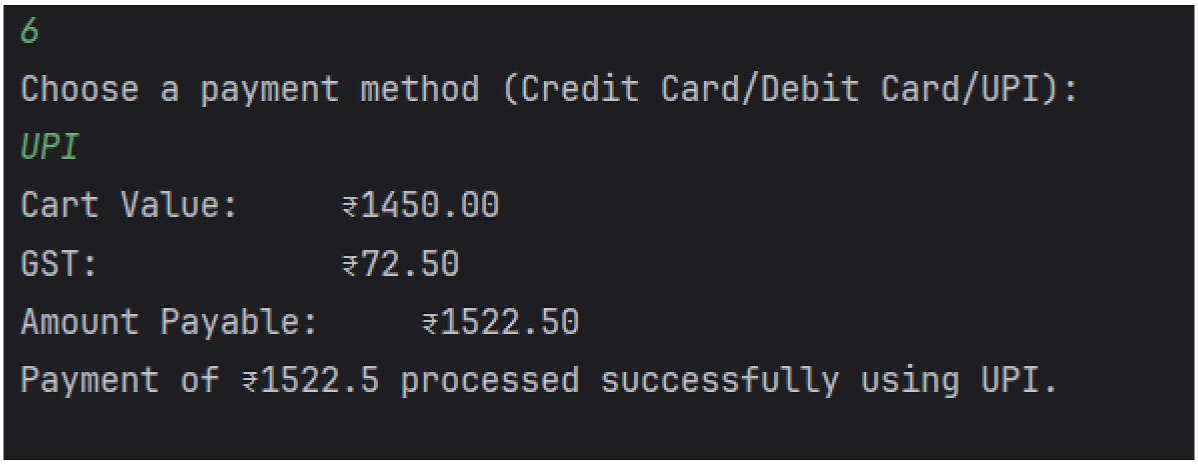
```
3
apple => quantity: 5 => total: 100.0 (without GST)
egg => quantity: 10 => total: 80.0 (without GST)
table => quantity: 1 => total: 1000.0 (without GST)
carrot => quantity: 5 => total: 50.0 (without GST)
mango => quantity: 5 => total: 300.0 (without GST)
```

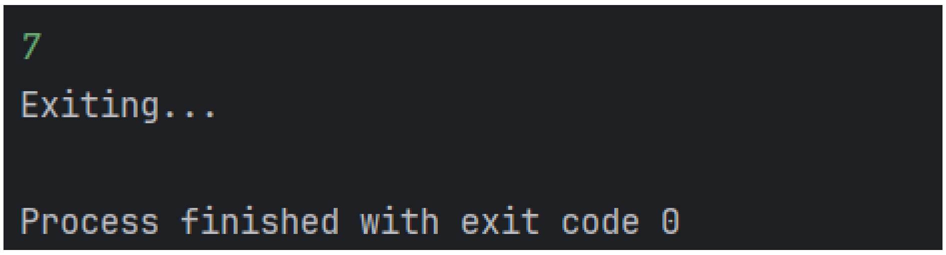


```
4
Enter item name to remove:
egg
Removed egg from the cart.
```


```
3
apple => quantity: 5 => total: 100.0 (without GST)
table => quantity: 1 => total: 1000.0 (without GST)
carrot => quantity: 5 => total: 50.0 (without GST)
mango => quantity: 5 => total: 300.0 (without GST)
```

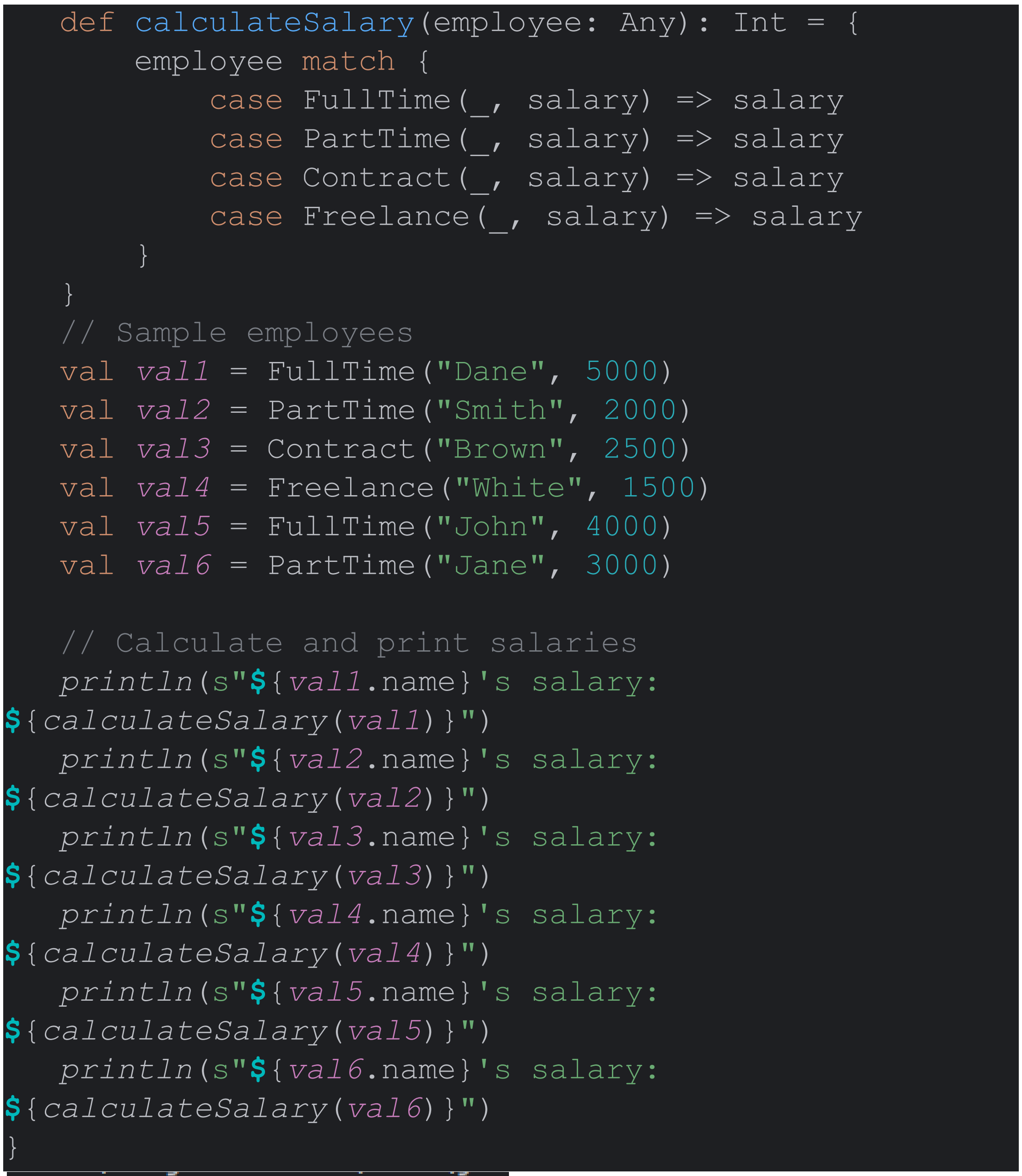

```
5
Cart Value:      ₹1450.00
GST:             ₹72.50
Amount Payable:      ₹1522.50
```


```
6
Choose a payment method (Credit Card/Debit Card/UPI):
UPI
Cart Value:      ₹1450.00
GST:             ₹72.50
Amount Payable:      ₹1522.50
Payment of ₹1522.5 processed successfully using UPI.
```

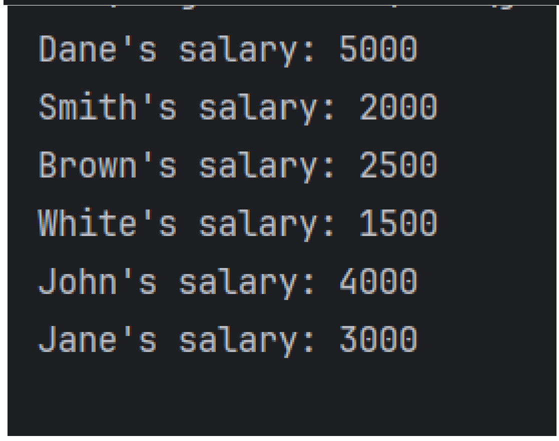

```
7
Exiting...

Process finished with exit code 0
```

3. **Problem Statement 3: Case Classes and Pattern Matching**
   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.

```scala
case class FullTime(name: String, salary: Int)
case class PartTime(name: String, salary: Int)
case class Contract(name: String, salary: Int)
case class Freelance(name: String, salary: Int)

object payroll extends App {
```



```scala
   def calculateSalary(employee: Any): Int = {
       employee match {
           case FullTime(_, salary) => salary
           case PartTime(_, salary) => salary
           case Contract(_, salary) => salary
           case Freelance(_, salary) => salary
       }
   }
   // Sample employees
   val val1 = FullTime("Dane", 5000)
   val val2 = PartTime("Smith", 2000)
   val val3 = Contract("Brown", 2500)
   val val4 = Freelance("White", 1500)
   val val5 = FullTime("John", 4000)
   val val6 = PartTime("Jane", 3000)

   // Calculate and print salaries
   println(s"${val1.name}'s salary:
${calculateSalary(val1)}")
   println(s"${val2.name}'s salary:
${calculateSalary(val2)}")
   println(s"${val3.name}'s salary:
${calculateSalary(val3)}")
   println(s"${val4.name}'s salary:
${calculateSalary(val4)}")
   println(s"${val5.name}'s salary:
${calculateSalary(val5)}")
   println(s"${val6.name}'s salary:
${calculateSalary(val6)}")
}
```

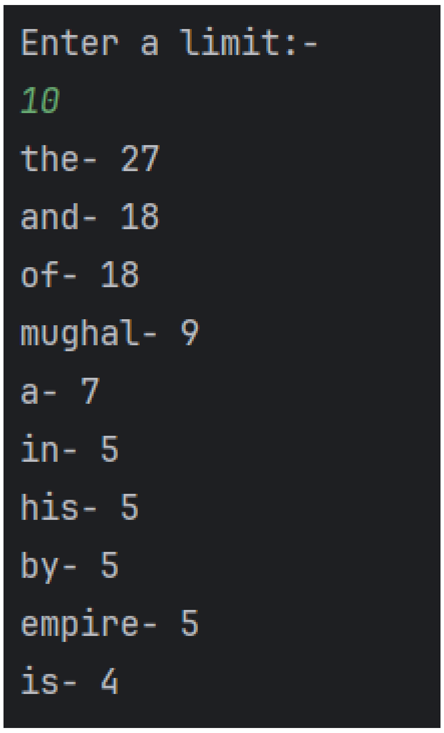


```
Dane's salary: 5000
Smith's salary: 2000
Brown's salary: 2500
White's salary: 1500
John's salary: 4000
Jane's salary: 3000
```

4. **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.

```scala
import scala.io.StdIn, scala.io.Source

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

   val filepath =
"C:\\Users\\Administrator\\IdeaProjects\\file1\\src\\main\
\scala\\mughals"
   val source = Source.fromFile(filepath)
     //reading N
   println("Enter a limit:- ")
   val ln = StdIn.readInt()

   val lines = source.getLines().toList  // read all the
files
   source.close()                        // closing read

   val words =
lines.flatMap(_.split("\\s+")).map(_.toLowerCase())
   val groupedWords = words.groupBy(identity)
   val wordCount =
groupedWords.mapValues(_.size).toSeq.sortBy(-_._2)
   wordCount.take(ln).foreach{case (word,count) =>
println(s"$word- $count")}

  }
}
```

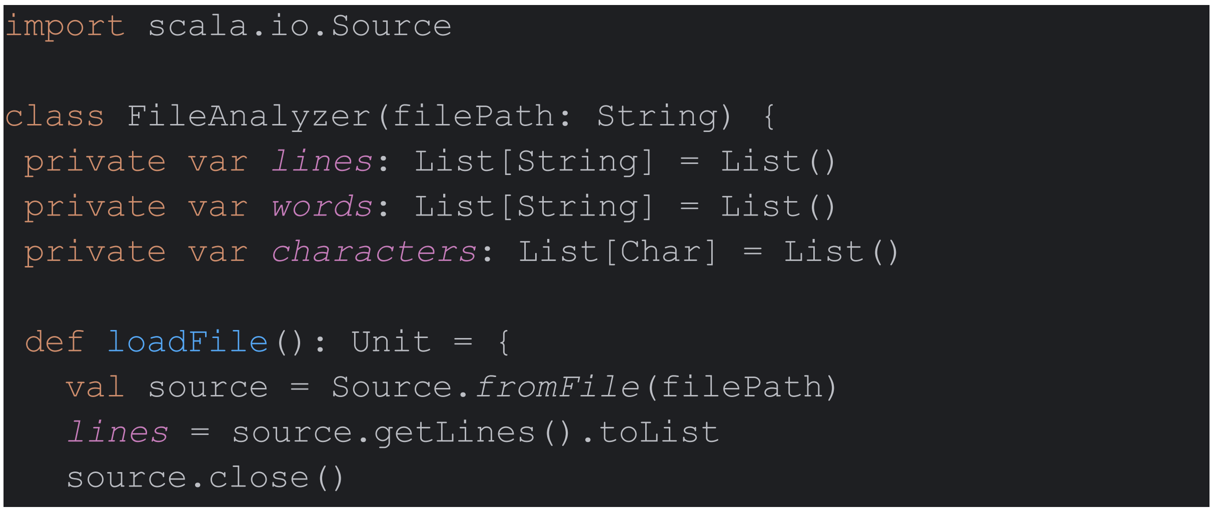


```
Enter a limit:-
10
the- 27
and- 18
of- 18
mughal- 9
a- 7
in- 5
his- 5
by- 5
empire- 5
is- 4
```

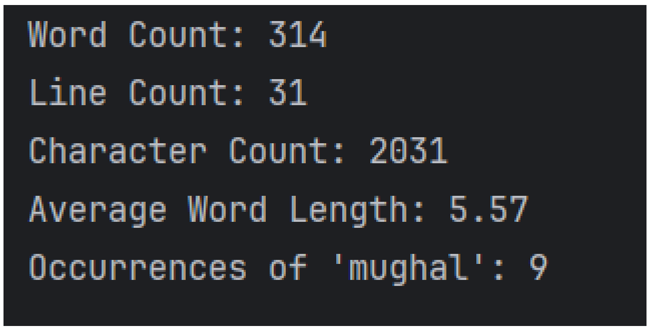
5. **Problem Statement 5: File Analysis Application in Scala**

   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.



```scala
import scala.io.Source

class FileAnalyzer(filePath: String) {
 private var lines: List[String] = List()
 private var words: List[String] = List()
 private var characters: List[Char] = List()

 def loadFile(): Unit = {
   val source = Source.fromFile(filePath)
   lines = source.getLines().toList
   source.close()
   words =
lines.flatMap(_.split("\\s+")).map(_.toLowerCase)
   characters = lines.flatMap(_.toCharArray)
 }

 def wordCount(): Int = words.length

 def lineCount(): Int = lines.length

 def characterCount(): Int = characters.length

 def averageWordLength(): Double = {
   val totalWords = words.length
   val totalChars = words.map(_.length).sum
   if (totalWords > 0) totalChars.toDouble / totalWords
else 0.0
 }

 def mostCommonStartingLetter(): Option[Char] = {
   if (words.isEmpty) None
   else {
     val startingLetters = words.map(_.head)

Some(startingLetters.groupBy(identity).maxBy(_._2.size)._1
)
   }
 }

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

object FileAnalysis {
 def main(args: Array[String]): Unit = {
   val filepath =
"C:\\Users\\Administrator\\IdeaProjects\\file1\\src\\main\
\scala\\mughals"
   val analyzer = new FileAnalyzer(filepath)
   analyzer.loadFile()

   println(s"Word Count: ${analyzer.wordCount()}")
   println(s"Line Count: ${analyzer.lineCount()}")
   println(s"Character Count:
${analyzer.characterCount()}")
   println(f"Average Word Length:
${analyzer.averageWordLength()}%.2f")
   println(s"Occurrences of 'mughal':
${analyzer.wordOccurrences("mughal")}")
   //println(s"Most Common Starting Letter:
${analyzer.mostCommonStartingLetter().getOrElse("None")}")

 }
}
```



```
Word Count: 314
Line Count: 31
Character Count: 2031
Average Word Length: 5.57
Occurrences of 'mughal': 9
```