Name - Sai Nikitha NSR, Shreedevi S B

Reg No - 3122225001118,3122225001129

Mini Project Report

Date - 22/12/23

Problem Statement

The problem addressed in this mini project is the development of a food delivery

application that allows users to browse a menu, customize their food items, place orders, and receive deliveries. The application includes a variety of food categories such as Pizza, Burger, Beverage, Appetizer, Dessert, and Pasta, each with customizable options.Lorem ipsum

Motivation of the problem

The motivation of this problem is to create a user-friendly and interactive food delivery management system that streamlines the whole ordering and delivery process. The project provides a practical implementation of object-oriented programming concepts and design principles.

Scope and Limitations

Scope

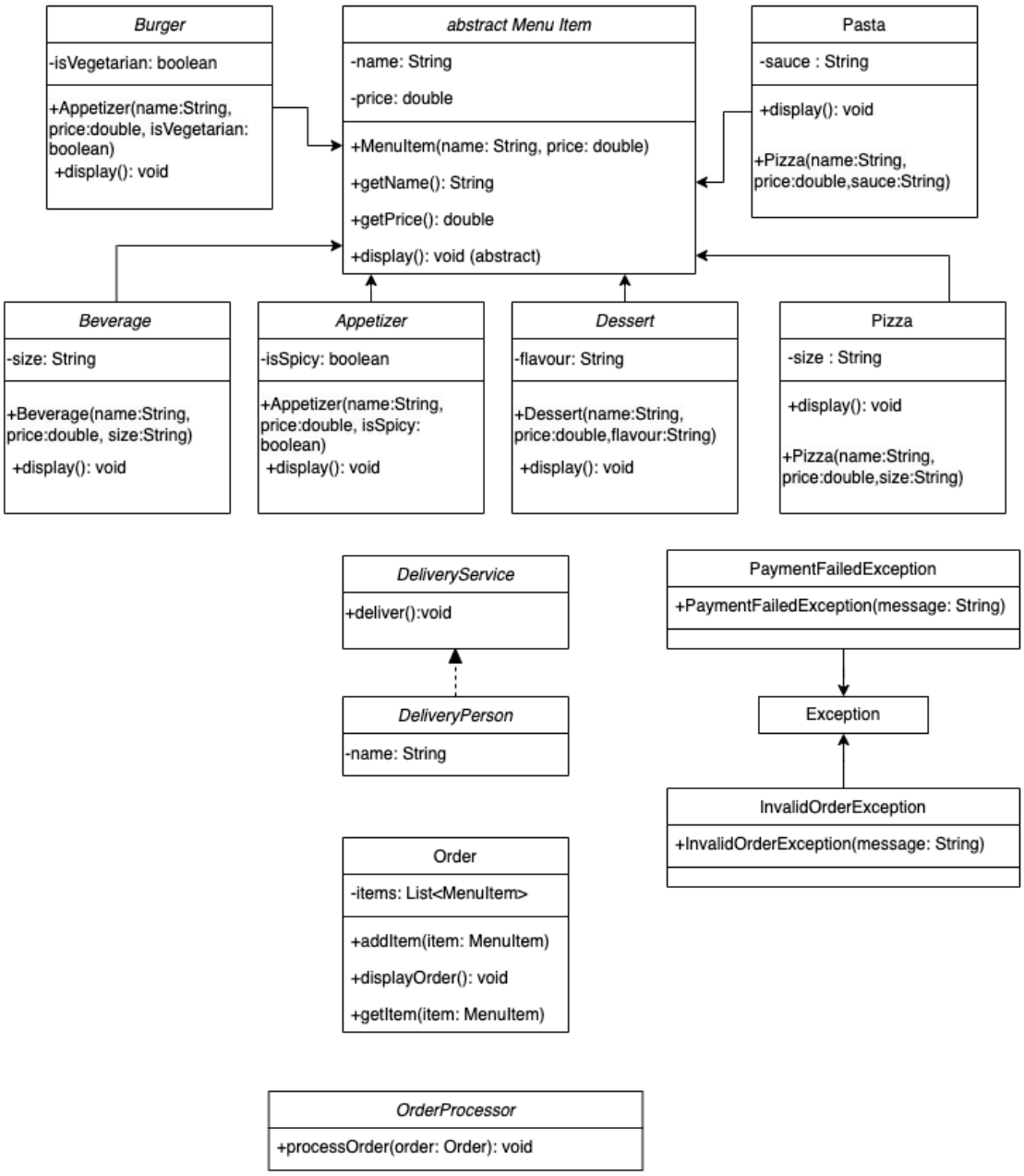
1. Order customization for different food categories.
2. Input handling of the program
3. Streamlined delivery process with a randomly selected delivery through file handling operations.
4. Exception handling for invalid ordering and payment failures.
5. Writing the order details to a bill file.

Limitations

1. Since it is a console-based application, it lacks a graphical user interface.
2. Since the payment failure is randomly generated there is no guarantee that the delivery will be successful.
3. There is no real time order updation of the delivery and the delivery person is chosen randomly.

Design of the solution (Class Diagram)

The solution is designed using object oriented principles, with a focus on encapsulation, inheritance, and polymorphism. The main class includes 'MenuItem' , 'Pizza' , 'Burger' , 'Beverage' , 'Appetizer' , 'Desert' , 'Pasta' , 'Order' , 'Menu' , 'DeliveryPerson' and "OrderProcessor'.


Burger
-isVegetarian: boolean
+Appetizer(name:String, price:double, isVegetarian: boolean)
+display(): void
abstract Menu Item
-name: String
-price: double
+MenuItem(name: String, price: double)
+getName(): String
+getPrice(): double
+display(): void (abstract)
Pasta
-sauce : String
+display(): void
+Pizza(name:String, price:double,sauce:String)
Beverage
-size: String
+Beverage(name:String, price:double, size:String)
+display(): void
Appetizer
-isSpicy: boolean
+Appetizer(name:String, price:double, isSpicy: boolean)
+display(): void
Dessert
-flavour: String
+Dessert(name:String, price:double,flavour:String)
+display(): void
Pizza
-size : String
+display(): void
+Pizza(name:String, price:double,size:String)
DeliveryService
+deliver():void
DeliveryPerson
-name: String
PaymentFailedException
+PaymentFailedException(message: String)
Exception
InvalidOrderException
+InvalidOrderException(message: String)
Order
-items: List<MenuItem>
+addItem(item: MenuItem)
+displayOrder(): void
+getItem(item: MenuItem)
OrderProcessor
+processOrder(order: Order): void

Modules Split-up:

1. **Menu Package :**

- It contains classes representing different food items as mentioned above.
- Each class extends the abstract class 'MenuItem' and includes customization options.

2. **DeliveryPersonClass :**

- It represents a delivery person with attributes like name and rating.
- Simulates the delivery process.

3. **Order Class :**

- Manages a list of items in the order.
- Calculates the total amount and displays the order details.

4. **Menu Class :**

- Manages a map of menu items.
- Provides methods to pass and retrieve items from the menu.

5. **OrderProcessor Class :**

- Simulates the entire order processing, handling exceptions for invalid orders and payment failures.

6. **FoodDeliveryApp Class :**

- Main class containing the application's entry point.
- Handles user input for ordering and coordinates the overall flow.

Implementation(Code):

Java

```java
//FoodDeliveryApp.java

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.util.*;

import menu.*;

class DeliveryPerson {

 private String name;

 private double rating;

 public DeliveryPerson(String name, double rating) {

  this.name = name;

  this.rating = rating;

 }

 public String getName() {

  return name;

 }
```

```
 public void deliver() {
  System.out.println(getName() + " will be delivering the
order...");
  System.out.println("Rating" + rating);
 }
}
class Order {
 private List<MenuItem> items;
 public Order() {
  items = new ArrayList<>();
 }
 public void addItem(MenuItem item) {
  items.add(item);
 }
 public List<MenuItem> getItems() {
  return items;
 }
 public void displayOrder() {
  for (MenuItem item : items) {
   item.display();
```

```
   item.displayCustomizationOptions(); // Display customization
options
  }
  public double calculateTotal() {
  double total = 0;
  for (MenuItem item : items) {
   total += item.getPrice();
  }
  return total;
 }
class Menu<T extends MenuItem> {
 private Map<String, T> items;
 public Menu() {
  items = new HashMap<>();
 }
 public void addItem(String key, T item) {
  items.put(key, item);
 }
 public T getItem(String key) {
  return items.get(key);
```

```
 }
 public Map<String, T> getItems() {
  return items;
 }
}
class InvalidOrderException extends Exception {
 public InvalidOrderException(String message) {
  super(message);
 }
}
class PaymentFailedException extends Exception {
 public PaymentFailedException(String message) {
  super(message);
 }
}
class OrderProcessor {
 public static void processOrder(Order order) throws
InvalidOrderException, PaymentFailedException {
  try {
   if (order == null || order.getItems().isEmpty()) {
```

```
    throw new InvalidOrderException("Invalid order: Order is
empty.");
   }
   if (somePaymentIssueOccurred()) {
    throw new PaymentFailedException("Payment failed: There was an
issue processing the payment.");
   }
  } catch (InvalidOrderException | PaymentFailedException e) {
   throw e;
  }
 }
 private static boolean somePaymentIssueOccurred() throws
PaymentFailedException {
  // Generate a random number between 1 and 50
  int randomNumber = new Random().nextInt(50) + 1;
  // Check if the random number is greater than 40
  if (randomNumber > 40) {
   throw new PaymentFailedException("Payment failed: There was an
issue processing the payment.");
  }
  // Simulate a situation where the payment issue did not occur
```

```
  return false;
 }
}
public class FoodDeliveryApp {
 private static Scanner scanner = new Scanner(System.in);
 private static List<DeliveryPerson> loadDeliveryPeople(String
filename) {
  List<DeliveryPerson> deliveryPeople = new ArrayList<>();
  try (BufferedReader br = new BufferedReader(new
FileReader(filename))) {
   String line;
   while ((line = br.readLine()) != null) {
    String[] parts = line.split(",");
    if (parts.length == 2) {
     String name = parts[0].trim();
     double rating = Double.parseDouble(parts[1].trim());
     DeliveryPerson deliveryPerson = new DeliveryPerson(name,
rating);
     deliveryPeople.add(deliveryPerson);
    }
   }
```

```
  } catch (IOException e) {
   System.out.println("Error loading delivery people data: " +
e.getMessage());
  }
  return deliveryPeople;
 }
 private static DeliveryPerson
getRandomDeliveryPerson(List<DeliveryPerson> deliveryPeople) {
  if (!deliveryPeople.isEmpty()) {
   Random random = new Random();
   return
deliveryPeople.get(random.nextInt(deliveryPeople.size()));
  } else {
   System.out.println("No delivery people available.");
   return null;
  }
 }
 public static void main(String[] args) {
  List<DeliveryPerson> deliveryPeople =
loadDeliveryPeople("delivery_people.csv");
```

```
  DeliveryPerson chosenDeliveryPerson =
getRandomDeliveryPerson(deliveryPeople);
  Menu<MenuItem> menu = createMenu();
  Order order = createOrder(menu);
  if (order != null) { // Check if the order is not null
   try {
    OrderProcessor.processOrder(order);
    chosenDeliveryPerson.deliver();
   } catch (InvalidOrderException | PaymentFailedException e) {
    System.out.println("Order processing failed: " +
e.getMessage());
   }
  }
  System.out.println("Exiting the application. Thank you!");
  // Close the scanner to prevent resource leaks
  scanner.close();
 }
 private static Menu<MenuItem> createMenu() {
  Menu<MenuItem> menu = new Menu<>();
  Scanner sc = new Scanner(System.in);
```

```
  menu.addItem("Margherita", new Pizza("Margherita", 199.00,
"Large", "Cheese, Tomato"));

  menu.addItem("Pepperoni", new Pizza("Pepperoni", 249.00, "Medium",
"Pepperoni, Cheese"));

  menu.addItem("Vegetarian", new Pizza("Vegetarian", 229.00,
"Large", "Mushrooms, Onions, Peppers"));

  menu.addItem("Hawaiian", new Pizza("Hawaiian", 449.00, "Medium",
"Ham, Pineapple, Cheese"));

  menu.addItem("MeatLovers", new Pizza("Meat Lovers", 499.00,
"Large", "Pepperoni, Sausage, Bacon"));

  menu.addItem("ClassicBurger", new Burger("Classic Burger",
159.00, false, true));

  menu.addItem("CheeseBurger", new Burger("Cheese Burger", 199.00,
true, true));

  menu.addItem("VeggieBurger", new Burger("Veggie Burger", 249.00,
false, true));

  menu.addItem("BaconBurger", new Burger("Bacon Burger", 299.00,
true, true));

  menu.addItem("SpicyChickenBurger", new Burger("Spicy Chicken
Burger", 339.00, false, true));


  menu.addItem("Cola", new Beverage("Cola", 80.00, "Medium"));

  menu.addItem("OrangeJuice", new Beverage("Orange Juice", 90.00,
"Large"));
```

```
  menu.addItem("IcedTea", new Beverage("Iced Tea", 100.00,
"Small"));

  menu.addItem("Lemonade", new Beverage("Lemonade", 80.00,
"Medium"));

  menu.addItem("Water", new Beverage("Water", 20.00, "Small"));

  menu.addItem("Nachos", new Appetizer("Nachos", 149.00, true));

  menu.addItem("MozzarellaSticks", new Appetizer("Mozzarella
Sticks", 149.00, false));

  menu.addItem("ChickenWings", new Appetizer("Chicken Wings",
249.00, true));

  menu.addItem("SpinachDip", new Appetizer("Spinach Dip", 45.00,
false));

  menu.addItem("OnionRings", new Appetizer("Onion Rings", 149.00,
false));

  menu.addItem("ChocolateCake", new Dessert("Chocolate Cake",
179.00, "Chocolate"));

  menu.addItem("Cheesecake", new Dessert("Cheesecake", 129.00,
"Plain"));

  menu.addItem("ApplePie", new Dessert("Apple Pie", 179.00,
"Apple"));

  menu.addItem("Brownie", new Dessert("Brownie", 99.00,
"Chocolate"));

  menu.addItem("IceCream", new Dessert("Ice Cream", 120.00,
"Vanilla"));
```

```
  menu.addItem("SpaghettiBolognese", new Pasta("Spaghetti
Bolognese", 259.85, "Bolognese Sauce"));

  menu.addItem("AlfredoPasta", new Pasta("Alfredo Pasta", 250.00,
"Alfredo Sauce"));

  menu.addItem("PestoPasta", new Pasta("Pesto Pasta", 349.00, "Pesto
Sauce"));

  menu.addItem("ChickenFettuccine", new Pasta("Chicken Fettuccine",
399.00, "Creamy Sauce"));

  menu.addItem("ShrimpScampi", new Pasta("Shrimp Scampi", 850.00,
"Garlic Butter Sauce"));

  return menu;

 }

 private static Order createOrder(Menu<MenuItem> menu) {

  Order order = new Order();

  boolean orderingComplete = false;  while (!orderingComplete) {

   System.out.println("Categories:");

   displayCategories();

   String category = scanner.nextLine();

   if (isValidCategory(category)) {

    if (category.equals("7")) {

     orderingComplete = true;
```

```
     break; // Skip the rest of the loop and go to the next
iteration
    } else if (category.equals("8")) {
     return null;
    }
    displayMenuByCategory(menu, category);
    System.out.print("Enter the number of the item you want to add
to your order: ");
    String input = scanner.nextLine();
    try {
     int itemNumber = Integer.parseInt(input);
     MenuItem menuItem = findMenuItemByNumber(menu, category,
itemNumber);
     if (menuItem != null) {
      if (menuItem instanceof Pizza) {
       Pizza pizza = (Pizza) menuItem;
       System.out.println("Customization Options for Pizza:");
       System.out.println("1. Size: " + pizza.getSize());
       System.out.println("2. Toppings: " + pizza.getToppings());
       System.out.print("Enter size for Pizza (Regular or Large,
or press Enter for default size): ");
```

```
        String newSize = scanner.nextLine().trim();
        if (!newSize.isEmpty()) {
         pizza.setSize(newSize);
        }
        System.out.print("Enter toppings for Pizza (or press Enter
for no toppings): ");
        String newToppings = scanner.nextLine().trim();
        if (!newToppings.isEmpty()) {
         pizza.setToppings(newToppings);
        }
      }
      else if (menuItem instanceof Burger) {
        Burger burger = (Burger) menuItem;
        System.out.println("Customization Options for Burger:");
        System.out.println("1. Has Cheese: " +
burger.isHasCheese());
        System.out.println("2. Fries Included: " +
burger.isFriesIncluded());
        System.out.print("1. Has Cheese (Yes/No): ");
        String cheeseInput = scanner.nextLine().trim();
        if (!cheeseInput.isEmpty()) {
```

```
          burger.setHasCheese(cheeseInput.equalsIgnoreCase("Yes"));
        }
        System.out.print("2. Fries Included (Yes/No): ");
        String friesInput = scanner.nextLine().trim();
        if (!friesInput.isEmpty()) {

burger.setFriesIncluded(friesInput.equalsIgnoreCase("Yes"));
        }
      }
      else if (menuItem instanceof Pasta) {
        Pasta pasta = (Pasta) menuItem;
        System.out.println("Customization Options for Pasta:");
        System.out.println("1. Sauce: " + pasta.getSauce());
        System.out.print("Enter sauce for Pasta: ");
        String newSauce = scanner.nextLine().trim();
        if (!newSauce.isEmpty()) {
          pasta.setSauce(newSauce);
        }
      }
      else if (menuItem instanceof Dessert) {
```

```
        Dessert dessert = (Dessert) menuItem;
        System.out.println("Customization Options for Dessert:");
        System.out.println("1. Flavor: " + dessert.getFlavor());
        System.out.print("Enter flavor for Dessert: ");
        String newFlavor = scanner.nextLine().trim();
        if (!newFlavor.isEmpty()) {
          dessert.setFlavor(newFlavor);
        }
      }
      else if (menuItem instanceof Appetizer) {
        Appetizer appetizer = (Appetizer) menuItem;
        System.out.println("Customization Options for
Appetizer:");
        System.out.println("1. Spicy: " + appetizer.isSpicy())
        System.out.print("Is it spicy? (Yes/No): ");
        String spicyInput = scanner.nextLine().trim();
        if (!spicyInput.isEmpty()) {
          appetizer.setSpicy(spicyInput.equalsIgnoreCase("Yes"));
        }
      }
```

```
      if (menuItem instanceof Beverage) {
        Beverage beverage = (Beverage) menuItem;
        System.out.println("Customization Options for Beverage:");
        System.out.println("1. Size: " + beverage.getSize());
        System.out.print("Enter size for Beverage (Small, Medium,
Large, or press Enter for default size): ");
        String newSize = scanner.nextLine().trim();
        if (!newSize.isEmpty()) {
          beverage.setSize(newSize);
        }
      }
      order.addItem(menuItem);
      System.out.println(menuItem.getName() + " added to the
order.");
     } else {
      System.out.println("Invalid item number. Please choose a
valid number from the menu.");
     }
    } catch (NumberFormatException e) {
     System.out.println("Invalid input. Please enter a number.");
    }
```

```
    } else {
     System.out.println("Invalid category. Please choose a valid category.");
    }
   }
   // Display order details after the loop
   if (orderingComplete) {
    displayColoredBill(order); // Display the entire order list as a colorful bill
    double totalAmount = order.calculateTotal();
    System.out.println("\u001B[1mTotal Amount: " + totalAmount + "/-\u001B[0m");
    // Write the bill to a separate file
    writeBillToFile(order, "bill.txt");

   }
   return order;
  }
  private static void displayColoredBill(Order order) {
   System.out.println("\u001B[1m\u001B[4mBill:\u001B[0m"); // Bold and underline the heading "Bill"
```

```
  for (MenuItem item : order.getItems()) {
   item.display();
   item.displayCustomizationOptions(); // Display customization
options
   System.out.println();
  }


 private static void writeBillToFile(Order order, String fileName) {
  try (BufferedWriter writer = new BufferedWriter(new
FileWriter(fileName))) {
   writer.write("Bill:");
   writer.newLine();
   for (MenuItem item : order.getItems()) {
    writer.write(item.getName() + ", Price: ₹" + item.getPrice());
    writer.newLine();
    writer.newLine();
   }
   double totalAmount = order.calculateTotal();
   writer.write("Total Amount: ₹" + totalAmount);
  } catch (IOException e) {
```

```
   System.out.println("Error writing the bill to the file: " +
e.getMessage());
  }
 }
 private static MenuItem findMenuItemByNumber(Menu<MenuItem> menu,
String category, int itemNumber) {
  int currentNumber = 1;
  for (MenuItem item : menu.getItems().values()) {
   String itemName = item.getClass().getSimpleName();
   if (isValidCategory(category) && itemBelongsToCategory(item,
category)) {
    if (currentNumber == itemNumber) {
     return item;
    }
    currentNumber++;
   }
  }
  return null;
 }
 private static boolean itemBelongsToCategory(MenuItem item, String
category) {
```

```
 switch (category) {
  case "1":
   return item instanceof Pizza;
  case "2":
   return item instanceof Burger;
  case "3":
   return item instanceof Beverage;
  case "4":
   return item instanceof Appetizer;
  case "5":
   return item instanceof Dessert;
  case "6":
   return item instanceof Pasta;
  default:
   return false;
 }
}
private static void displayCategories() {
 System.out.println("1. Pizza");
 System.out.println("2. Burger");
```

```
  System.out.println("3. Beverage");
  System.out.println("4. Appetizer");
  System.out.println("5. Dessert");
  System.out.println("6. Pasta");
  System.out.println("7. Done Ordering");
  System.out.println("8. Exit application");
  System.out.println("Choose a category by entering its
number:");
 }
 private static boolean isValidCategory(String category) {
  return category.matches("[1-8]");
 }
 private static void displayMenuByCategory(Menu<MenuItem> menu,
String category) {
  switch (category) {
   case "1":
    displayMenu(menu, Pizza.class);
    break;
   case "2":
    displayMenu(menu, Burger.class);
```

```
    break;
   case "3":
    displayMenu(menu, Beverage.class);
    break;
   case "4":
    displayMenu(menu, Appetizer.class);
    break;
   case "5":
    displayMenu(menu, Dessert.class);
    break;
   case "6":
    displayMenu(menu, Pasta.class);
    break;
  }
 }
 private static <T extends MenuItem> void displayMenu(Menu<MenuItem>
menu, Class<T> categoryClass) {
  int itemNumber = 1;
  for (MenuItem item : menu.getItems().values()) {
   if (categoryClass.isInstance(item)) {
```

```
    System.out.println(itemNumber + ". " + item.getName());
    itemNumber++; }}}}
```

Java

```java
//Appetizer class
package menu;
import java.util.Scanner;
public class Appetizer extends MenuItem {
  private boolean isSpicy;
  public Appetizer(String name, double price, boolean isSpicy) {
    super(name, price);
    this.isSpicy = isSpicy;
  }
  public boolean isSpicy() {
      return isSpicy;
  }
  public void setSpicy(boolean isSpicy) {
      this.isSpicy = isSpicy;
  }
```

```
  @Override
  public void display() {
    String spiceLevel = isSpicy ? "Spicy" : "Mild";
    System.out.println("\u001B[31mAppetizer: " + getName() + ",
Spice Level: " + spiceLevel + ", Price: ₹"
        + getPrice() + "\u001B[0m");
  }
  public void displayCustomizationOptions(Scanner scanner) {
    System.out.println("Customization Options for Appetizer:");
    System.out.println("1. Spicy: " + isSpicy());
    System.out.print("Is it spicy? (Yes/No): ");
    String spicyInput = scanner.nextLine().trim();
    if (!spicyInput.isEmpty()) {
      setSpicy(spicyInput.equalsIgnoreCase("Yes"));
    }
  } }
```

Java

```
package menu;
import java.util.Scanner;
```

```
public class Beverage extends MenuItem {
 private String size;
 public Beverage(String name, double price, String size) {
  super(name, price);
  this.size = size;
 }
 public String getSize() {
   return size;
 }
 public void setSize(String size) {
   this.size = size;
 }
 @Override
 public void display() {
  System.out.println(
    "\u001B[34mBeverage: " + getName() + ", Size: " + size + ", Price:
₹" + getPrice() + "\u001B[0m");
 }
 public void displayCustomizationOptions(Scanner scanner) {
  System.out.println("Customization Options for Beverage:");
```

```
  System.out.println("1. Size: " + getSize());

  System.out.print("Enter size for Beverage (Small, Medium,
Large, or press Enter for default size): ");

  String newSize = scanner.nextLine().trim();

  if (!newSize.isEmpty()) {

   setSize(newSize);

  }

 }

}
```

Java

```
package menu;

import java.util.Scanner;

public class Burger extends MenuItem {

 private boolean hasCheese;

 private boolean friesIncluded; // New field for fries

 public Burger(String name, double price, boolean hasCheese, boolean
friesIncluded) {

  super(name, price);

  this.hasCheese = hasCheese;
```

```
 this.friesIncluded = friesIncluded;
}
public boolean isHasCheese() {
 return hasCheese;
}
public void letHasCheese(boolean hasCheese) {
 this.hasCheese = hasCheese;
}
public boolean isFriesIncluded() {
 return friesIncluded;
}
public void setFriesIncluded(boolean friesIncluded) {
 this.friesIncluded = friesIncluded;
}
@Override
public void display() {
 System.out.println("\u001B[32mBurger: " + getName() + ", Has
Cheese: " + hasCheese + ", Fries Included: "
   + friesIncluded + ", Price: ₹" + getPrice() + "\u001B[0m");
}
```

```
 public void displayCustomizationOptions(Scanner scanner) {
  System.out.println("Customization Options for Burger:");
  System.out.println("1. Has Cheese: " + isHasCheese());
  System.out.println("2. Fries Included: " + isFriesIncluded());
  System.out.print("1. Has Cheese (Yes/No): ");
  String cheeseInput = scanner.nextLine().trim();
  if (!cheeseInput.isEmpty()) {
   setHasCheese(cheeseInput.equalsIgnoreCase("Yes"));
  }
  System.out.print("2. Fries Included (Yes/No): ");
  String friesInput = scanner.nextLine().trim();
  if (!friesInput.isEmpty()) {
   setFriesIncluded(friesInput.equalsIgnoreCase("Yes"));
  }
 }
}
```

Java

```
package menu;
```

```
import java.util.Scanner;

public class Dessert extends MenuItem {

 private String flavor;

 public Dessert(String name, double price, String flavor) {

  super(name, price);

  this.flavor = flavor;

 }

 public String getFlavor() {

   return flavor;

 }

 public void setFlavor(String flavor) {

   this.flavor = flavor;

 }

 public void display() {

  System.out.println(

    "\u001B[35mDessert: " + getName() + ", Flavor: " + flavor + ",
Price: ₹" + getPrice() + "\u001B[0m");

 }

 public void displayCustomizationOptions(Scanner scanner) {

  System.out.println("Customization Options for Dessert:");
```

```
 System.out.println("1. Flavor: " + getFlavor());
 System.out.print("Enter flavor for Dessert: ");
 String newFlavor = scanner.nextLine().trim();
 if (!newFlavor.isEmpty()) {
  setFlavor(newFlavor);
 }
}}
```

Java

```
package menu;
public abstract class MenuItem {
 private String name;
 private double price;
 public MenuItem(String name, double price) {
  this.name = name;
  this.price = price;
 }
 public String getName() {
  return name;
```

```
 }
 public double getPrice() {
  return price;
 }
 public abstract void display();
 // Add a method to display customization options
 public void displayCustomizationOptions() {
  // Default implementation (can be overridden by subclasses)
 }}
```

Java

```
package menu;
import java.util.Scanner;
public class Pizza extends MenuItem {
 private String size;
 private String toppings;
 public Pizza(String name, double price, String size, String toppings)
{
  super(name, price);
  this.size = size;
```

```
 this.toppings = toppings;
 }
 public String getSize() {
  return size;
 }
 public void setSize(String size) {
  this.size = size;
 }
 public String getToppings() {
  return toppings;
 }
 public void setToppings(String toppings) {
  this.toppings = toppings;
 }
 @Override
 public void display() {
  System.out.println("\u001B[32mPizza: " + getName() + ", Size: " +
size + ", Price: ₹" + getPrice() + "\u001B[0m");
 }
 public void displayCustomizationOptions() {
```

```
  System.out.println("Customization Options for Pizza:");
  System.out.println("1. Size: " + size);
  System.out.println("2. Toppings: " + toppings);
 }
}
```

Java

```
package menu;
import java.util.Scanner;
public class Pizza extends MenuItem {
 private String size;
 private String toppings;
 public Pizza(String name, double price, String size, String toppings)
{
  super(name, price);
  this.size = size;
  this.toppings = toppings;
 }
 public String getSize() {
  return size;
```

```
 }
 public void setSize(String size) {
  this.size = size;
 }
 public String getToppings() {
  return toppings;
 }
 public void setToppings(String toppings) {
  this.toppings = toppings;
 }
 @Override
 public void display() {
  System.out.println("\u001B[32mPizza: " + getName() + ", Size: " +
size + ", Price: ₹" + getPrice() + "\u001B[0m");
 }
 public void displayCustomizationOptions() {
  System.out.println("Customization Options for Pizza:");
  System.out.println("1. Size: " + size);
  System.out.println("2. Toppings: " + toppings);
 }
```

```
}
```

Java

```java
package menu;

import java.util.Scanner;

public class Pasta extends MenuItem {

 private String sauce;

 public Pasta(String name, double price, String sauce) {

  super(name, price);

  this.sauce = sauce;

 }

 public String getSauce() {

   return sauce;

 }

 public void setSauce(String sauce) {

   this.sauce = sauce;

 }

 @Override

 public void display() {
```

```
  System.out.println(
    "\u001B[36mPasta: " + getName() + ", Sauce: " + sauce + ", Price:
₹" + getPrice() + "\u001B[0m");
 }
 public void displayCustomizationOptions(Scanner scanner) {
  System.out.println("Customization Options for Pasta:");
  System.out.println("1. Sauce: " + getSauce());
  System.out.print("Enter sauce for Pasta: ");
  String newSauce = scanner.nextLine().trim();
  if (!newSauce.isEmpty()) {
   setSauce(newSauce); }}}
```

Output Screenshots :

Ideal Case

```
(base) tanu@Tanushrees-MacBook-Air MiniProject % java FoodDeliveryApp
Categories:
1. Pizza
2. Burger
3. Beverage
4. Appetizer
5. Dessert
6. Pasta
7. Done Ordering
8. Exit application
Choose a category by entering its number:
6
1. Shrimp Scampi
2. Alfredo Pasta
3. Chicken Fettuccine
4. Spaghetti Bolognese
5. Pesto Pasta
Enter the number of the item you want to add to your order: 3
Customization Options for Pasta:
1. Sauce: Creamy Sauce
Enter sauce for Pasta: Creamy Sauce
Chicken Fettuccine added to the order.
Categories:
1. Pizza
```

```
2. Burger
3. Beverage
4. Appetizer
5. Dessert
6. Pasta
7. Done Ordering
8. Exit application
Choose a category by entering its number:
5
1. Cheesecake
2. Brownie
3. Apple Pie
4. Ice Cream
5. Chocolate Cake
Enter the number of the item you want to add to your order: 4
Customization Options for Dessert:
1. Flavor: Vanilla
Enter flavor for Dessert: Vanilla
Ice Cream added to the order.
Categories:
1. Pizza
2. Burger
3. Beverage
4. Appetizer
```

```
5. Dessert
6. Pasta
7. Done Ordering
8. Exit application
Choose a category by entering its number:
7
Bill:
Pasta: Chicken Fettuccine, Sauce: Creamy Sauce, Price: ₹399.0

Dessert: Ice Cream, Flavor: Vanilla, Price: ₹120.0

Total Amount: 519.0/-
Jane Smith will be delivering the order...
Rating 4.8
Exiting the application. Thank you!
(base) tanu@Tanushrees-MacBook-Air MiniProject %
```

Payment Failed Exception Case:

```
(base) tanu@Tanushrees-MacBook-Air MiniProject % java FoodDeliveryApp
Categories:
1. Pizza
2. Burger
3. Beverage
4. Appetizer
5. Dessert
6. Pasta
7. Done Ordering
8. Exit application
Choose a category by entering its number:
3
1. Water
2. Orange Juice
3. Lemonade
4. Cola
5. Iced Tea
Enter the number of the item you want to add to your order: 4
Customization Options for Beverage:
1. Size: Medium
Enter size for Beverage (Small, Medium, Large, or press Enter for default si
ze): Large
Cola added to the order.
Categories:
1. Pizza
2. Burger
3. Beverage
4. Appetizer
5. Dessert
6. Pasta
7. Done Ordering
8. Exit application
Choose a category by entering its number:
2
1. Spicy Chicken Burger
2. Classic Burger
3. Bacon Burger
4. Veggie Burger
5. Cheese Burger
```

```
5. Cheese Burger
Enter the number of the item you want to add to your order: 3
Customization Options for Burger:
1. Has Cheese: true
2. Fries Included: true
1. Has Cheese (Yes/No): Yes
2. Fries Included (Yes/No): Yes
Bacon Burger added to the order.
Categories:
1. Pizza
2. Burger
3. Beverage
4. Appetizer
5. Dessert
6. Pasta
7. Done Ordering
8. Exit application
Choose a category by entering its number:
7
Bill:
Beverage: Cola, Size: Large, Price: ₹80.0

Burger: Bacon Burger, Has Cheese: true, Fries Included: true, Price: ₹299.0

Total Amount: 379.0/-
Order processing failed: Payment failed: There was an issue processing the p
ayment.
Exiting the application. Thank you!
(base) tanu@Tanushrees-MacBook-Air MiniProject %
```

Exit Application case:

```
(base) tanu@Tanushrees-MacBook-Air MiniProject % java FoodDeliveryApp
Categories:
1. Pizza
2. Burger
3. Beverage
4. Appetizer
5. Dessert
6. Pasta
7. Done Ordering
8. Exit application
Choose a category by entering its number:
1
1. Meat Lovers
2. Hawaiian
3. Margherita
4. Vegetarian
5. Pepperoni
Enter the number of the item you want to add to your order: 5
Customization Options for Pizza:
1. Size: Medium
2. Toppings: Pepperoni, Cheese
Enter size for Pizza (Regular or Large, or press Enter for default size): Medium
Enter toppings for Pizza (or press Enter for no toppings):
Pepperoni added to the order.
```

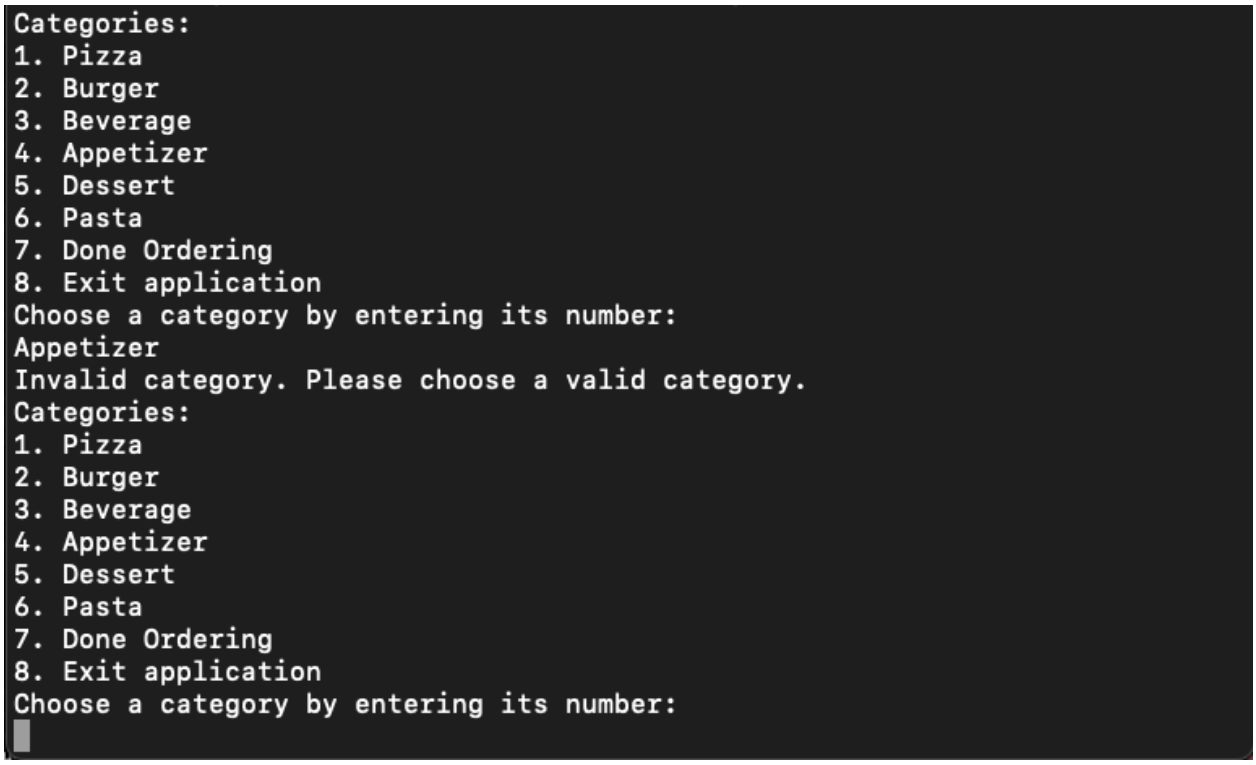
```
Categories:
1. Pizza
2. Burger
3. Beverage
4. Appetizer
5. Dessert
6. Pasta
7. Done Ordering
8. Exit application
Choose a category by entering its number:
3
1. Water
2. Orange Juice
3. Lemonade
4. Cola
5. Iced Tea
Enter the number of the item you want to add to your order: 3
Customization Options for Beverage:
1. Size: Medium
Enter size for Beverage (Small, Medium, Large, or press Enter for default size):
 Medium
Lemonade added to the order.
Categories:
1. Pizza
```

```
2. Burger
3. Beverage
4. Appetizer
5. Dessert
6. Pasta
7. Done Ordering
8. Exit application
Choose a category by entering its number:
5
1. Cheesecake
2. Brownie
3. Apple Pie
4. Ice Cream
5. Chocolate Cake
Enter the number of the item you want to add to your order: 1
Customization Options for Dessert:
1. Flavor: Plain
Enter flavor for Dessert: Plain
Cheesecake added to the order.
Categories:
1. Pizza
2. Burger
3. Beverage
4. Appetizer
```

```
5. Dessert
6. Pasta
7. Done Ordering
8. Exit application
Choose a category by entering its number:
8
Exiting the application. Thank you!
(base) tanu@Tanushrees-MacBook-Air MiniProject %
```

Invalid Input Exception:

```
(base) tanu@Tanushrees-MacBook-Air MiniProject % java FoodDeliveryApp
Categories:
1. Pizza
2. Burger
3. Beverage
4. Appetizer
5. Dessert
6. Pasta
7. Done Ordering
8. Exit application
Choose a category by entering its number:
10
Invalid category. Please choose a valid category.
```



Object-Oriented Features Used:

Our Food Delivery App, SnapBite, leverages key principles, facilitating a modular and extensible architecture. Encapsulation is used to encapsulate the properties of food items with their distinct features by using separate classes such as Pizza, Dessert etc. Inheritance is employed by using a base class MenuItem which is a super class of all the specific menu item classes such as Burger, Beverage, Appetizer and others. The method display is made an abstract method, which in turn makes the class MenuItem an abstract class. This display() method is then given a different display() implementation in every subclass thus implementing abstraction. The method displayCustomization() in class MenuItem is overridden by the subclasses to display specific customizations for each subclass. This shows polymorphism. The code also handles exceptions, uses packages for modularity and employs the use of Generic Types and Collections Framework which makes it very easy to use and interactive.

Inference and Future Extension:

The inference from the Food Delivery Application project is that it successfully demonstrates key principles of object-oriented programming (OOP) such as encapsulation, inheritance, and polymorphism. The modular design, encapsulating different food items into classes and organizing them within a package structure, enhances code readability and maintainability. The inheritance hierarchy, with a common base class (MenuItem) and specialized subclasses (Pizza, Burger, Beverage, etc.), promotes code reusability and facilitates the addition of new food items with minimal modifications.

Future extensions of this project involve incorporating additional features to enhance user experience and functionality.

1. User Interface (UI) Development: Integrate a graphical user interface (GUI) to provide a more user-friendly experience for customers and restaurant staff. This could include order customization, real-time tracking, and interactive menus.

2. Database Integration: Implement a database system to store and manage menu items, user profiles, and order history. This would allow for persistent data storage, user account management, and personalized recommendations based on order history.

3. User Authentication and Authorization: Enhance security by implementing user authentication and authorization mechanisms. This ensures that only authorized users can place orders and access certain functionalities.

4. Payment Integration: Integrate secure payment gateways to enable online transactions, supporting various payment methods and ensuring the confidentiality of sensitive information.

5. Order Management System: Develop a comprehensive order management system for restaurant staff, including order processing, inventory management, and order tracking.

7. Feedback and Rating System: Implement a feedback and rating system to gather customer opinions, allowing restaurants to enhance their services based on user feedback.

By considering these future extensions, the Food Delivery Application can evolve into a robust and feature-rich platform that meets the evolving needs of users and businesses in the food delivery industry.