**Introduction:**

In this assignment, we explore the concept of a Coffee Shop management system is built on object-oriented programming (OOP). The Coffee Shop system consists of various classes representing different entities such as beverages, customers, and the coffee shop itself. The primary objective of the system is to facilitate the ordering and management of coffee items within the coffee shop.

**Design:**

The simplified class diagram for the library application is as follows:

**Main/driver class:** CoffeeShopTest

**Parent class:** Beverage

**Child class:** Coffee, Tea,customer

**Group class:** CoffeeShop

* The CoffeeShopTest class serves as the main entry point for the application and contains the main method. It's responsible for testing the functionality of the coffee shop.
* The Beverage class represents a generic beverage item with attributes such as name, price, and availability. It serves as the parent class for specific types of beverages.
* Coffee and Tea class inherits from the Beverage class, representing a specific type of beverage, which is coffee and tea. It extends the functionality of Beverage by providing methods specific to coffee and tea items. Customer class represents a customer of the coffee shop. It contains a name attribute and a list of ordered coffees. It provides methods to retrieve the customer's name and ordered coffees, as well as to place an order.
* CoffeeShop class acts as a manager class for the coffee shop operations. It maintains a list of coffees in the menu and a list of customers. The CoffeeShop class provides methods to add coffees and customers, display the menu, search for coffee by name, serve coffee to customers, and display customer orders.

**Actual Code :**

class **Beverage** {

protected String name;

protected double price;

protected boolean available;

public Beverage(String name, double price) {

this.name = name;

this.price = price;

this.available = true;

}

public String getName() {

return name;

}

public double getPrice() {

return price;

}

public boolean isAvailable() {

return available;

}

public void setAvailable(boolean available) {

this.available = available;

}

@Override

public String toString() {

return name + " - $" + price + " - " + (available ? "Available" : "Out of Stock");

}

}

class **Coffee** extends **Beverage** {

public Coffee(String name, double price) {

super(name, price);

}

}

class **Tea** extends **Beverage** {

public Tea(String name, double price) {

super(name, price);

}

}

import java.util.ArrayList;

import java.util.List;

class **CoffeeShop** {

private String name;

private List<Beverage> menu;

private List<Customer> customers;

public CoffeeShop(String name) {

this.name = name;

this.menu = new ArrayList<>();

this.customers = new ArrayList<>();

}

public void addBeverage(Beverage beverage) {

menu.add(beverage);

}

public void addCustomer(Customer customer) {

customers.add(customer);

}

public void displayMenu() {

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

for (Beverage beverage : menu) {

System.out.println(beverage);

}

System.out.println();

}

public Beverage searchBeverageByName(String beverageName) {

for (Beverage beverage : menu) {

if (beverage.getName().equals(beverageName)) {

return beverage;

}

}

return null;

}

public void serveBeverage(String beverageName, String customerName) {

Beverage beverage = searchBeverageByName(beverageName);

if (beverage.getName().equals(beverageName)) {

for (Customer customer : customers) {

if (customer.getName().equals(customerName)) {

customer.orderBeverage(beverage);

break;

}

}

}

}

public void displayCustomerOrders(String customerName) {

for (Customer customer : customers) {

if (customer.getName().equals(customerName)) {

System.out.println("Orders for " + customerName + ":");

for (Beverage beverage : customer.getOrderedBeverages()) {

System.out.println(beverage);

}

System.out.println();

break;

}

}

}

}

import java.util.ArrayList;

import java.util.List;

class **Customer** {

private String name;

private List<Beverage> orderedBeverages;

public Customer(String name) {

this.name = name;

this.orderedBeverages = new ArrayList<>();

}

public String getName() {

return name;

}

public List<Beverage> getOrderedBeverages() {

return orderedBeverages;

}

public void orderBeverage(Beverage beverage) {

orderedBeverages.add(beverage);

System.out.println(name + " ordered: " + beverage.getName());

System.out.println();

}

}

public class **CoffeeShopTest** {

public static void main(String[] args) {

CoffeeShop coffeeShop = new CoffeeShop("Caffe 24");

coffeeShop.addBeverage(new Coffee("Espresso", 5.0));

coffeeShop.addBeverage(new Coffee("Latte", 3.0));

coffeeShop.addBeverage(new Coffee("Cappuccino", 4.0));

coffeeShop.addBeverage(new Tea("Green Tea", 2.0));

coffeeShop.addBeverage(new Tea("Black Tea", 2.0));

coffeeShop.addBeverage(new Beverage("Coca Cola", 1.0));

coffeeShop.addCustomer(new Customer("Aritra"));

coffeeShop.addCustomer(new Customer("Prottoy"));

coffeeShop.addCustomer(new Customer("Arman"));

coffeeShop.addCustomer(new Customer("Tamim"));

coffeeShop.addCustomer(new Customer("Niloy"));

coffeeShop.displayMenu();

coffeeShop.serveBeverage("Espresso", "Aritra");

coffeeShop.serveBeverage("Cappuccino", "Prottoy");

coffeeShop.serveBeverage("Green Tea", "Arman");

coffeeShop.serveBeverage("Coca Cola", "Tamim");

coffeeShop.displayCustomerOrders("Aritra");

coffeeShop.displayCustomerOrders("Prottoy");

coffeeShop.displayCustomerOrders("Arman");

coffeeShop.displayCustomerOrders("Tamim");

}

}

**Explanation:**

**1.** **Beverage Class:**

* Attributes:
  + name: A string representing the name of the beverage.
  + price: A double representing the price of the beverage.
* Methods:
  + Beverage(String name, double price): Constructor to initialize the name, price, and availability status of the beverage.
  + getName(): Returns the name of the restaurant.
  + getPrice(): Returns the base price
  + isAvailable(): Method to check if the beverage is available.
  + setAvailable(boolean available): Method to set the availability status of the

beverage

* + toString(): Overrides the toString() method to provide a string representation of the restaurant

**2. Coffee class (extends Beverage):**

* Inherits all attributes and methods from the Beverage class.
* Constructor: Initializes a coffee beverage with the given name, price.

**3.** . **Tea class (extends Beverage):**

* Inherits all attributes and methods from the Beverage class.
* Constructor: Initializes a tea beverage with the given name, price.

**4. Customer Class:**

* Attributes**:**
  + name: A string representing the name of the customer.
  + orderedCoffees: A list of Beverage objects representing the coffees ordered by the customer.
* Methods:
  + Customer(String name): Constructor to initialize the name of the customer and create an empty list for ordered coffees.
  + getName(): Method to retrieve the name of the customer.
  + getOrderedBeverages (): Method to retrieve the list of ordered beverages.
* orderBeverage(Beverage beverage): Method to place an order for a beverage.
  + - Adds the beverages to the list of ordered coffees if it is available.
    - Updates the availability status of the beverage items.
    - Prints a message indicating the success or failure of the order.

**5. CoffeeShop Class:**

* Attributes:
* name : A string representing the name of the coffee shop.
  + menu: A list of Beverage objects representing the menu items available at the coffee shop.
  + customers: A list of Customer objects representing the customers of the coffee shop.
* Methods:
  + CoffeeShop(String name): Constructor to initialize the name of the coffee shop and create empty lists for the menu and customers.
  + addBeverage(Beverage beverage): Method to add a coffee to the menu.
  + addCustomer(Customer customer): Method to add a customer to the list of customers.
  + displayMenu(): Method to display the menu items.
  + searchBeverageByName(String beverageName): Method to search for a coffee by its name in the menu.
  + serveBeverage (String beverageName, String customerName): Method to serve a coffee to a customer.
  + displayCustomerOrders(String customerName): Method to display the orders of a specific customer.

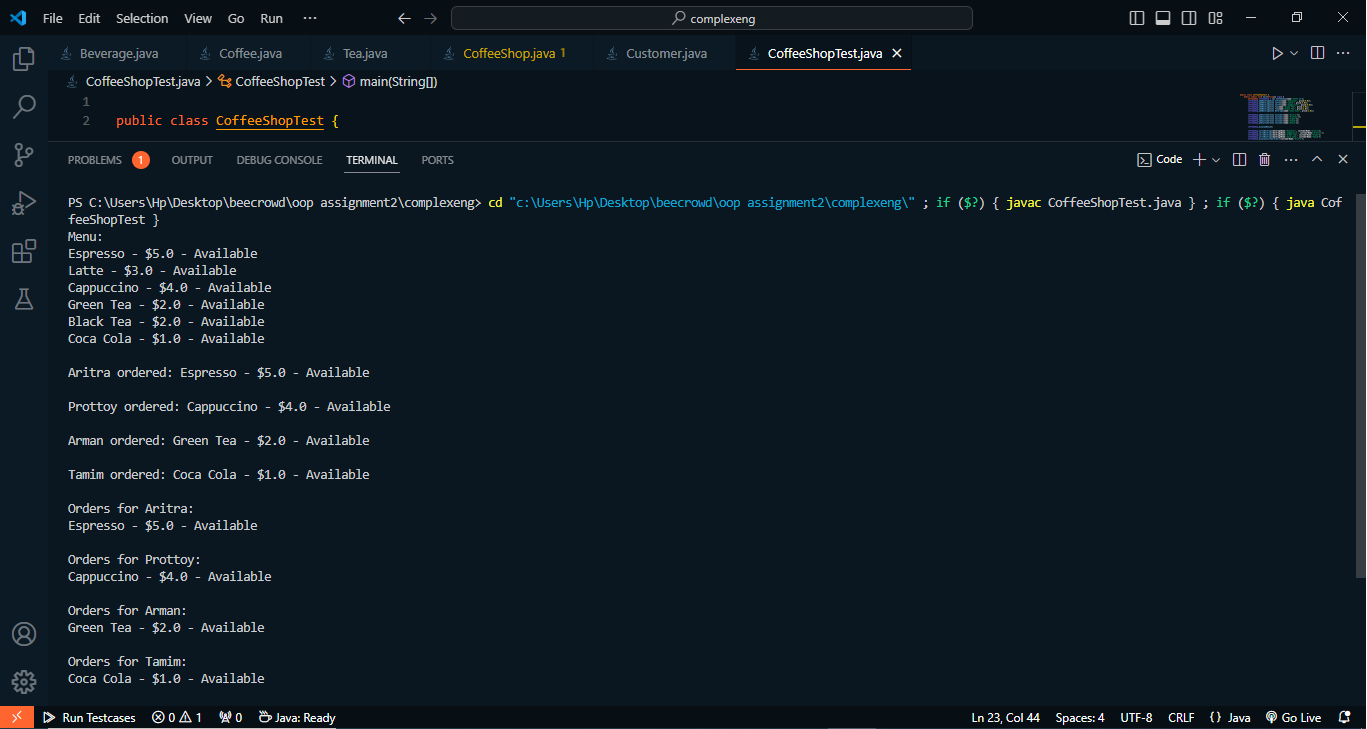
**6. CoffeeShopTest class (Main/driver class):**

* Contains the main method to test the functionality of the CoffeeShop class by adding beverage, customers, displaying the menu, serving coffees,tea, and displaying customer orders.
* **Inheritance:** Coffee and Tea classes inherit from the Beverage class. They inherit properties and methods such as name, price, getName(), and getPrice() from the Beverage class.
* **Polymorphism:** Polymorphism is evident in the CoffeeShop class, particularly in the addBeverage() and addCustomer() methods. These methods accept objects of type Beverage and Customer respectively, but they operate on them through their superclass references (Beverage for Coffee and a generic Object reference for Customer). This allows for flexibility in handling different types of beverages and customers without needing separate methods for each subtype.  
  Another example of polymorphism is seen in the searchBeverageByName() method of the CoffeeShop class. It returns a Beverage object, but it can handle any object of type Beverage due to inheritance.
* **Encapsulation:** Data encapsulation is achieved by making properties private and providing public getter and setter methods. For instance, the Beverage class encapsulates the type and available properties.

**Method Overloading :** In the Beverage class, we have a constructor public Beverage(String name, double price) which is overloaded to accept different parameters. The addBeverage() and addCustomer() methods in the CoffeeShop class can be considered as overloaded methods as they accept different types of parameters (Beverage and Customer objects, respectively).

**Method Overriding:** The toString() method is overridden in the Beverage class. The purpose of overriding toString() is to provide a custom string representation of the object. In this case, it returns a string containing the name, price, and availability status of the beverage.

**ScreenShot:**

****