**Homework 6**

**Github:** <https://github.com/Dubs2310/CS5800-Homework-6.git>

**State & Chain of Responsibility**

* Snack.java

public class Snack {  
 private String name;  
 private int quantity;  
 private double price;  
  
 public Snack(String name, int quantity, double price) {  
 this.name = name;  
 this.quantity = quantity;  
 this.price = price;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getQuantity() {  
 return quantity;  
 }  
  
 public double getPrice() {  
 return price;  
 }  
  
 public void decrementQuantity() {  
 this.quantity -= 1;  
 }  
}

* StateOfVendingMachine.java

public interface StateOfVendingMachine {  
 void selectSnack(VendingMachine machine, String snackName);  
 void insertMoney(VendingMachine machine, double amount);  
 void dispenseSnack(VendingMachine machine);  
}

* Idle.java

public class Idle implements StateOfVendingMachine {  
 private static Idle *instance* = null;  
 private Idle() {}  
 public static Idle getInstance() {  
 if (*instance* == null) *instance* = new Idle();  
 return *instance*;  
 }  
  
 @Override  
 public void selectSnack(VendingMachine machine, String snackName) {  
 Snack snack = machine.getSnackFromInventoryWithName(snackName);  
 if (snack == null) {  
 System.*out*.println("Snack not found");  
 return;  
 }  
 System.*out*.println("You selected 1 of " + snackName);  
 machine.setSelectedSnack(snack);  
 machine.setState(WaitingForMoney.*getInstance*());  
 }  
  
 @Override  
 public void insertMoney(VendingMachine machine, double amount) {  
 System.*out*.println("Please select a snack before inserting money");  
 }  
  
 @Override  
 public void dispenseSnack(VendingMachine machine) {  
 System.*out*.println("Please select a snack before you try to dispense one");  
 }  
}

* WaitingForMoney.java

public class WaitingForMoney implements StateOfVendingMachine {  
 private static WaitingForMoney *instance* = null;  
 private WaitingForMoney() {}  
 public static WaitingForMoney getInstance() {  
 if (*instance* == null) *instance* = new WaitingForMoney();  
 return *instance*;  
 }  
  
 @Override  
 public void selectSnack(VendingMachine machine, String snackName) {  
 System.*out*.println("Please wait until current transaction is complete to select another snack");  
 }  
  
 @Override  
 public void insertMoney(VendingMachine machine, double amount) {  
 machine.setAmountInserted(machine.getAmountInserted() + amount);  
 double snackPrice = machine.getSelectedSnack().getPrice();  
 double amountInserted = machine.getAmountInserted();  
 if (amountInserted >= snackPrice)  
 machine.setState(DispensingSnack.*getInstance*());  
 else  
 System.*out*.print("Please insert more money... Amount needed: $" + snackPrice + ", ");  
 System.*out*.println("Amount inserted so far: $" + amountInserted);  
 }  
  
 @Override  
 public void dispenseSnack(VendingMachine machine) {  
 System.*out*.println("Please wait until current transaction is complete to dispense snack");  
 }  
}

* DispensingSnack.java

public class DispensingSnack implements StateOfVendingMachine {  
 private static DispensingSnack *instance* = null;  
 private DispensingSnack() {}  
 public static DispensingSnack getInstance() {  
 if (*instance* == null) *instance* = new DispensingSnack();  
 return *instance*;  
 }  
  
 @Override  
 public void selectSnack(VendingMachine machine, String snackName) {  
 System.*out*.println("Please wait until the current snack has finished dispensing before selecting another");  
 }  
  
 @Override  
 public void insertMoney(VendingMachine machine, double amount) {  
 System.*out*.println("Please wait until the current snack has finished dispensing before inserting more money");  
 }  
  
 @Override  
 public void dispenseSnack(VendingMachine machine) {  
 machine.getDispenser().dispenseSnack(machine);  
 }  
}

* SnackDispenseHandler.java

public abstract class SnackDispenseHandler {  
 private SnackDispenseHandler next;  
  
 public SnackDispenseHandler(SnackDispenseHandler next) {  
 this.next = next;  
 }  
  
 public void dispenseSnack(VendingMachine machine) {  
 if (next != null)  
 next.dispenseSnack(machine);  
 }  
}

* Coke.java

public class Coke extends SnackDispenseHandler {  
 public Coke(SnackDispenseHandler next) {  
 super(next);  
 }  
  
 public void dispenseSnack(VendingMachine machine) {  
 Snack snack = machine.getSelectedSnack();  
  
 if (!snack.getName().equalsIgnoreCase("Coke")) {  
 super.dispenseSnack(machine);  
 return;  
 }  
  
 double change = machine.getAmountInserted();  
 if (snack.getQuantity() > 0) {  
 change -= snack.getPrice();  
 snack.decrementQuantity();  
 System.*out*.print("Enjoy your Coke... ");  
 } else {  
 System.*out*.print("Sorry, looks like we're out of Coke... ");  
 }  
  
 System.*out*.println("Here's your change: $" + change);  
 machine.setState(Idle.*getInstance*());  
 machine.setSelectedSnack(null);  
 machine.setAmountInserted(0);  
 }  
}

* Pepsi.java

public class Pepsi extends SnackDispenseHandler {  
 public Pepsi(SnackDispenseHandler next) {  
 super(next);  
 }  
  
 public void dispenseSnack(VendingMachine machine) {  
 Snack snack = machine.getSelectedSnack();  
  
 if (!snack.getName().equalsIgnoreCase("Pepsi")) {  
 super.dispenseSnack(machine);  
 return;  
 }  
  
 double change = machine.getAmountInserted();  
 if (snack.getQuantity() > 0) {  
 change -= snack.getPrice();  
 snack.decrementQuantity();  
 System.*out*.print("Enjoy your Pepsi... ");  
 } else {  
 System.*out*.print("Sorry, looks like we're out of Pepsi... ");  
 }  
  
 System.*out*.println("Here's your change: $" + change);  
 machine.setState(Idle.*getInstance*());  
 machine.setSelectedSnack(null);  
 machine.setAmountInserted(0);  
 }  
}

* Cheetos.java

public class Cheetos extends SnackDispenseHandler {  
 public Cheetos(SnackDispenseHandler next) {  
 super(next);  
 }  
  
 public void dispenseSnack(VendingMachine machine) {  
 Snack snack = machine.getSelectedSnack();  
  
 if (!snack.getName().equalsIgnoreCase("Cheetos")) {  
 super.dispenseSnack(machine);  
 return;  
 }  
  
 double change = machine.getAmountInserted();  
 if (snack.getQuantity() > 0) {  
 change -= snack.getPrice();  
 snack.decrementQuantity();  
 System.*out*.print("Enjoy your Cheetos... ");  
 } else {  
 System.*out*.print("Sorry, looks like we're out of Cheetos... ");  
 }  
  
 System.*out*.println("Here's your change: $" + change);  
 machine.setState(Idle.*getInstance*());  
 machine.setSelectedSnack(null);  
 machine.setAmountInserted(0);  
 }  
}

* Doritos.java

public class Doritos extends SnackDispenseHandler {  
 public Doritos(SnackDispenseHandler next) {  
 super(next);  
 }  
  
 public void dispenseSnack(VendingMachine machine) {  
 Snack snack = machine.getSelectedSnack();  
  
 if (!snack.getName().equalsIgnoreCase("Doritos")) {  
 super.dispenseSnack(machine);  
 return;  
 }  
  
 double change = machine.getAmountInserted();  
 if (snack.getQuantity() > 0) {  
 change -= snack.getPrice();  
 snack.decrementQuantity();  
 System.*out*.print("Enjoy your Doritos... ");  
 } else {  
 System.*out*.print("Sorry, looks like we're out of Doritos... ");  
 }  
  
 System.*out*.println("Here's your change: $" + change);  
 machine.setState(Idle.*getInstance*());  
 machine.setSelectedSnack(null);  
 machine.setAmountInserted(0);  
 }  
}

* KitKat.java

public class KitKat extends SnackDispenseHandler {  
 public KitKat(SnackDispenseHandler next) {  
 super(next);  
 }  
  
 public void dispenseSnack(VendingMachine machine) {  
 Snack snack = machine.getSelectedSnack();  
  
 if (!snack.getName().equalsIgnoreCase("KitKat")) {  
 super.dispenseSnack(machine);  
 return;  
 }  
  
 double change = machine.getAmountInserted();  
 if (snack.getQuantity() > 0) {  
 change -= snack.getPrice();  
 snack.decrementQuantity();  
 System.*out*.print("Enjoy your KitKat... ");  
 } else {  
 System.*out*.print("Sorry, looks like we're out of KitKat... ");  
 }  
  
 System.*out*.println("Here's your change: $" + change);  
 machine.setState(Idle.*getInstance*());  
 machine.setSelectedSnack(null);  
 machine.setAmountInserted(0);  
 }  
}

* Snickers.java

public class Snickers extends SnackDispenseHandler {  
 public Snickers(SnackDispenseHandler next) {  
 super(next);  
 }  
  
 public void dispenseSnack(VendingMachine machine) {  
 Snack snack = machine.getSelectedSnack();  
  
 if (!snack.getName().equalsIgnoreCase("Snickers")) {  
 super.dispenseSnack(machine);  
 return;  
 }  
  
 double change = machine.getAmountInserted();  
 if (snack.getQuantity() > 0) {  
 change -= snack.getPrice();  
 snack.decrementQuantity();  
 System.*out*.print("Enjoy your Snickers... ");  
 } else {  
 System.*out*.print("Sorry, looks like we're out of Snickers... ");  
 }  
  
 System.*out*.println("Here's your change: $" + change);  
 machine.setState(Idle.*getInstance*());  
 machine.setSelectedSnack(null);  
 machine.setAmountInserted(0);  
 }  
}

* VendingMachine.java

import java.util.ArrayList;  
import java.util.List;  
  
public class VendingMachine {  
 private StateOfVendingMachine state;  
 private SnackDispenseHandler dispenser;  
 private List<Snack> inventory;  
  
 private Snack selectedSnack = null;  
 private double amountInserted = 0;  
  
 public VendingMachine() {  
 state = Idle.*getInstance*();  
 dispenser = new Coke(new Pepsi(new Cheetos(new Doritos(new KitKat(new Snickers(null))))));  
 inventory = new ArrayList<>();  
 }  
  
 public void setState(StateOfVendingMachine state) {  
 this.state = state;  
 }  
  
 public SnackDispenseHandler getDispenser() {  
 return dispenser;  
 }  
  
 public void addSnackToInventory(Snack snack) { inventory.add(snack); }  
  
 public Snack getSnackFromInventoryWithName(String snackName) {  
 for (Snack snack: inventory)  
 if (snack.getName().equalsIgnoreCase(snackName))  
 return snack;  
 return null;  
 }  
  
 public Snack getSelectedSnack() {  
 return selectedSnack;  
 }  
  
 public void setSelectedSnack(Snack selectedSnack) {  
 this.selectedSnack = selectedSnack;  
 }  
  
 public double getAmountInserted() {  
 return amountInserted;  
 }  
  
 public void setAmountInserted(double amountInserted) {  
 this.amountInserted = amountInserted;  
 }  
  
 public void selectSnack(String snackName) {  
 state.selectSnack(this, snackName);  
 }  
  
 public void insertMoney(double amount) {  
 state.insertMoney(this, amount);  
 }  
  
 public void dispenseSnack() {  
 state.dispenseSnack(this);  
 }  
}

* Main.java

public class Main {  
 public static void main(String[] args) {  
 VendingMachine machine = new VendingMachine();  
 machine.addSnackToInventory(new Snack("Coke", 5, 1.25));  
 machine.addSnackToInventory(new Snack("Pepsi", 3, 1.50));  
 machine.addSnackToInventory(new Snack("Cheetos", 2, 1.75));  
 machine.addSnackToInventory(new Snack("Doritos", 4, 1.50));  
 machine.addSnackToInventory(new Snack("KitKat", 0, 1.00));  
 machine.addSnackToInventory(new Snack("Snickers", 1, 2.00));  
  
 // selecting a snack that doesn't exist  
 machine.selectSnack("Pringles");  
 System.*out*.println();  
  
 // trying to insert money or dispensing before selecting snack  
 machine.insertMoney(2.5);  
 machine.dispenseSnack();  
 System.*out*.println();  
  
 // selecting one Snickers  
 machine.selectSnack("Snickers");  
 System.*out*.println();  
  
 // trying to select another snack or dispensing while waiting for Snickers money  
 machine.selectSnack("Pepsi");  
 machine.dispenseSnack();  
 System.*out*.println();  
  
 // inserting money less than Snickers price and attempting to dispense  
 machine.insertMoney(0.5);  
 machine.dispenseSnack();  
 System.*out*.println();  
  
 // inserting more money  
 machine.insertMoney(2.0);  
 System.*out*.println();  
  
 // trying to select a snack or insert more money while Snickers is dispensing  
 machine.selectSnack("Pepsi");  
 machine.insertMoney(0.7);  
 System.*out*.println();  
  
 // dispensing Snickers  
 machine.dispenseSnack();  
 System.*out*.println();  
  
 // trying to insert money and dispense snack again  
 machine.insertMoney(1.3);  
 machine.dispenseSnack();  
 System.*out*.println();  
  
 // selecting another Snickers, inserting money and dispensing it again  
 machine.selectSnack("Snickers");  
 machine.insertMoney(2.0);  
 machine.dispenseSnack();  
 System.*out*.println();  
  
 // selecting a Pepsi, inserting money and dispensing it (x4)  
 for (int i = 0; i < 4; ++i) {  
 machine.selectSnack("Pepsi");  
 machine.insertMoney(1.5);  
 machine.dispenseSnack();  
 System.*out*.println();  
 }  
 }  
}

* Output

**A screenshot of a computer program

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Text Editor UML**

* Class Diagram

**A picture containing text, receipt, parallel, font

Description automatically generated**

* Object Diagram
  + Before

A screenshot of a computer

Description automatically generated with medium confidence

* + After

A screenshot of a computer

Description automatically generated with medium confidence