Yogurt & Smoothie Shops Inc. (YSSI)

Project Description

This is a Requirements Specification Document for Yogurt & Smoothie Shops Inc. (YSSI). YSSI just celebrated the grand opening of a new brick & mortar store in Flushing, Queens, NY: Yogurt & Smoothie Shop (YSS). YSS, like the other stores in YSSI's franchise, sells yogurt & smoothies. Customers will buy yogurts & smoothies in YSS through the computer system prepared by QCSE. The system takes orders, processes payments, and generates a sales report at the end of the day.

Purpose (why the customer requires this project):

YSS requires this computer system for taking orders from customers. With it, patrons can place a complex order of yogurts and smoothies, with a variety of sizes, flavors, and toppings. The addition of cashless payment via credit card also adds to the ease of their transactions. Both these features built into the program will help YSS become a popular community staple. Lastly, management can review the sales report generated by the system.

Business Drivers/Business Model:

The software was engineered for the purpose of increasing YSSI profits. By removing verbal communication out of an order's equation, the likelihood of employee error is greatly reduced. Such mistakes, which require us to refund customers, will be a thing of the past. Similarly, the ease of a visual and kinesthetic interface (on tablets) enhances customer experience, as does the option of credit card payment—no more fiddling around for paper money and coins in a wallet. These all promote customer satisfaction, and a satisfied customer will return to YSS and spend more. Also, management can fine tune their business decisions based on the daily sales report. Because the system is intuitive, no employee training is required, so downtime for the business is minimal. Therefore, our software reduces waste and increases profits. This outweighs the upfront cost of installing & maintaining new tablets.

Scope

We will deliver the system as a fully developed cross-platform, mobile application. We will not include business tools for analytics; instead, our application only provides the raw data of sales to be interpreted. Likewise, YSSI will be responsible for changing the sizes, flavors, and toppings listed on the menu, based upon on availability. We will also not handle the network infrastructure in the brick-and-mortar store, to process payments and communicate with other devices. For example, routers, CAT5e/CAT6 ethernet cables for wired connections, and Wi-Fi 5/Wi-Fi 6 access points, are all up to YSSI to provide.

Definitions

1. A tablet, such an iPad manufactured by Apple or an Android one by Samsung, is a single board computer with a touchscreen (on which customers can click to place their orders for yogurts/smoothies, with their size/flavor/toppings). To reduce bulk cost, traditional peripherals such as the 3.5mm audio jack may be removed.
2. A database is form of electronic storage, the application's back-end, in which daily sales reports can be created, read, updated, and deleted; an example of one is Microsoft SQL Server.

Use Case Table:

https://www.usability.gov/how-to-and-tools/methods/use-cases.html

Acceptance Criteria:

Will be used to determine when the project is done.

Value	Expected Result	Expected Message (System output)
System will boot & run in 20 seconds	Pass	Cash Register ========== 1. Take Order. 2. Produce daily report 3. Exit ==========
Take an order in 5 seconds (select "1")	Pass	Choose an item from the menu ========== 1. Yogurt 2. Smoothie 3. Exit ==========
Order yogurt in 2 seconds (select "1")	Pass	You ordered a Yogurt Enter the flavor of Yogurt ---------- 1. Original 2. White Peach 3. Mango 4. Vanilla 5. Chocolate 6. Pistachio 7. Exit ----------
Order flavor of yogurt in 2 seconds.	Pass	You order flavored Yogurt Enter the size of Yogurt ---------- 1. Small Original Yogurt 2. Regular Original Yogurt 3. Large Original Yogurt ----------
Order size of yogurt in 2 seconds.	Pass	Topping for the Yogurt? 1: YES 2: NO
Check out in 3 seconds.	Pass	Thank you, come again
Enter Credit Card Number 1111111111111111 (16 digits)	Pass	Enter the expiration month: ==========
Enter Credit Card Expiration month 12 (2 digits)	Pass	Enter the cvv: ==========
Enter Credit Card CVV 123 (3 digits long)	Pass	$ paid with Credit Card ********** Order is ready. **********

Produce daily sales report in 5 seconds	Pass	Net Sales : $ Tax Owed : $ Total Sales + Tax : $

Assumptions and Constraints

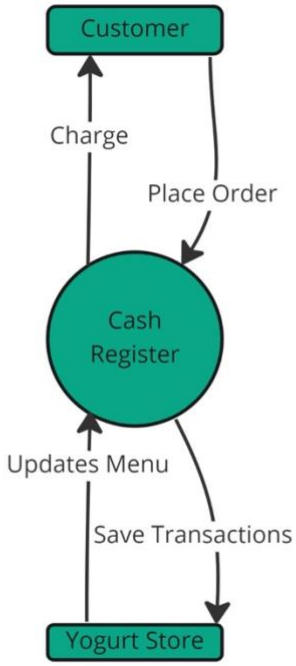
Management and employees of YSS will be available to review and test the options and flow of the order system, to fine tune it according to their experience. Use of Java code, using Oracle's JDK & JRE.

Platform Requirements Specification

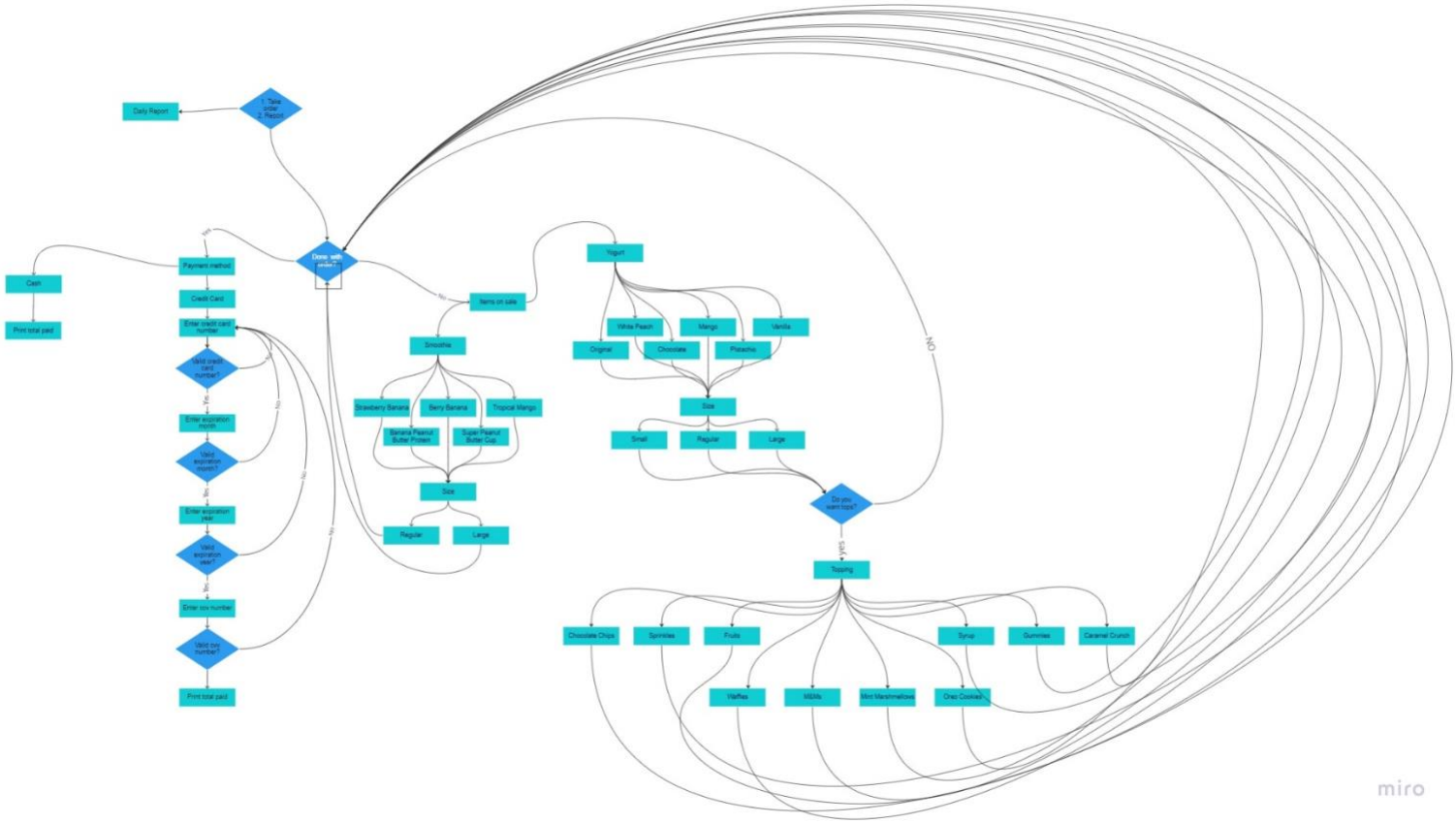
iOS/iPad OS 14.0+ (for iPads/iPhones), Android 10.0+ (for tablets/Android smart phones), or any modern browser, such as Microsoft Edge, Google Chrome, or Mozilla Firefox.
4GB of RAM: DDR4-2666 or higher
Quad-Core CPU (x86-64 or ARM): 1.5GHz or higher

Context Diagram and the P

Context Diagram:



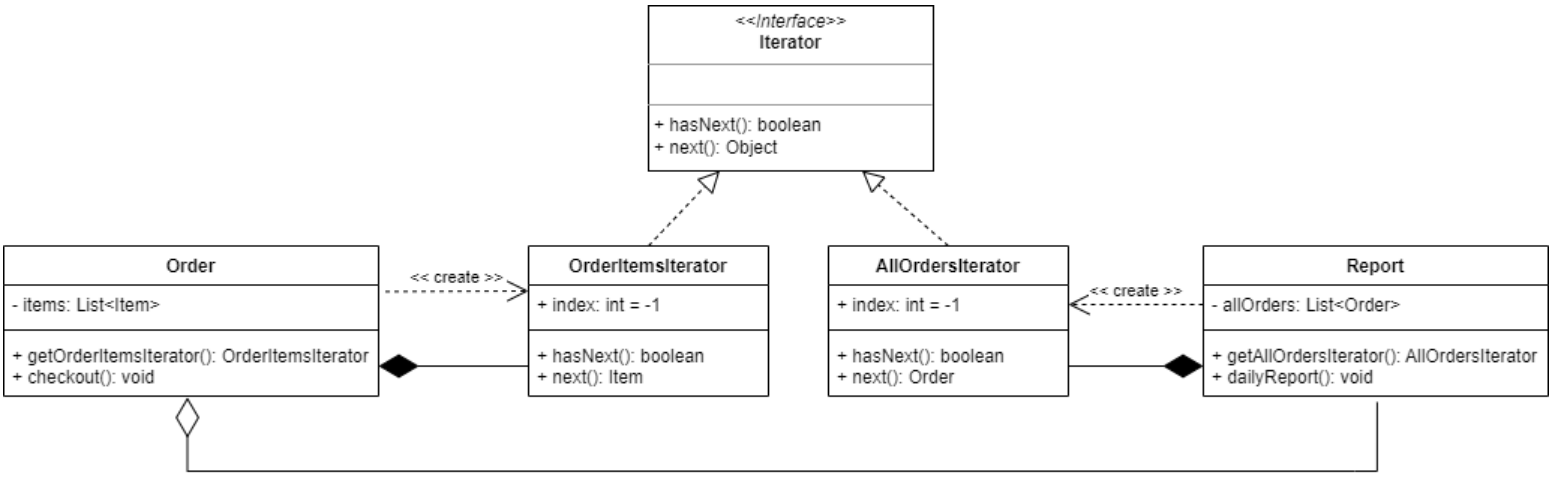
Process Flow Diagram:



Design Patterns

Design Pattern Name: Iterator Pattern

UML diagram for design pattern:



Java Classes that implement design pattern:
Implementation of the Iterator Pattern inside the Order and Report concrete classes. Each of these classes contains: 1) a private class which implements my Iterator interface, 2) a public method which returns that private class, and 3) a method which utilizes an instance of that iterator. Removal of code that is not relevant to my Iterator Pattern implementation, for this document.

Iterator.java

```
public interface Iterator {
	public boolean hasNext();
	public Object next();
}
```

Order.java

```
import java.util.ArrayList;
import java.util.List;

public class Order {
	private List<Item> items;

	public Order() {
		items = new ArrayList<>();
	}

	public class OrderItemsIterator implements Iterator {
		public int index;

		public OrderItemsIterator() {
			index = -1; // This indicates our place in the list. Initialized to -1 so the 0th element is
returned by next() on its first call.
		}

		public boolean hasNext() {
			return index < items.size() - 1;
		}
```

```
		public Item next() {
			index++;
			return items.get(index);
		}
	}

	public OrderItemsIterator getOrderItemsIterator() {
		return new OrderItemsIterator();
	}

	public void checkout() {
		OrderItemsIterator orderItemsIterator = new OrderItemsIterator();
		while (orderItemsIterator.hasNext()) {
			orderItemsIterator.next().printItem();
		}
	}
}
```

Report.java

```
import java.util.ArrayList;
import java.util.List;

public class Report {
	private List<Order> allOrders;

	public Report() {
		allOrders = new ArrayList<>();
	}

	public class AllOrdersIterator implements Iterator {
		public int index;

		public AllOrdersIterator() {
			index = -1; // This indicates our place in the list. Initialized to -1 so the 0th element is
returned by next() on its first call.
		}

		public boolean hasNext() {
			return index < allOrders.size() - 1;
		}

		public Order next() {
			index++;
			return allOrders.get(index);
		}
	}

	public AllOrdersIterator getAllOrdersIterator() {
		return new AllOrdersIterator();
	}

	void dailyReport() {
		double netSales = 0.0;
		double taxOwed = 0.0;
```

```
		AllOrdersIterator allOrdersIterator = new AllOrdersIterator();
		while (allOrdersIterator.hasNext()) {
			Order order = allOrdersIterator.next();
			netSales += order.getSubtotal();
			taxOwed += order.getSalesTax();
		}
	}
}
```

Code for two-unit tests and the paragraph for 1 component test
Unit Tests:

OrderAndReportIteratorTest.java

```
import Order.OrderItemsIterator;
import Report.AllOrdersIterator;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class OrderAndReportIteratorTest {
	private Order orderSmallMangoYogurt() {
		Order order = new Order();
		order.addItem(new Item("Mango Yogurt", 4.95, "Small"));
		return order;
	}

	@Test
	public void testOrderItemIterator() {
		Order order = orderSmallMangoYogurt();
		OrderItemsIterator orderItemsIterator = order.getOrderItemsIterator();
		while(orderItemsIterator.hasNext()) {
			Item orderItem = orderItemsIterator.next();
			assertTrue(orderItem.getName().equals("Mango Yogurt")
					&& orderItem.getSize().equals("Small"));
		}
	}

	@Test
	public void testReportAllOrdersIterator() {
		Report report = new Report();
		double netSubtotal = 0.0;
		report.addOrder(orderSmallMangoYogurt());
		report.addOrder(orderSmallMangoYogurt());
		AllOrdersIterator allOrdersIterator = report.getAllOrdersIterator();
		while(allOrdersIterator.hasNext()) {
			netSubtotal += allOrdersIterator.next().getSubtotal();
		}
		assertTrue(netSubtotal == 9.9);
	}
}
```

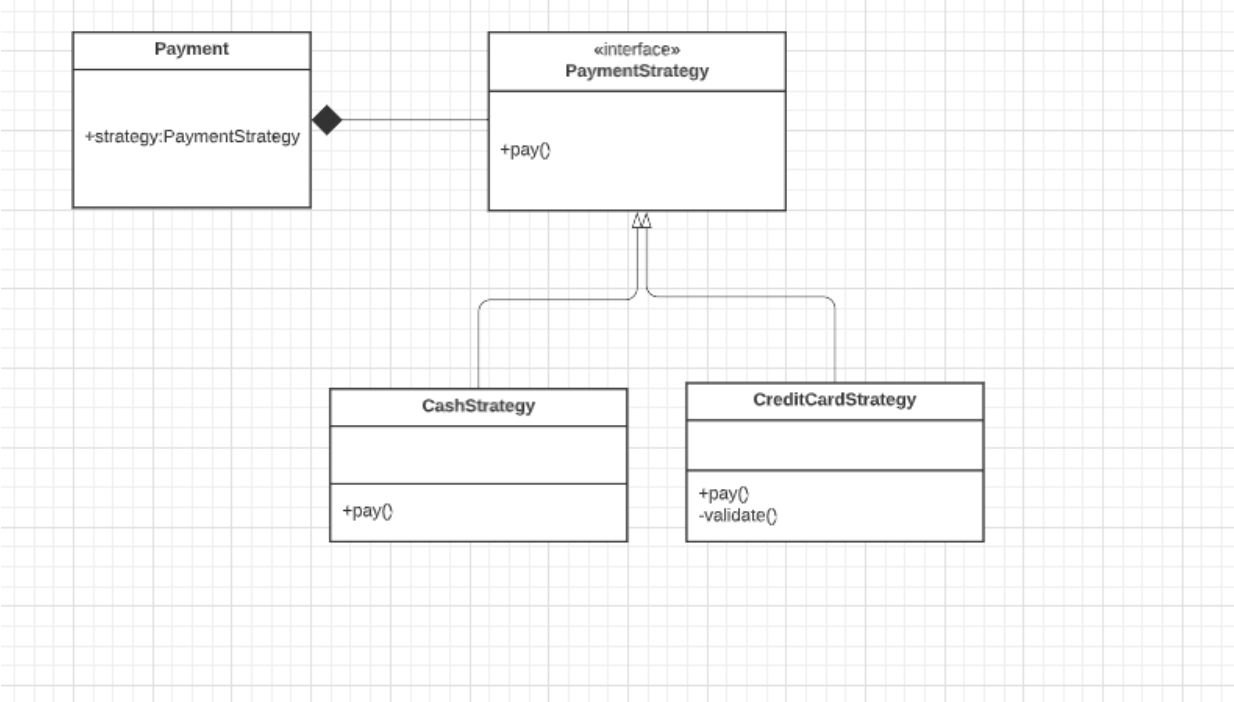
Component Test: I am a customer making an order at the yogurt & smoothie shop. I order two items: one small mango yogurt and one regular strawberry banana smoothie. When I check out, the name, size, and price of both of these items which I ordered will print out onto the screen before I pay:

Small Mango Yogurt $4.95

Strawberry Banana Smoothie $6.95

Design Pattern Name: Strategy Pattern

UML diagram for design pattern:



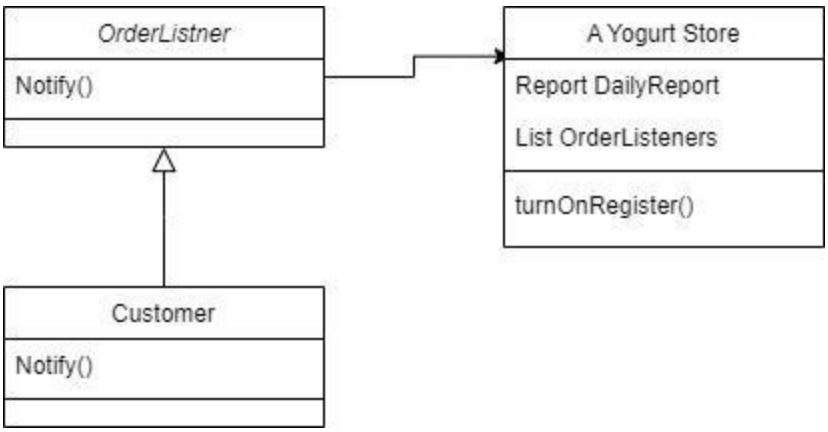
Java Classes that implement design pattern

```
public class CashStrategy implements PaymentStrategy {
	public boolean pay(double total) {
	    System.out.println(String.format("$%.2f paid with Cash", total));
	    return true;
	}
}

public class StrategyFactory {
	private PaymentStrategy paymentStrategy;

	public StrategyFactory(int paymentOption) {
		if (paymentOption == 1) {
			this.paymentStrategy = new CashStrategy();
		} else if (paymentOption == 2) {
			this.paymentStrategy = new CreditCardStrategy();
		}
	}

	public PaymentStrategy paymentStrategy() {
		return this.paymentStrategy;
	}
}
```

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Payment {

	public void processPayment(double total) {
		System.out.println("Select your payment method");
		System.out.println("============================");
		System.out.println("	1. $ Cash $   ");
		System.out.println("	2. Credit Card   ");
		System.out.println("============================");

		try {
			BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

			int paymentOption = Integer.parseInt(br.readLine());

			StrategyFactory factory = new StrategyFactory(paymentOption);

			boolean paid = factory.paymentStrategy().pay(total);
			while (!paid) {
				System.out.println("payment failed. Please try again.");
				System.out.println("============================");
				paid = factory.paymentStrategy().pay(total);
			}

		} catch (Exception e) {

		}

	}
}

public interface PaymentStrategy {
	public boolean pay(double total);
}

public class StrategyFactory {
	private PaymentStrategy paymentStrategy;

	public StrategyFactory(int paymentOption) {
		if (paymentOption == 1) {
			this.paymentStrategy = new CashStrategy();
		} else if (paymentOption == 2) {
			this.paymentStrategy = new CreditCardStrategy();
		}
	}

	public PaymentStrategy paymentStrategy() {
		return this.paymentStrategy;
	}
}
```

Design Pattern Name: Observer Pattern

UML diagram for design pattern



Java Classes that implement design

```
public interface OrderListener {
   void notify(int orderReady);
}


public class Customer implements OrderListener{
   @Override
   public void notify(int orderReady) {
      System.out.println("\n*****************************************\n"
            +"\nOrder " + orderReady+ " is ready.\n"+
            "\n*****************************************\n");
   }
}


import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

public class AYogurtStore {

   private Report dailyReport;
   AYogurtStore(){
      dailyReport = new Report();
   }
   void turnOnRegister() throws IOException {

      OrderBuilder orderBuilder = new OrderBuilder();
      Order order;
      Payment payment = new Payment();
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

      int choice = 0;
```

```
        while(choice < 3){

            System.out.println("       Cash Register       ");
            System.out.println("============================");
            System.out.println("       1. Take Order.   ");
            System.out.println("       2. Produce daily report   ");
            System.out.println("       3. Exit           ");
            System.out.println("============================");
            choice = Integer.parseInt(br.readLine());
            if(choice == 1){
                order = orderBuilder.prepareOrder();
                order.checkout();
                dailyReport.addOrder(order);
                var grandTotal = order.getGrandTotal();
                if (grandTotal > 0) {
                    payment.processPayment(grandTotal);
                }

                notify(order.getOrderNumber());
            }
            if(choice ==2){
                dailyReport.dailyReport();
            }
            if(choice<1){ System.out.println("invalid choice\n");}
        }

        System.out.println("Register Shutting Down.....");

    }
    private List<OrderListener> orderListeners = new ArrayList<>();

    public void addlistener(OrderListener orderListener){
        orderListeners.add(orderListener);
    }
    public void removeListener(OrderListener orderListener){
        orderListeners.remove(orderListener);
    }
    public void notify(int something){
        for(OrderListener l: orderListeners){
            l.notify(something);
        }
    }
}
```

Code for two unit tests and the paragraph for 1 component test
Unit Tests:

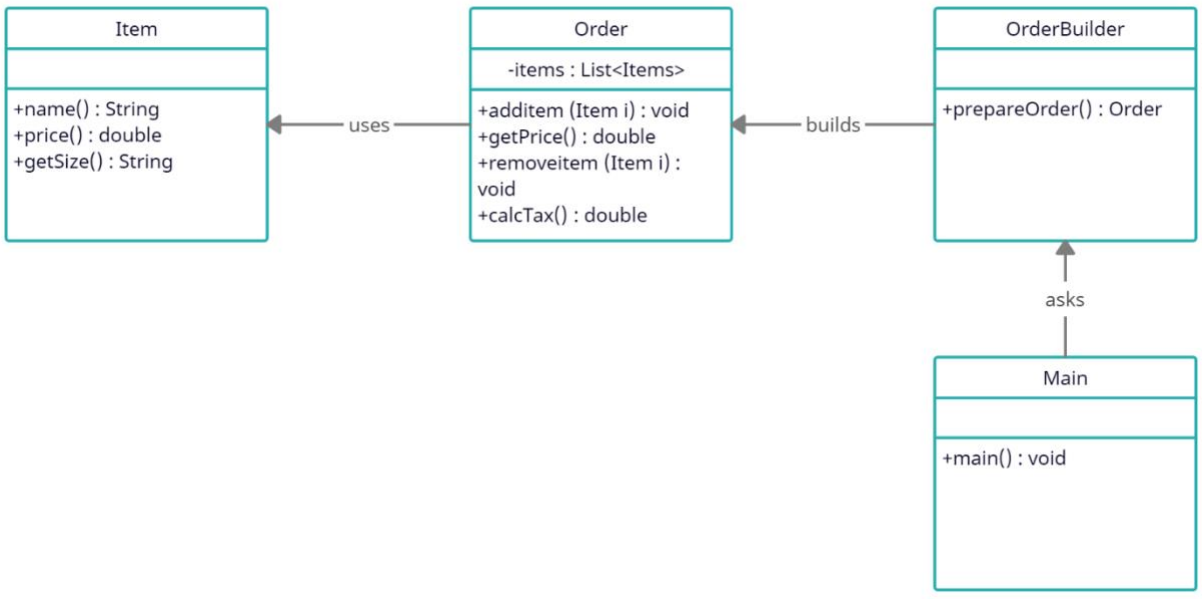
```
import org.junit.jupiter.api.Test;

import java.io.IOException;

import static org.junit.jupiter.api.Assertions.*;

public class ObserverTest{
    @Test
    void testOrder1() throws IOException {
```

```
        AYogurtStore store = new AYogurtStore();
        store.addlistener(new Customer());
        Order o = new Order();

        store.notify(o.getOrderNumber());
        assertTrue(o.getOrderNumber()==1);
    }

    @Test
    void testOrderIncreases(){
        AYogurtStore store = new AYogurtStore();
        store.addlistener(new Customer());
        Order o = new Order();
        store.notify(o.getOrderNumber());
        Order o1 = new Order();
        store.notify(o1.getOrderNumber());
        Order o2 = new Order();
        store.notify(o2.getOrderNumber());

        assertTrue(o2.getOrderNumber()==3);
    }
}
```

Component Test: The user is an employee of a yogurt store using a register. A customer comes in and wants to order something, we sell yogurt and smoothies. Choose the Flavor then the size, repeat if they want to order more items. If done, the total for order is calculated and they may choose to pay with cash or card. The customer is notified that the order is ready. At the end of day the employee may look at the daily sales report which will display the net sales made.

Design Pattern Name: Builder Pattern

UML diagram for design pattern



Java Classes that implement design pattern

```
import java.io.IOException;
```

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class OrderBuilder {

    Toppings toppings = new Toppings();
    public Order prepareOrder() throws IOException {
        Order itemsOrder = new Order();
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        MenuInterface yogurtMenu = new YogurtMenu();
        MenuInterface smoothieMenu = new SmoothieMenu();
        MenuInterface toppingsMenu = new ToppingsMenu();


        int yogurtandsmoothiechoice = 0;

        while (yogurtandsmoothiechoice != 3) {
            yogurtMenu.getMenu();
            yogurtandsmoothiechoice = Integer.parseInt(br.readLine());
            switch (yogurtandsmoothiechoice) {
                case 1: {
                    yogurtMenu.getFlavour();
                    int yogurtchoice = Integer.parseInt(br.readLine());
                    switch (yogurtchoice) {
                        case 1: {
                            yogurtMenu.getSize("Original");
                            int originalyogurtsize = Integer.parseInt(br.readLine());
                            switch (originalyogurtsize) {
                                case 1:
                                    itemsOrder.addItem(new Item("original yogurt", 4.95, "Small"));

                                    toppingsMenu.getMenu();
                                    toppings.prepareTopping(itemsOrder);
                                    break;
                                case 2:
                                    itemsOrder.addItem(new Item("original yogurt", 6.95, "Regular"));
                                    toppingsMenu.getMenu();
                                    toppings.prepareTopping(itemsOrder);
                                    break;
                                case 3:
                                    itemsOrder.addItem(new Item("original yogurt", 7.95, "Large"));
                                    toppingsMenu.getMenu();
                                    toppings.prepareTopping(itemsOrder);
                                    break;
                            }
                            break;
                        }


                        case 2: {
                            yogurtMenu.getSize("White Peach");
                            int whitepeachyogurtsize = Integer.parseInt(br.readLine());
                            switch (whitepeachyogurtsize) {
                                case 1:
                                    itemsOrder.addItem(new Item("White Peach yogurt", 4.95, "small"));
                                    toppingsMenu.getMenu();
```

```
            toppings.prepareTopping(itemsOrder);
            break;
        case 2:
            itemsOrder.addItem(new Item("White Peach yogurt", 6.95, "Regular"));
            toppingsMenu.getMenu();
            toppings.prepareTopping(itemsOrder);
            break;
        case 3:
            itemsOrder.addItem(new Item("White Peach yogurt", 7.95, "Large"));
            toppingsMenu.getMenu();
            toppings.prepareTopping(itemsOrder);
            break;
    }
    break;
}
case 3: {
    yogurtMenu.getSize("Mango");
    int mangoyogurtsize = Integer.parseInt(br.readLine());
    switch (mangoyogurtsize) {
        case 1:
            itemsOrder.addItem(new Item("Mango yogurt", 4.95, "small"));
            toppingsMenu.getMenu();
            toppings.prepareTopping(itemsOrder);
            break;
        case 2:
            itemsOrder.addItem(new Item("Mango yogurt", 6.95, "regular"));
            toppingsMenu.getMenu();
            toppings.prepareTopping(itemsOrder);
            break;
        case 3:
            itemsOrder.addItem(new Item("Mango yogurt", 7.95, "large"));
            toppingsMenu.getMenu();
            toppings.prepareTopping(itemsOrder);
            break;
    }
    break;
}
case 4: {
    yogurtMenu.getSize("Vanilla");
    int vanillayogurtsize = Integer.parseInt(br.readLine());
    switch (vanillayogurtsize) {
        case 1:
            itemsOrder.addItem(new Item("Vanilla yogurt", 4.95, "small"));
            toppingsMenu.getMenu();
            toppings.prepareTopping(itemsOrder);
            break;
        case 2:
            itemsOrder.addItem(new Item("Vanilla yogurt", 6.95, "regular"));
            toppingsMenu.getMenu();
            toppings.prepareTopping(itemsOrder);
            break;
        case 3:
            itemsOrder.addItem(new Item("Vanilla yogurt", 7.95, "large"));
            toppingsMenu.getMenu();
            toppings.prepareTopping(itemsOrder);
            break;
```

```
            }
            break;
        }
        case 5: {
            yogurtMenu.getSize("Chocolate");
            int chocolateyogurtsize = Integer.parseInt(br.readLine());
            switch (chocolateyogurtsize) {
                case 1:
                    itemsOrder.addItem(new Item("Chocolate yogurt", 4.95, "small"));
                    toppingsMenu.getMenu();
                    toppings.prepareTopping(itemsOrder);
                    break;
                case 2:
                    itemsOrder.addItem(new Item("Chocolate yogurt", 6.95, "large"));
                    toppingsMenu.getMenu();
                    toppings.prepareTopping(itemsOrder);
                    break;
                case 3:
                    itemsOrder.addItem(new Item("Chocolate yogurt", 4.95, "large"));
                    toppingsMenu.getMenu();
                    toppings.prepareTopping(itemsOrder);
                    break;
            }
            break;
        }
        case 6: {
            yogurtMenu.getSize("Pistachio");
            int Pistachioyogurtsize = Integer.parseInt(br.readLine());
            switch (Pistachioyogurtsize) {
                case 1:
                    itemsOrder.addItem(new Item("Pistachio yogurt", 4.95, "small"));
                    toppingsMenu.getMenu();
                    toppings.prepareTopping(itemsOrder);
                    break;
                case 2:
                    itemsOrder.addItem(new Item("Pistachio yogurt", 6.95, "regular"));
                    toppingsMenu.getMenu();
                    toppings.prepareTopping(itemsOrder);
                    break;
                case 3:
                    itemsOrder.addItem(new Item("Pistachio yogurt", 7.95, "large"));
                    toppingsMenu.getMenu();
                    toppings.prepareTopping(itemsOrder);
                    break;
            }
            break;
        }
        default: System.out.println("testing default");
    }
    break;
}
case 2: {
    smoothieMenu.getFlavour();
    int smoothie = Integer.parseInt(br.readLine());
    switch (smoothie) {
        case 1: {
```

```
    smoothieMenu.getSize("Strawberry Banana");
    int strawberrybananasize = Integer.parseInt(br.readLine());
    switch (strawberrybananasize) {
        case 1:
            itemsOrder.addItem(new Item("Strawberry Banana Smoothie", 6.75, "regular"));
            break;

        case 2:
            itemsOrder.addItem(new Item("Strawberry Banana Smoothie", 7.75, "large"));
            break;

    }
    break;
}
case 2: {
    smoothieMenu.getSize("Berry Banana");
    int berrybananasize = Integer.parseInt(br.readLine());
    switch (berrybananasize) {
        case 1:
            itemsOrder.addItem(new Item("Berry Banana Smoothie", 6.75, "regular"));
            break;

        case 2:
            itemsOrder.addItem(new Item("Berry Banana Smoothie", 7.75, "large"));
            break;

    }

    break;
}
case 3: {
    smoothieMenu.getSize("Topical Mango");
    int tropicalmangosize = Integer.parseInt(br.readLine());
    switch (tropicalmangosize) {
        case 1:
            itemsOrder.addItem(new Item("Tropical Mango Smoothie", 6.75, "regular"));
            break;

        case 2:
            itemsOrder.addItem(new Item("Tropical Mango Smoothie", 7.75, "large"));
            break;

    }
    break;
}
case 4: {
    smoothieMenu.getSize("Banana Peanut Butter Protein");
    int bananapeanutbuttersize = Integer.parseInt(br.readLine());
    switch (bananapeanutbuttersize) {
        case 1:
            itemsOrder.addItem(new Item("Banana Peanut Butter", 6.95, "regular"));
            break;

        case 2:
            itemsOrder.addItem(new Item("Banana Peanut Butter", 7.95, "large"));
            break;
```

```
                    }
                    break;
                }
                case 5: {
                    smoothieMenu.getSize("Super Peanut Butter Cup");
                    int superpeanutbutter = Integer.parseInt(br.readLine());
                    switch (superpeanutbutter) {
                        case 1:
                            itemsOrder.addItem(new Item("Super Peanut Butter Cup", 6.95, "regular"));
                            break;

                        case 2:
                            itemsOrder.addItem(new Item("Super Peanut Butter Cup", 7.95, "large"));
                            break;

                    }
                    break;
                }
            }
        }
        case 3:
          System.out.println("\nThank you, come again\n");
          break;
        default:
          System.out.println("You have not chosen an item from the list");
          break;

      }


    }return itemsOrder;
  }
}
```

Code for two unit tests and the paragraph for 1 component test
Unit Tests:

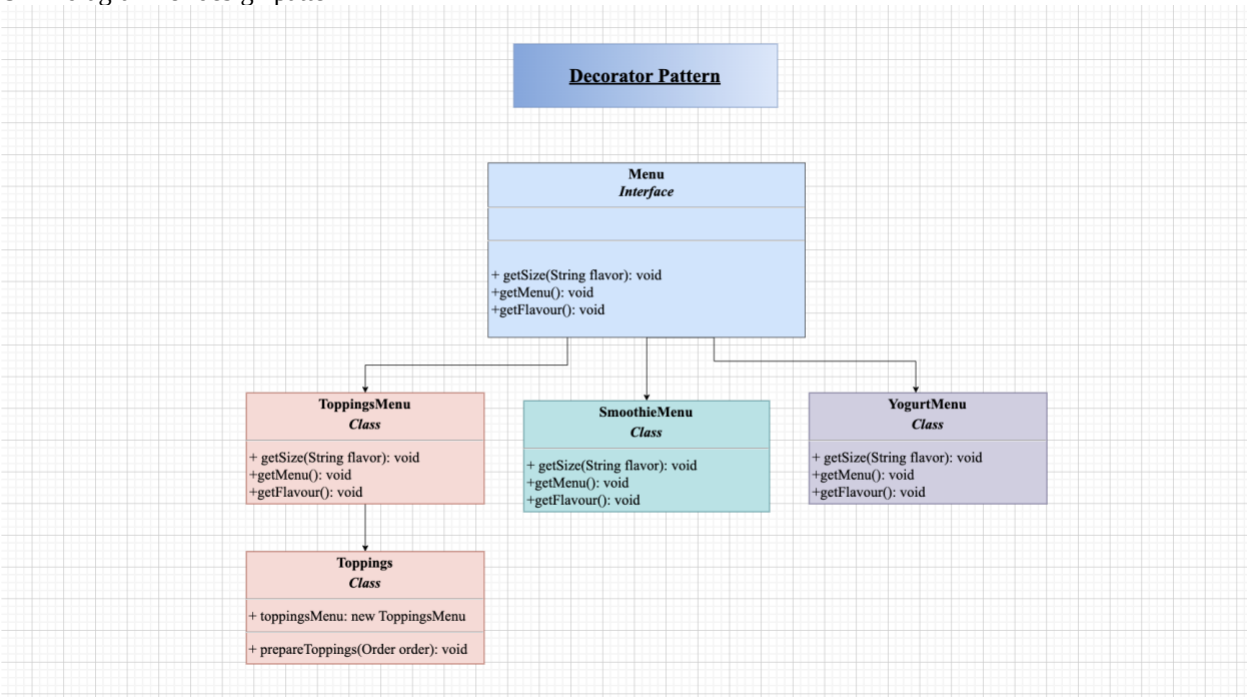
```
import java.io.BufferedReader;
import java.io.InputStreamReader;

import static org.junit.jupiter.api.Assertions.*;

class OrderBuilderTest {

  @org.junit.jupiter.api.Test
  void prepareYogurt() {
    OrderedMenus itemsOrder=new OrderedMenus();
    BufferedReader br =new BufferedReader(new InputStreamReader(System.in));
    int InputStreamReader = 0;
    assertFalse(InputStreamReader==1);
    }
  }

import java.io.BufferedReader;
import java.io.InputStreamReader;

import static org.junit.jupiter.api.Assertions.*;
```

```
class OrderBuilderTest {

    @org.junit.jupiter.api.Test
    void prepareYogurt() {
        OrderedMenus itemsOrder=new OrderedMenus();
        BufferedReader br =new BufferedReader(new InputStreamReader(System.in));
        int InputStreamReader = 3;
        assertTrue(InputStreamReader==3);
        }
    }
```

Component Test: We are going through the possible order. We first check to see if the customer orders a yogurt or a smoothie. If they exit, they are proceeding to the checkout page. If they choose a yogurt, it will display the options. After choosing a flavor, it will let you choose a size and toppings. After choosing one, it will return to the main menu page. You are given the choice to choose another option or checkout. If you do choose a smoothie, you will be given an option to choose a size. Once a size is chosen, it will return to the main menu page. If you accidentally press yogurt and want to go back to the main menu, you just have to exit that page to go to the main menu.

Design Pattern Name: Decorator Pattern

UML diagram for design pattern



Java Classes that implement your design pattern

```
public class YogurtMenu implements MenuInterface {
    @Override
    public void getMenu() {
        System.out.println(" Choose an item from the menu ");
        System.out.println("============================");
        System.out.println("       1. Yogurt   ");
        System.out.println("       2. Smoothie   ");
        System.out.println("       3. Exit          ");
        System.out.println("============================");
```

```
    }

    @Override
    public void getSize(String flavour) {
        System.out.println("You ordered " + flavour + " Yogurt");
        System.out.println("Enter the size of Yogurt");
        System.out.println("-----------------------------------");
        System.out.println("    1." + " Small" + " " + flavour + " Yogurt ");
        System.out.println("    2." + " Regular" + " " + flavour + " Yogurt ");
        System.out.println("    3." + " Large " + "" + flavour + " Yogurt ");
        System.out.println("-----------------------------------");
    }

    @Override
    public void getFlavour() {
        System.out.println("You ordered a Yogurt");
        System.out.println("\n\n");
        System.out.println(" Enter the flavor of Yogurt ");
        System.out.println("-------------------------------");
        System.out.println("        1. Original        ");
        System.out.println("        2. White Peach        ");
        System.out.println("        3. Mango        ");
        System.out.println("        4. Vanilla        ");
        System.out.println("        5. Chocolate        ");
        System.out.println("        6. Pistachio        ");
        System.out.println("        7. Exit           ");
        System.out.println("-------------------------------");
    }


}

public class SmoothieMenu implements MenuInterface{
    @Override
    public void getMenu() {

    }

    @Override
    public void getSize(String flavour) {
        System.out.println("You ordered " + flavour);
        System.out.println("Enter the size of Smoothie");
        System.out.println("-----------------------------------");
        System.out.println("    1." + " Regular" + " " + flavour);
        System.out.println("    2." + " Large" + " " + flavour);
        System.out.println("-----------------------------------");
    }

    @Override
    public void getFlavour() {
        System.out.println(" Smoothie ");
        System.out.println("============================");
        System.out.println("        1. Strawberry Banana            ");
        System.out.println("        2. Berry Banana            ");
        System.out.println("        3. Tropical Mango            ");
        System.out.println("        4. Banana Peanut Butter Protein            ");
```

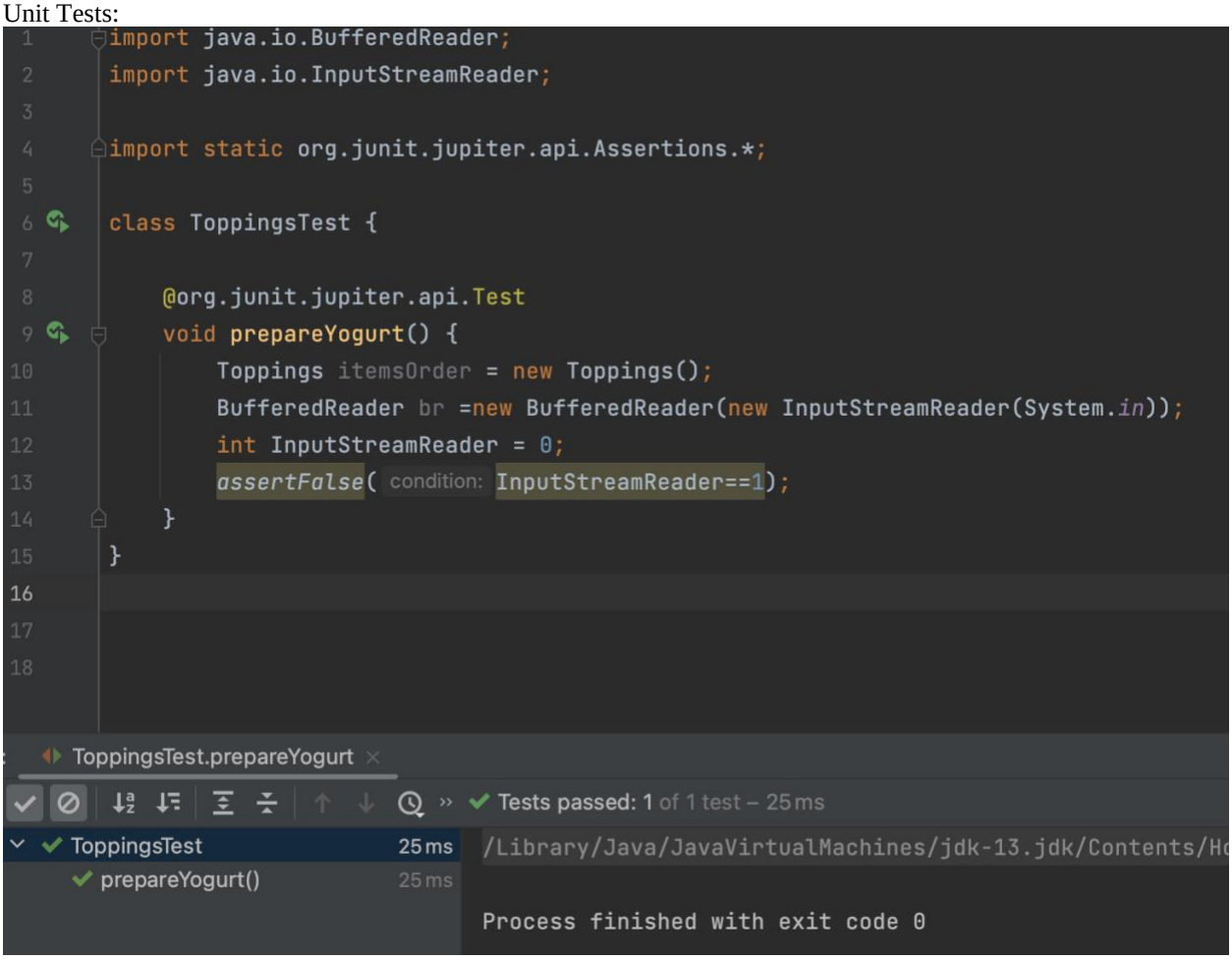
```
        System.out.println("       5. Super Peanut Butter Cup            ");
        System.out.println("       6. Exit            ");
        System.out.println("============================");
    }
}

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Toppings {
    MenuInterface toppingsMenu = new ToppingsMenu();

    public void prepareTopping(Order order) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        int choose = 0;
        while(choose!= 2){
            choose =  Integer.parseInt(br.readLine());
            switch (choose){
                case 1:
                    toppingsMenu.getFlavour();
                    int chooseToppingsFlavour = Integer.parseInt(br.readLine());
                    switch (chooseToppingsFlavour){
                        case 1:
                            order.addItem(new Item("Chocolate Chip", 1.95, ""));
                            break;
                        case 2:
                            order.addItem(new Item("Sprinkles", 1.7, ""));
                            break;
                        case 3:
                            order.addItem(new Item("Fresh Fruits", 1.3, ""));
                            break;
                        case 4:
                            order.addItem(new Item("Syrup Topping", 1.3, ""));
                            break;
                        case 5:
                            order.addItem(new Item("Gummies", 1.3, ""));
                            break;
                        case 6:
                            order.addItem(new Item("Caramel Crunch", 1.3, ""));
                            break;
                        case 7:
                            order.addItem(new Item("Waffles", 1.3, ""));
                            break;
                        case 8:
                            order.addItem(new Item("M&Ms", 1.3, ""));
                            break;
                        case 9:
                            order.addItem(new Item("Mini Marshmallows", 1.3, ""));
                            break;
                        case 10:
                            order.addItem(new Item("Oreo Cookies", 1.3, ""));
                            break;
                        default:
                            System.out.println("Main Menu:");
                    }
```

```
                break;
            }
            break;
        }

    }
}

public class ToppingsMenu implements MenuInterface{
    @Override
    public void getMenu() {
        System.out.println(" Topping for the Yogurt?");
        System.out.println("       1: YES");
        System.out.println("       2: NO");
    }

    @Override
    public void getSize(String flavour) {

    }

    @Override
    public void getFlavour() {
        System.out.println(" Toppings: ");
        System.out.println("============================");
        System.out.println("       1. Chocolate Chip");
        System.out.println("       2. Sprinkles");
        System.out.println("       3. Fresh Fruits");
        System.out.println("       4. Syrup Topping");
        System.out.println("       5. Gummies");
        System.out.println("       6. Caramel Crunch");
        System.out.println("       7. Waffles");
        System.out.println("       8. M&Ms");
        System.out.println("       9. Mini Marshmallows");
        System.out.println("       10. Oreo Cookies");
        System.out.println("       11. Return to the Main Menu");
        System.out.println("============================");
    }
}

import java.io.BufferedReader;
import java.io.InputStreamReader;

import static org.junit.jupiter.api.Assertions.*;

class ToppingsTest {

    @org.junit.jupiter.api.Test
    void prepareYogurt() {
        Toppings itemsOrder = new Toppings();
        BufferedReader br =new BufferedReader(new InputStreamReader(System.in));
        int InputStreamReader = 0;
        assertFalse(InputStreamReader==1);
    }
}
```

```
import java.io.BufferedReader;
      import java.io.InputStreamReader;

      import static org.junit.jupiter.api.Assertions.*;

class ToppingsTest {

   @org.junit.jupiter.api.Test
   void prepareYogurt() {
      YogurtMenu itemsOrder = new YogurtMenu();
      BufferedReader br =new BufferedReader(new InputStreamReader(System.in));
      int InputStreamReader = 2;
      assertFalse(InputStreamReader==1);
   }
}
```

Code for two-unit tests and the paragraph for 1 component test

Unit Tests:



```java
import java.io.BufferedReader;
        import java.io.InputStreamReader;

        import static org.junit.jupiter.api.Assertions.*;

class ToppingsTest {

    @org.junit.jupiter.api.Test
    void prepareYogurt() {
        YogurtMenu itemsOrder = new YogurtMenu();
        BufferedReader br =new BufferedReader(new InputStreamReader(System.in));
        int InputStreamReader = 2;
        assertFalse( condition: InputStreamReader==1);
    }
}
```

ToppingsTest.prepareYogurt

Tests passed: 1 of 1 test – 23 ms

ToppingsTest 23 ms

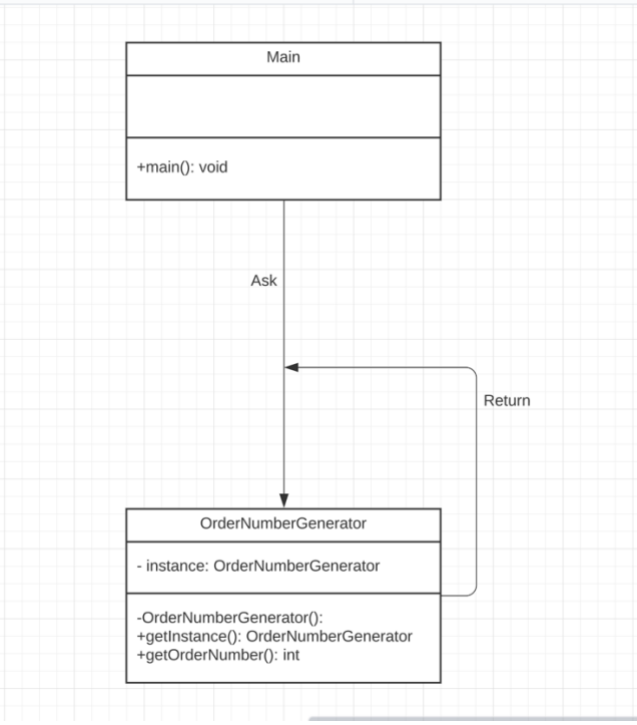
prepareYogurt() 23 ms

Component Test: A user can get his favorite flavor of frozen yogurt with different toppings or if he's in the mood of getting a yummy smoothie, then our YogurtShop is the place to go in summer. Our shop serves a variety of frozen yogurt flavors and smoothies which he or she can enjoy with our wide range of toppings! So for instance, a small cup of mango yogurt with M&Ms will charge the user $6.75 in total.

```
Mango yogurt     $4.95
M&Ms     $1.3
Your order number is: 1
subtotal   :       $ 6.25
tax        :       $ 0.50
total      :       $ 6.75
```

Design Pattern Name: Singleton Pattern

UML diagram for design pattern:



Java Classes that implement your design pattern:

```
class Main {
   public static void main(String[] args) {
      int orderNum;
      OrderNumberGenerator object = OrderNumberGenerator.getInstance();
      orderNum = object.getOrderNumber();
      System.out.println("Your Order Number is: " + orderNum);
   }
}


public class OrderNumberGenerator {
   private static OrderNumberGenerator instance = new OrderNumberGenerator();
   private OrderNumberGenerator(){}
   public static OrderNumberGenerator getInstance(){
      return instance;
   }
   int orderNumber = 0;
   public int getOrderNumber(){
      orderNumber++;
      return orderNumber;
   }
}
```

```
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class OrderNumberGeneratorTest {

    @Test
    void getInstance() {
        OrderNumberGenerator orderNumberGenerator = OrderNumberGenerator.getInstance();
        orderNumberGenerator.getOrderNumber();
        OrderNumberGenerator orderNumberGenerator1 = OrderNumberGenerator.getInstance();
        orderNumberGenerator1.getOrderNumber();
        OrderNumberGenerator orderNumberGenerator2 = OrderNumberGenerator.getInstance();
        int orderNumber = orderNumberGenerator2.getOrderNumber();
        assertTrue(orderNumber==3);
    }

    @Test
    void getOrderNumber() {
        OrderNumberGenerator orderNumberGenerator = OrderNumberGenerator.getInstance();
        orderNumberGenerator.getOrderNumber();
        OrderNumberGenerator orderNumberGenerator1 = OrderNumberGenerator.getInstance();
        int orderNumber = orderNumberGenerator1.getOrderNumber();
        assertTrue(orderNumber==2);
    }
}
```

Code for two-unit tests and the paragraph for 1 component test
Unit Tests:

Component Test: I am a customer of AYogurtStore and I ordered a small mango yogurt and paid $7.51 with cash, my order number is number 1. The customer after me ordered a large berry banana smoothie paid $7.75 with cash and his order number will be number 2.