Kevin Wong
CS4800
HW8 State and Chain of Responsibility
May 6th, 2024

Github repo link: https://github.com/kdub8/CS4800-HW8-State-and-CoR.git

---

Source code:

```
package HW8.Snacks;
import HW8.Snack;
import HW8.SnackDispenseHandler;

/**
* The type Cheetos dispenser handler.
*/
public class CheetosDispenserHandler extends SnackDispenseHandler {
   /**
    * Instantiates a new Cheetos dispenser handler.
    *
    * @param next the next dispenser handler in the chain
    */
   public CheetosDispenserHandler(SnackDispenseHandler next) {
       super(next);
   }

   /**
    * Dispenses the specified snack.
    *
    * @param snack the snack to dispense
    * @return true if the snack was successfully dispensed, false otherwise
    */
   public boolean dispense(Snack snack){
       System.out.println("Vending Machine Mechanical Arm Moving to
Cheetos...");
       if (snack.getName().equals("Cheetos")){
           System.out.println("Please wait...Dispensing Cheetos...");
           if (snack.getQuantity() > 0){
               System.out.println("Successfully dispensed Cheetos.");
               snack.setQuantity(snack.getQuantity() - 1);
               return true;
           } else {
               System.out.println("Sorry, Vending Machine Out of Cheetos. Your
money will be returned soon...");
               return false;
           }
```

```java
        } else {
            // Pass the request to the next dispenser handler in the chain

            return super.dispense(snack);
        }
    }
}
```

```java
package HW8.Snacks;
import HW8.Snack;
import HW8.SnackDispenseHandler;

/**
* The type Coke dispenser handler.
*/
public class CokeDispenserHandler extends SnackDispenseHandler {

    /**
     * Instantiates a new Coke dispenser handler.
     *
     * @param next the next dispenser handler in the chain
     */
    public CokeDispenserHandler(SnackDispenseHandler next) {
        super(next);
    }

    /**
     * Dispenses the specified snack.
     *
     * @param snack the snack to dispense
     * @return true if the snack was successfully dispensed, false otherwise
     */
    public boolean dispense(Snack snack){
        System.out.println("Vending Machine Mechanical Arm Moving to Coke...");
        if (snack.getName().equals("Coke")){
            System.out.println("Please wait...Dispensing Coke...");
            if (snack.getQuantity() > 0){
                System.out.println("Successfully dispensed Coke.");
                snack.setQuantity(snack.getQuantity() - 1);
                return true;
            } else {
                System.out.println("Sorry, Vending Machine Out of Coke. Your
money will be returned soon...");
                return false;
            }
        } else {
            // Pass the request to the next dispenser handler in the chain
            return super.dispense(snack);
        }
    }
}
```

```
package HW8.Snacks;
import HW8.Snack;
import HW8.SnackDispenseHandler;

/**
* The type Doritos dispenser handler.
*/
public class DoritosDispenserHandler extends SnackDispenseHandler {
    /**
     * Instantiates a new Doritos dispenser handler.
     *
     * @param next the next dispenser handler in the chain
     */
    public DoritosDispenserHandler(SnackDispenseHandler next) {
        super(next);
    }

    /**
     * Dispenses the specified snack.
     *
     * @param snack the snack to dispense
     * @return true if the snack was successfully dispensed, false otherwise
     */
    public boolean dispense(Snack snack){
        System.out.println("Vending Machine Mechanical Arm Moving to
Doritos...");
        if (snack.getName().equals("Doritos")){
            System.out.println("Please wait...Dispensing Doritos...");
            if (snack.getQuantity() > 0){
                System.out.println("Successfully dispensed Doritos.");
                snack.setQuantity(snack.getQuantity() - 1);
                return true;
            } else {
                System.out.println("Sorry, Vending Machine Out of Doritos. Your
money will be returned soon...");
                return false;
            }
        } else {
            // Pass the request to the next dispenser handler in the chain
            return super.dispense(snack);
        }
    }
}
```

```
package HW8.Snacks;
import HW8.Snack;
```

```java
import HW8.SnackDispenseHandler;

/**
* The type Kit kat dispenser handler.
*/
public class KitKatDispenserHandler extends SnackDispenseHandler {
   /**
     * Instantiates a new Kit kat dispenser handler.
     *
     * @param next the next
     */
   public KitKatDispenserHandler(SnackDispenseHandler next) {
       super(next);
   }


   /**
     * Dispenses the specified snack.
     *
     * @param snack the snack to dispense
     * @return true if the snack was successfully dispensed, false otherwise
     */
   public boolean dispense(Snack snack){
       System.out.println("Vending Machine Mechanical Arm Moving to
KitKat...");
       if (snack.getName().equals("KitKat")){
           System.out.println("Please wait...Dispensing KitKat...");
           if (snack.getQuantity() > 0){
               System.out.println("Successfully dispensed KitKat.");
               snack.setQuantity(snack.getQuantity() - 1);
               return true;
           } else {
               System.out.println("Sorry, Vending Machine Out of KitKat. Your
money will be returned soon...");
               return false;
           }
       } else {
           // Pass the request to the next dispenser handler in the chain
           return super.dispense(snack);
       }
   }
}
```

```java
package HW8.Snacks;
import HW8.Snack;
import HW8.SnackDispenseHandler;

/**
* The type Pepsi dispenser handler.
```

```
*/
public class PepsiDispenserHandler extends SnackDispenseHandler {
   /**
    * Instantiates a new Pepsi dispenser handler.
    *
    * @param next the next
    */
   public PepsiDispenserHandler(SnackDispenseHandler next) {
       super(next);
   }

   /**
    * Dispenses the specified snack.
    *
    * @param snack the snack to dispense
    * @return true if the snack was successfully dispensed, false otherwise
    */
   public boolean dispense(Snack snack){
       System.out.println("Vending Machine Mechanical Arm Moving to Pepsi...");
       if (snack.getName().equals("Pepsi")){
           System.out.println("Please wait...Dispensing Pepsi...");
           if (snack.getQuantity() > 0){
               System.out.println("Successfully dispensed Pepsi.");
               snack.setQuantity(snack.getQuantity() - 1);
               return true;
           } else {
               System.out.println("Sorry, Vending Machine Out of Pepsi. Your
money will be returned soon...");
               return false;
           }
       } else {
           // Pass the request to the next dispenser handler in the chain
           return super.dispense(snack);
       }
   }
}
```

```
package HW8.Snacks;

import HW8.Snack;
import HW8.SnackDispenseHandler;

/**
* The type Snickers dispenser handler.
*/
public class SnickersDispenserHandler extends SnackDispenseHandler {

   /**
```

```
     * Instantiates a new Snickers dispenser handler.
     *
     * @param next the next
     */
    public SnickersDispenserHandler(SnackDispenseHandler next) {
        super(next);
    }

    /**
     * Dispenses the specified snack.
     *
     * @param snack the snack to dispense
     * @return true if the snack was successfully dispensed, false otherwise
     */
    public boolean dispense(Snack snack){
        System.out.println("Vending Machine Mechanical Arm Moving to
Snickers...");
        if (snack.getName().equals("Snickers") ){
            System.out.println("Please wait...Dispensing Snickers...");
            if (snack.getQuantity() > 0){
                System.out.println("Successfully dispensed Snickers.");
                snack.setQuantity(snack.getQuantity() - 1);
                return true;
            } else {
                System.out.println("Sorry, Vending Machine Out of Snickers. Your
money will be returned soon...");
                return false;
            }
        } else {
            // Pass the request to the next dispenser handler in the chain
            return super.dispense(snack);
        }
    }
}
```

```
package HW8.VendingMachineStates;

import HW8.StateOfVendingMachine;
import HW8.VendingMachine;

/**
* Represents the state of the vending machine when it is waiting for the user
to insert money after selecting a snack.
*/
public class WaitingForMoneyState implements StateOfVendingMachine {

    /**
     * {@inheritDoc}
     * If the customer changes their mind about the snack, checks if the new
snack option is valid.
     */
    @Override
    public void snackOptionPressed(VendingMachine vendingMachine, String
snackName) {
        // This if customer changes their mind of what they want
        // Check to see if snackName is valid.
        if (vendingMachine.isValidSnackOption(snackName)){
            vendingMachine.setState(new WaitingForMoneyState());

vendingMachine.setSnackOptionChosen(vendingMachine.getSnack(snackName));
        } else {
            System.out.println(snackName + " is not a valid option!");
        }
    }

    /**
     * {@inheritDoc}
     * Adds the inserted money to the total, checks if enough money is inserted
to dispense the selected snack,
     * and updates the vending machine's state accordingly.
     */
    @Override
    public void moneyInserted(VendingMachine vendingMachine, Double money) {
        vendingMachine.setMoneyInserted(vendingMachine.getMoneyInserted() +
money);
        Double moneyInserted = vendingMachine.getMoneyInserted();
        Double snackPrice = vendingMachine.getSnackOptionChosen().getPrice();
        System.out.println("Inserted Total: $" + moneyInserted);
        if (moneyInserted >= snackPrice){
            vendingMachine.setState(new DispensingSnackState());
            System.out.println("Required amount of $"
+vendingMachine.getSnackOptionChosen().getPrice() + " fulfilled. Vending
Machine is now ready to dispense.");
        } else {
            System.out.println("Not enough money inserted. Price for " +
vendingMachine.getSnackOptionChosen().getName() +" is: $" + snackPrice);
        }
    }

    /**
     * {@inheritDoc}
     * Prints a message indicating that the snack cannot be dispensed yet, as
the required amount of money has not been inserted.
     */
    @Override
    public void dispenseSnack(VendingMachine vendingMachine) {
        System.out.println("Cannot dispense, not enough money! Money inserted:
$" + vendingMachine.getMoneyInserted() +
               " Price for " + vendingMachine.getSnackOptionChosen().getName()
+ " is: $" + vendingMachine.getSnackOptionChosen().getPrice());
    }

    /**
     * {@inheritDoc}
     * Resets the snack selection and transitions back to the idle state,
keeping the money in the machine for a refund.
     */
    @Override
    public void processRefund(VendingMachine vendingMachine) {
        // Resets selection and keeps money in moneyInserted
        System.out.println("Snack option has been deselected, please collect
your refund at the change dispenser.");
        vendingMachine.setState(new IdleState());
    }
}
```

```
package HW8.VendingMachineStates;

import HW8.StateOfVendingMachine;
import HW8.VendingMachine;
/**
* Represents the state of the vending machine when it is idle, waiting for
user input.
*/
public class IdleState implements StateOfVendingMachine {

   /**
    * {@inheritDoc}
    * Checks if the selected snack option is valid. If valid, sets the vending
machine's state
    * to WaitingForMoneyState and sets the selected snack option.
    */
   @Override
   public void snackOptionPressed(VendingMachine vendingMachine, String
snackName) {
       // Check to see if snackName is valid.
       if (vendingMachine.isValidSnackOption(snackName)){
           vendingMachine.setState(new WaitingForMoneyState());

vendingMachine.setSnackOptionChosen(vendingMachine.getSnack(snackName));
           System.out.println(vendingMachine.getSnackOptionChosen().getName() +
" chosen.");
       } else {
           System.out.println(snackName + " is not a valid option!");
       }
   }
   /**
    * {@inheritDoc}
    * Prints a message indicating that a snack option must be selected before
inserting money.
    */
   @Override
   public void moneyInserted(VendingMachine vendingMachine, Double money) {
       System.out.println("Please select a valid snack option before inserting
money");
   }
   /**
    * {@inheritDoc}
    * Prints a message indicating that a snack and money must be selected
before dispensing.
    */
   @Override
   public void dispenseSnack(VendingMachine vendingMachine) {
       System.out.println("Cannot dispense snack before selecting snack and
inserting money.");
   }
   /**
    * {@inheritDoc}
    * Empty implementation as refund is not applicable in the idle state.
    */
   @Override
   public void processRefund(VendingMachine vendingMachine) {
   }
}
```

```
package HW8.VendingMachineStates;
import HW8.StateOfVendingMachine;
import HW8.VendingMachine;
import HW8.Snack;

/**
* The state representing the vending machine when a snack is being dispensed.
*/
public class DispensingSnackState implements StateOfVendingMachine {

    /**
     * {@inheritDoc}
     * In this state, selecting a snack is not allowed.
     */
    @Override
    public void snackOptionPressed(VendingMachine vendingMachine, String
snackName) {
        System.out.println("Cannot select snack during dispensation");
    }
    /**
     * {@inheritDoc}
     * In this state, inserting more money than required is not allowed.
     */
    @Override
    public void moneyInserted(VendingMachine vendingMachine, Double money) {
        System.out.println("Cannot insert more money than required.");
    }
    /**
     * {@inheritDoc}
     * Dispenses the selected snack if available and updates the vending
machine's state.
     */
    @Override
    public void dispenseSnack(VendingMachine vendingMachine) {
        Snack snackOption = vendingMachine.getSnackOptionChosen();

        boolean success =
vendingMachine.getSnackDispenser().dispenseSnack(snackOption);
        double snackPrice = snackOption.getPrice();
        double moneyInserted = vendingMachine.getMoneyInserted();

        if (success){
            // Increment the total money earned by the vending machine
            vendingMachine.setMoneyEarned(vendingMachine.getMoneyEarned() +
snackPrice);
            // Deduct the price of the snack from the total money inserted
            vendingMachine.setMoneyInserted(moneyInserted - snackPrice);
        }
        // Transition to the idle state after dispensing the snack
        vendingMachine.setState(new IdleState());
    }
    /**
     * {@inheritDoc}
     * Refunding is not possible while a snack is being dispensed.
     */
    @Override
    public void processRefund(VendingMachine vendingMachine) {
        System.out.println("Cannot issue refund. Snack is being dispensed...");
    }

}
```

```
package HW8;

/**
* Represents a snack item in the vending machine.
```

```java
*/
public class Snack {
   int quantity;
   String name;

   Double price;


   /**
    * Default constructor.
    */
   public Snack() {
   }

   /**
    * Constructor to initialize a snack with name, price, and quantity.
    *
    * @param name      The name of the snack.
    * @param price     The price of the snack.
    * @param quantity The quantity of the snack available.
    */
   public Snack(String name, Double price, int quantity) {
       this.name = name;
       this.price = price;
       this.quantity = quantity;
   }


   /**
    * Gets price.
    *
    * @return the price
    */
   public Double getPrice() {
       return price;
   }

   /**
    * Sets price.
    *
    * @param price the price
    */
   public void setPrice(Double price) {
       this.price = price;
   }

   /**
    * Gets quantity.
    *
```

```java
     * @return the quantity
     */
    public int getQuantity() {
        return quantity;
    }

    /**
     * Sets quantity.
     *
     * @param quantity the quantity
     */
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    /**
     * Gets name.
     *
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * Sets name.
     *
     * @param name the name
     */
    public void setName(String name) {
        this.name = name;
    }
}
```

```java
package HW8;
/**
* Abstract class representing a handler for dispensing snacks in a vending
machine.
*/
public abstract class SnackDispenseHandler {
   private SnackDispenseHandler next;
   private Snack snack;
   /**
     * Constructs a new snack dispenser handler with the next handler in the
chain.
     *
     * @param next The next handler in the chain.
     */
   public SnackDispenseHandler(SnackDispenseHandler next){
        this.next = next;
    }

    /**
     * Dispenses the specified snack, delegating to the next handler if
necessary.
     *
     * @param snack The snack to dispense.
     * @return True if the snack was successfully dispensed, false otherwise.
     */
    public boolean dispense(Snack snack){
        if (next != null){
            return next.dispense(snack);
        }
        return false;
    }
}
```

```
package HW8;

import HW8.Snacks.*;
/**
* Class representing a snack dispenser in a vending machine.
*/
public class SnackDispenser {

    private SnackDispenseHandler chainOfResponsibility =
            new CokeDispenserHandler(
                    new PepsiDispenserHandler(
                            new CheetosDispenserHandler(
                                    new DoritosDispenserHandler(
                                            new KitKatDispenserHandler(
                                                    new
SnickersDispenserHandler(
                                                            null))))));
    /**
     * Dispenses the specified snack using the chain of responsibility.
     *
     * @param snack The snack to dispense.
     * @return True if the snack was successfully dispensed, false otherwise.
     */
    public boolean dispenseSnack(Snack snack){
        return chainOfResponsibility.dispense(snack);
    }
}
```

```
package HW8;
/**
```

```java
* Interface representing the different states of a vending machine.
*/
public interface StateOfVendingMachine {
   /**
    * Handles the event of a snack option being pressed.
    *
    * @param vendingMachine The vending machine.
    * @param snackName      The name of the snack.
    */
   void snackOptionPressed(VendingMachine vendingMachine, String snackName);
   /**
    * Handles the event of money being inserted.
    *
    * @param vendingMachine The vending machine.
    * @param money           The amount of money inserted.
    */
   void moneyInserted(VendingMachine vendingMachine, Double money);
   /**
    * Handles the event of dispensing a snack.
    *
    * @param vendingMachine The vending machine.
    */
   void dispenseSnack(VendingMachine vendingMachine);
   /**
    * Handles the event of a refund being requested.
    *
    * @param vendingMachine The vending machine.
    */
   void processRefund(VendingMachine vendingMachine);
}
```

```java
package HW8;

import HW8.VendingMachineStates.IdleState;

import java.util.HashMap;
/**
* Represents a vending machine that dispenses snacks.
*/
public class VendingMachine {

   private StateOfVendingMachine stateOfVendingMachine = new IdleState();
   private SnackDispenser snackDispenser = new SnackDispenser();
   private Snack snackOptionChosen;
   private Double moneyInserted = 0.0;
   private Double moneyEarned = 0.0;

   private static HashMap<String, Snack> snacks = new HashMap<>();
```

```java
    static {
        // Initialize the snacks available in the vending machine
        // Format: Name, Price, Quantity
        Snack coke = new Snack("Coke", 3.50, 5);
        Snack pepsi = new Snack("Pepsi", 3.25, 2);
        Snack cheetos = new Snack("Cheetos", 2.25, 3);
        Snack doritos = new Snack("Doritos", 2.25, 1);
        Snack kitkat = new Snack("KitKat", 1.50, 6);
        Snack snickers = new Snack("Snickers", 1.50, 1);
        snacks.put("Coke", coke);
        snacks.put("Pepsi", pepsi);
        snacks.put("Cheetos", cheetos);
        snacks.put("Doritos", doritos);
        snacks.put("KitKat", kitkat);
        snacks.put("Snickers", snickers);
    }

    public VendingMachine() {
    }

    public StateOfVendingMachine getState(){
        return stateOfVendingMachine;
    }

    public void setState(StateOfVendingMachine state){
        this.stateOfVendingMachine = state;
    }
    /**
     * Insert money into the vending machine.
     *
     * @param money The amount of money to insert.
     */
    public void insertMoney(Double money){
        getState().moneyInserted(this, money);
    }
    /**
     * Select a snack from the vending machine.
     *
     * @param snackOption The name of the snack to select.
     */
    public void selectSnack(String snackOption){
        getState().snackOptionPressed(this, snackOption);
    }

    public Snack getSnackOptionChosen() {
        return snackOptionChosen;
    }

    public void setSnackOptionChosen(Snack snackOptionChosen) {
        this.snackOptionChosen = snackOptionChosen;
    }


    public Double getMoneyInserted() {
        return moneyInserted;
    }

    public void setMoneyInserted(Double moneyInserted) {
        this.moneyInserted = moneyInserted;
    }

    public Double getMoneyEarned() {
        return moneyEarned;
    }

    public void setMoneyEarned(Double moneyEarned) {
        this.moneyEarned = moneyEarned;
    }

    public double getSnackPrice(String name){
        if (isValidSnackOption(name)){
            return snacks.get(name).getPrice();
        }
        return 0.0;
    }

    public Snack getSnack(String name){
        if (isValidSnackOption(name)){
            return snacks.get(name);
        }
        return null;
    }
    /**
     * Check if a snack option is valid.
     *
     * @param name The name of the snack option.
     * @return True if the snack option is valid, false otherwise.
     */
    public boolean isValidSnackOption(String name){
        return snacks.containsKey(name);
    }

    public SnackDispenser getSnackDispenser() {
        return snackDispenser;
    }
    /**
     * Dispense the selected snack.
     */
    public void dispenseSnack(){
        getState().dispenseSnack(this);
    }
    /**
     * Retrieve change from the vending machine.
     *
     * @return The amount of change retrieved.
     */
    public double retrieveChange(){
        getState().processRefund(this);
        if (this.moneyInserted <= 0) {
            System.out.println("Cannot issue refund.");
            this.moneyInserted = 0.0;
        } else {
            System.out.println("Giving $" + this.moneyInserted + " for change");
        }
        double change = this.moneyInserted;
        this.moneyInserted = 0.0;
        return change;
    }
}
```

Junit tests

```
package tests;

import HW8.VendingMachine;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

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

public class VendingMachineTests {
    private VendingMachine vendingMachine;

    @BeforeEach
    public void setUp() {
        vendingMachine = new VendingMachine();
    }

    @Test
    public void testInsertMoney() {
        vendingMachine.selectSnack("Cheetos");
        vendingMachine.insertMoney(5.0);
        assertEquals(5.0, vendingMachine.getMoneyInserted(), 0.001);
    }
```

```java
   @Test
   public void testSelectSnack() {
       vendingMachine.selectSnack("Snickers");
       assertEquals("Snickers",
vendingMachine.getSnackOptionChosen().getName());
   }

   @Test
   public void testRetrieveChange() {
       vendingMachine.selectSnack("Snickers");
       vendingMachine.insertMoney(5.0);
       vendingMachine.dispenseSnack();
       double change = vendingMachine.retrieveChange();
       assertEquals(5.0 - 1.50, change, 0.001);

       vendingMachine.selectSnack("Pepsi");
       vendingMachine.insertMoney(10.0);
       vendingMachine.dispenseSnack();
       double change_2 = vendingMachine.retrieveChange();
       assertEquals(10.0 - 3.25, change_2, 0.001);
   }

   @Test
   public void testMoneyEarned() {
       vendingMachine.selectSnack("KitKat");
       vendingMachine.insertMoney(5.0);
       vendingMachine.dispenseSnack();
       assertEquals(5.0 - vendingMachine.retrieveChange(),
vendingMachine.getMoneyEarned());
   }

   @Test
   public void testGetSnackPrice () {
       assertEquals(2.25, vendingMachine.getSnackPrice("Cheetos"));
   }

   @Test
   public void testIsValidSnack() {
       assertEquals(true, vendingMachine.isValidSnackOption("Coke"));
       assertEquals(true, vendingMachine.isValidSnackOption("Cheetos"));
       assertEquals(true, vendingMachine.isValidSnackOption("Doritos"));
       assertEquals(true, vendingMachine.isValidSnackOption("Pepsi"));
       assertEquals(true, vendingMachine.isValidSnackOption("KitKat"));
       assertEquals(true, vendingMachine.isValidSnackOption("Snickers"));
       assertEquals(false, vendingMachine.isValidSnackOption("Funyuns"));
       assertEquals(false, vendingMachine.isValidSnackOption("Cookies"));
   }
```

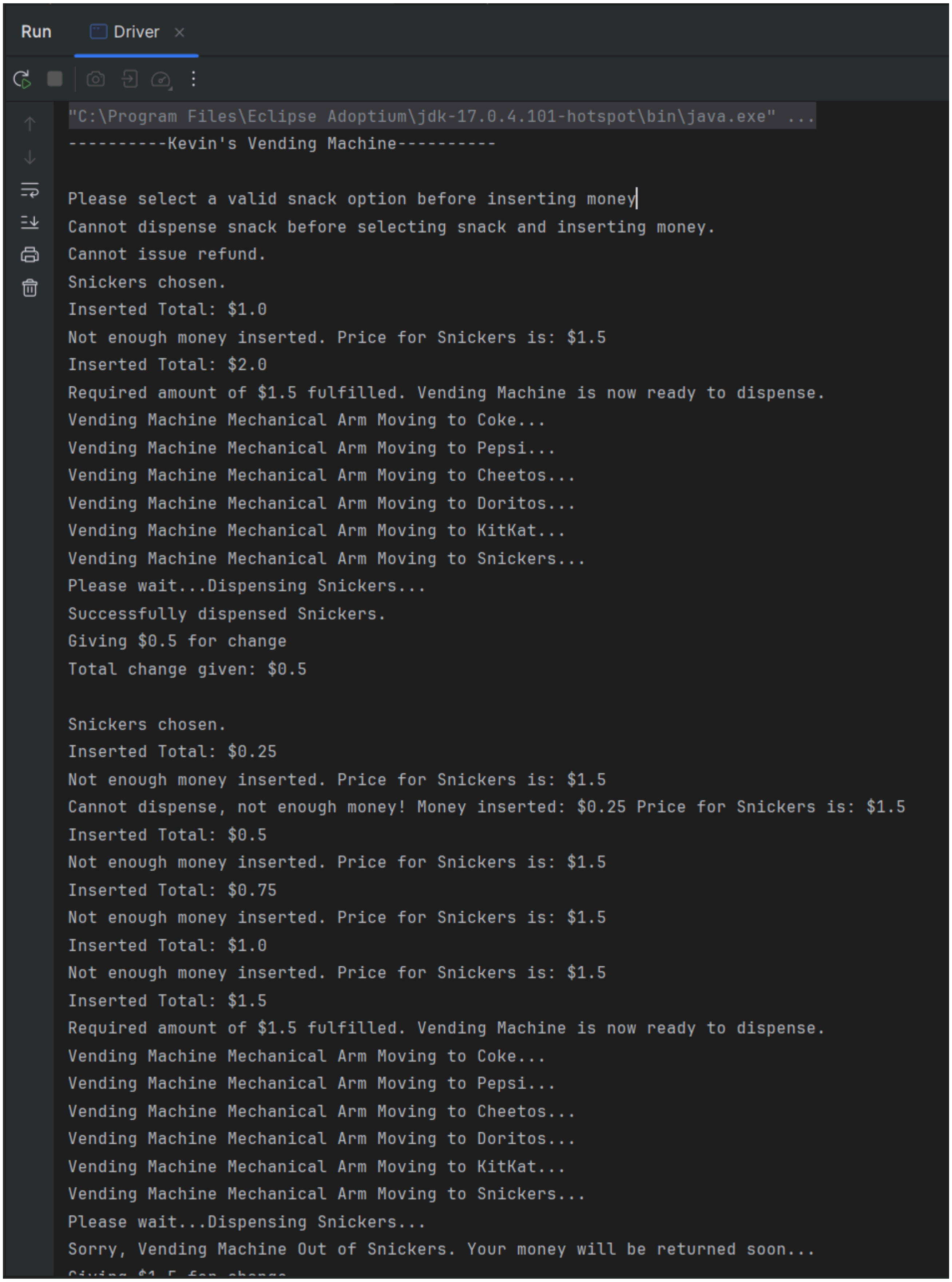
```
}
```

Driver

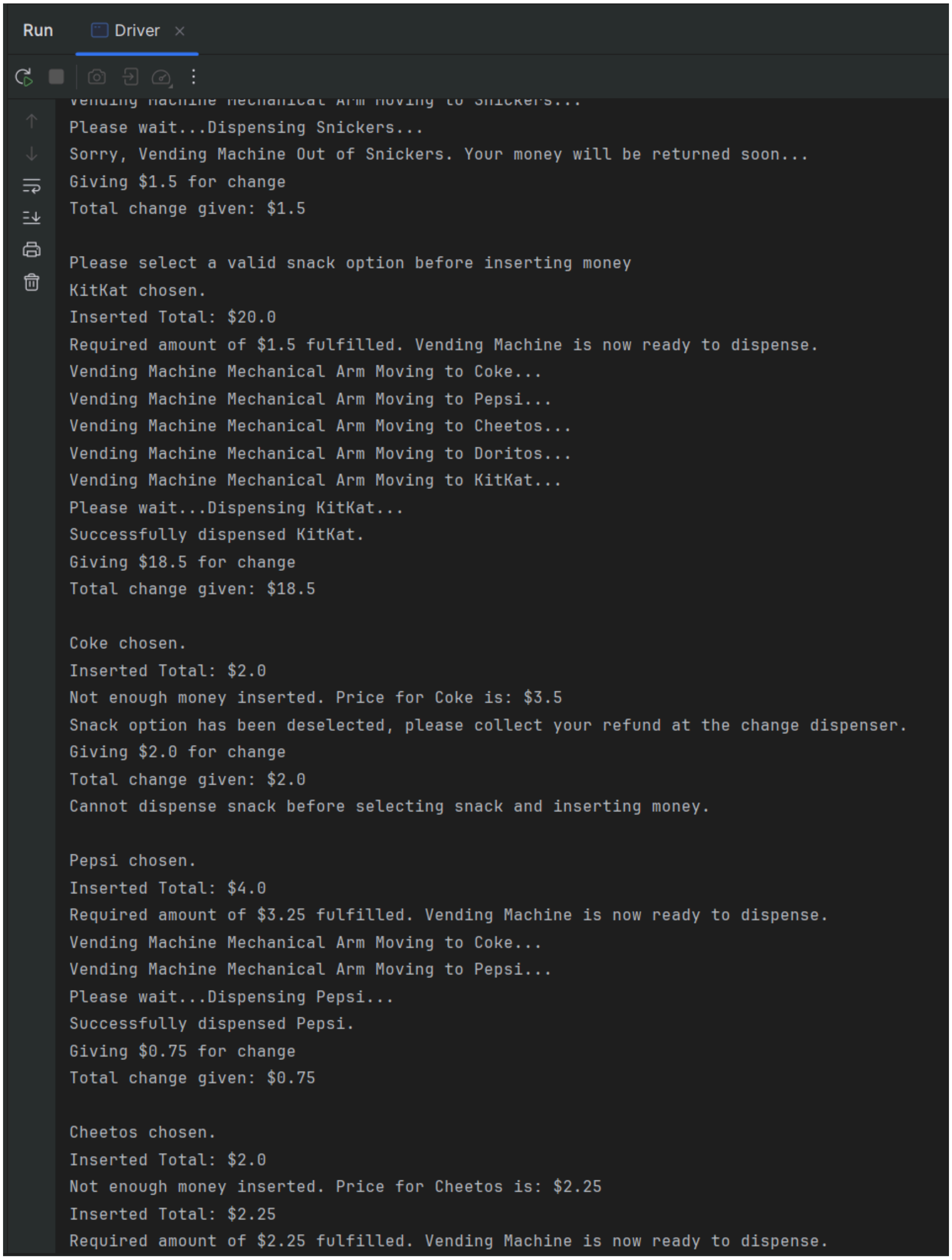
```
package HW8;
/**
* The main driver class to demonstrate the functionality of Kevin's Vending
Machine.
*/
public class Driver {

   public static void main(String[] args){
       System.out.println("----------Kevin's Vending Machine----------\n");
       VendingMachine vendingMachine = new VendingMachine();

       // Testing various scenarios
       vendingMachine.insertMoney(5.0);  //should print out "Please select an
option before inserting money"
       vendingMachine.dispenseSnack();     //should print out "Cannot dispense
snack before selecting snack and inserting money."
       vendingMachine.retrieveChange();    //prints "Giving 0.0 for change"
       vendingMachine.selectSnack("Snickers");
       vendingMachine.insertMoney(1.0);
       vendingMachine.insertMoney(1.0);    //rejected because snickers is $1
and change is given

       vendingMachine.dispenseSnack();
       System.out.println("Total change given: $" +
vendingMachine.retrieveChange());
       System.out.println();

       vendingMachine.selectSnack("Snickers");
       vendingMachine.insertMoney(0.25);
       vendingMachine.dispenseSnack();     //prints out "Not enough money!
Money given: 0.25 Price for Snickers is: 1.0"
       vendingMachine.insertMoney(0.25);
       vendingMachine.insertMoney(0.25);
       vendingMachine.insertMoney(0.25);
       vendingMachine.insertMoney(0.50);
       vendingMachine.dispenseSnack(); //snickers are out of stock
       System.out.println("Total change given: $" +
vendingMachine.retrieveChange());
       System.out.println();
       vendingMachine.insertMoney(20.0);
       vendingMachine.selectSnack("KitKat");
       vendingMachine.insertMoney(20.0);
       vendingMachine.dispenseSnack();
       System.out.println("Total change given: $" +
vendingMachine.retrieveChange());
       System.out.println();

       vendingMachine.selectSnack("Coke");
       vendingMachine.insertMoney(2.0);
       System.out.println("Total change given: $" +
vendingMachine.retrieveChange());
       vendingMachine.dispenseSnack(); //attempt to dispense coke after getting
change, should be rejected
       System.out.println();

       vendingMachine.selectSnack("Pepsi");
       vendingMachine.insertMoney(4.0);
       vendingMachine.dispenseSnack();
       System.out.println("Total change given: $" +
vendingMachine.retrieveChange());
       System.out.println();
       //no issues

       vendingMachine.selectSnack("Cheetos");
       vendingMachine.insertMoney(2.0);
       vendingMachine.insertMoney(0.25);
       vendingMachine.dispenseSnack();
       System.out.println("Total change given: $" +
vendingMachine.retrieveChange());
       System.out.println();
       //no issues

       vendingMachine.selectSnack("Funyuns");
       vendingMachine.selectSnack("Doritos");
       vendingMachine.insertMoney(10.0);
       vendingMachine.dispenseSnack();
       System.out.println("Total change given: $" +
vendingMachine.retrieveChange());
       System.out.println();
       //no issues
   }
}
```

Driver output from IDE terminal with 6 different snacks and the Chain of Responsibility in the required order of Coke, Pepsi, Cheetos, Doritos, KitKat, and then Snickers.

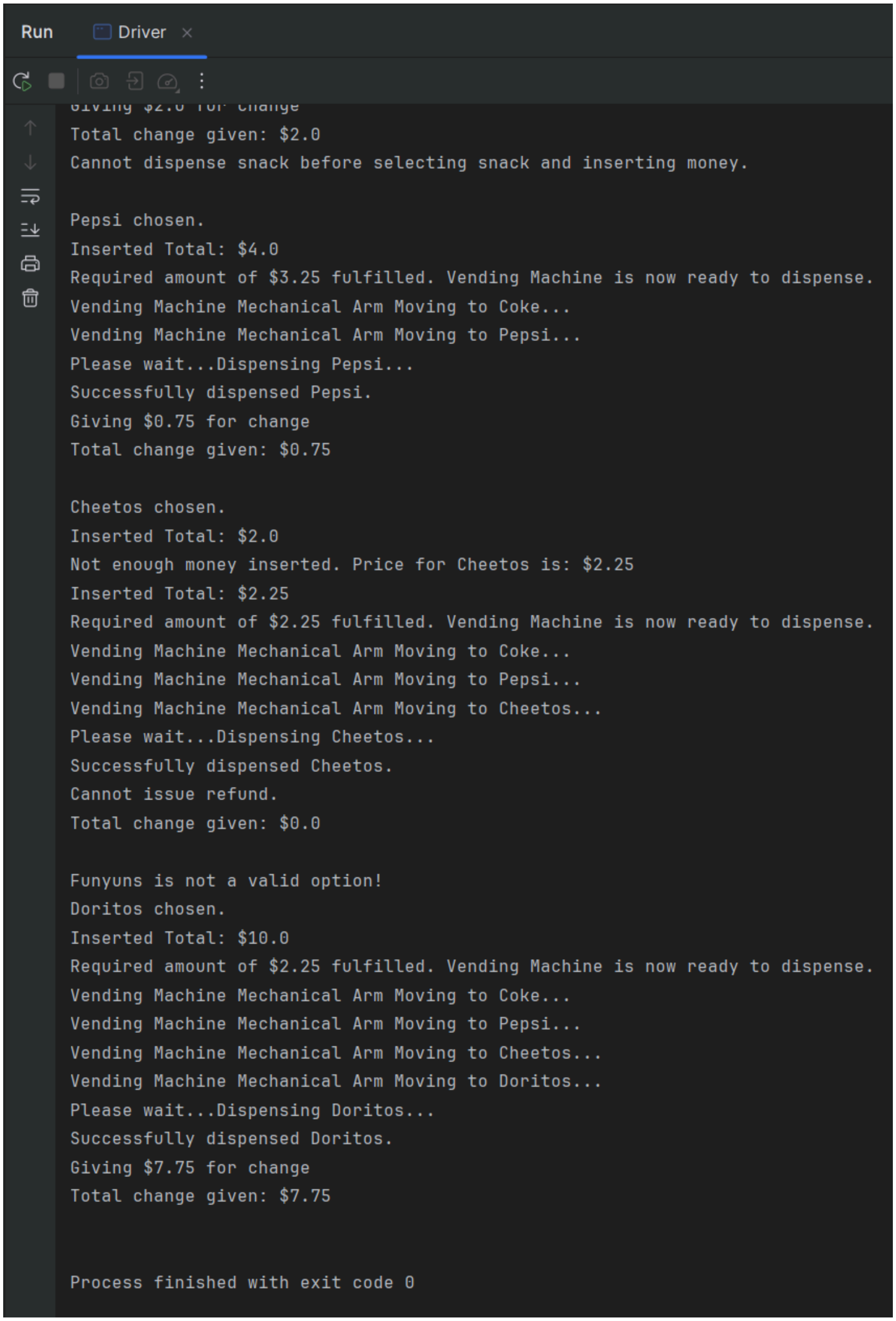
Driver includes the required case where the quantity hits 0 with snickers. (Bottom of the 1st output screenshot and top of the 2nd output screenshot)



```
"C:\Program Files\Eclipse Adoptium\jdk-17.0.4.101-hotspot\bin\java.exe" ...
----------Kevin's Vending Machine----------

Please select a valid snack option before inserting money
Cannot dispense snack before selecting snack and inserting money.
Cannot issue refund.
Snickers chosen.
Inserted Total: $1.0
Not enough money inserted. Price for Snickers is: $1.5
Inserted Total: $2.0
Required amount of $1.5 fulfilled. Vending Machine is now ready to dispense.
Vending Machine Mechanical Arm Moving to Coke...
Vending Machine Mechanical Arm Moving to Pepsi...
Vending Machine Mechanical Arm Moving to Cheetos...
Vending Machine Mechanical Arm Moving to Doritos...
Vending Machine Mechanical Arm Moving to KitKat...
Vending Machine Mechanical Arm Moving to Snickers...
Please wait...Dispensing Snickers...
Successfully dispensed Snickers.
Giving $0.5 for change
Total change given: $0.5

Snickers chosen.
Inserted Total: $0.25
Not enough money inserted. Price for Snickers is: $1.5
Cannot dispense, not enough money! Money inserted: $0.25 Price for Snickers is: $1.5
Inserted Total: $0.5
Not enough money inserted. Price for Snickers is: $1.5
Inserted Total: $0.75
Not enough money inserted. Price for Snickers is: $1.5
Inserted Total: $1.0
Not enough money inserted. Price for Snickers is: $1.5
Inserted Total: $1.5
Required amount of $1.5 fulfilled. Vending Machine is now ready to dispense.
Vending Machine Mechanical Arm Moving to Coke...
Vending Machine Mechanical Arm Moving to Pepsi...
Vending Machine Mechanical Arm Moving to Cheetos...
Vending Machine Mechanical Arm Moving to Doritos...
Vending Machine Mechanical Arm Moving to KitKat...
Vending Machine Mechanical Arm Moving to Snickers...
Please wait...Dispensing Snickers...
Sorry, Vending Machine Out of Snickers. Your money will be returned soon...
Giving $1.5 for change
```



```
Vending Machine Mechanical Arm Moving to Snickers...
Please wait...Dispensing Snickers...
Sorry, Vending Machine Out of Snickers. Your money will be returned soon...
Giving $1.5 for change
Total change given: $1.5

Please select a valid snack option before inserting money
KitKat chosen.
Inserted Total: $20.0
Required amount of $1.5 fulfilled. Vending Machine is now ready to dispense.
Vending Machine Mechanical Arm Moving to Coke...
Vending Machine Mechanical Arm Moving to Pepsi...
Vending Machine Mechanical Arm Moving to Cheetos...
Vending Machine Mechanical Arm Moving to Doritos...
Vending Machine Mechanical Arm Moving to KitKat...
Please wait...Dispensing KitKat...
Successfully dispensed KitKat.
Giving $18.5 for change
Total change given: $18.5

Coke chosen.
Inserted Total: $2.0
Not enough money inserted. Price for Coke is: $3.5
Snack option has been deselected, please collect your refund at the change dispenser.
Giving $2.0 for change
Total change given: $2.0
Cannot dispense snack before selecting snack and inserting money.

Pepsi chosen.
Inserted Total: $4.0
Required amount of $3.25 fulfilled. Vending Machine is now ready to dispense.
Vending Machine Mechanical Arm Moving to Coke...
Vending Machine Mechanical Arm Moving to Pepsi...
Please wait...Dispensing Pepsi...
Successfully dispensed Pepsi.
Giving $0.75 for change
Total change given: $0.75

Cheetos chosen.
Inserted Total: $2.0
Not enough money inserted. Price for Cheetos is: $2.25
Inserted Total: $2.25
Required amount of $2.25 fulfilled. Vending Machine is now ready to dispense.
```

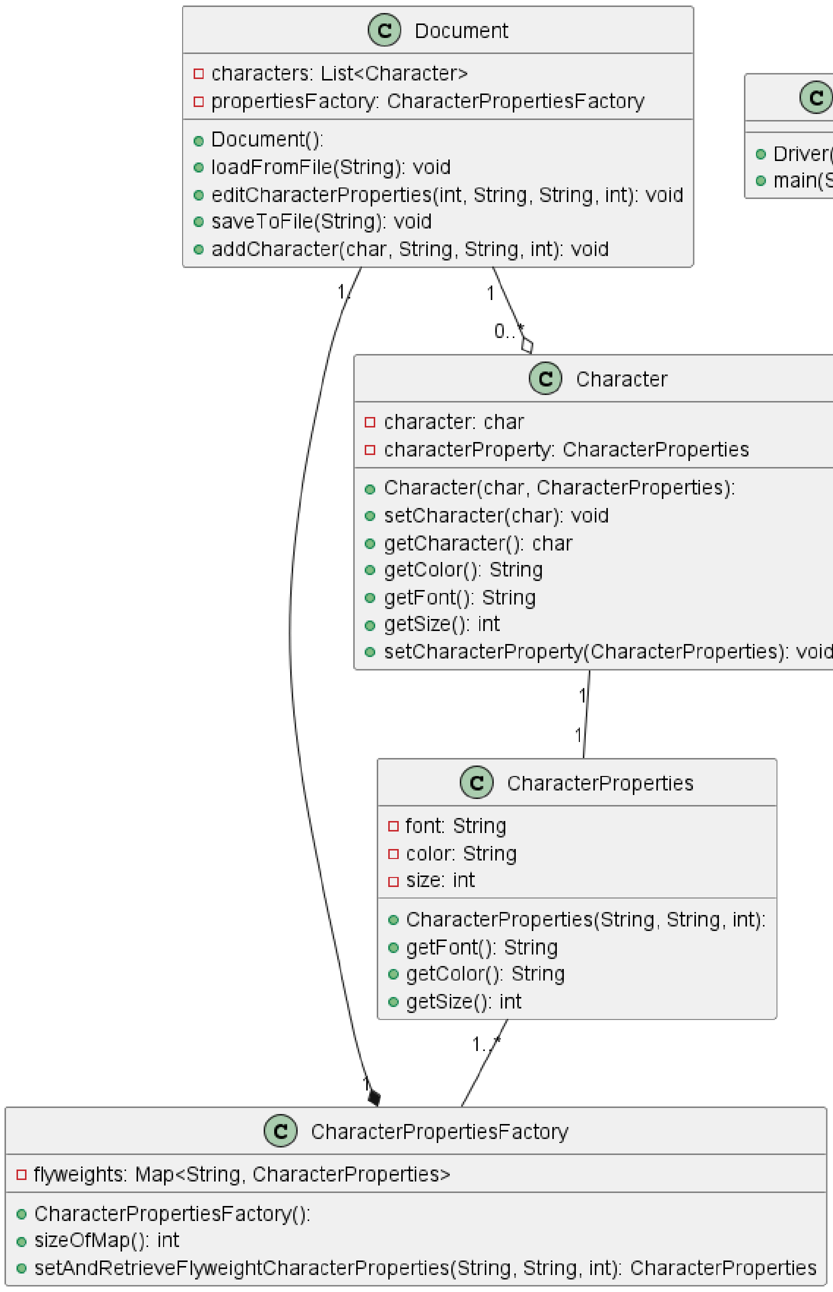


Run   Driver

```
Total change given: $2.0
Cannot dispense snack before selecting snack and inserting money.

Pepsi chosen.
Inserted Total: $4.0
Required amount of $3.25 fulfilled. Vending Machine is now ready to dispense.
Vending Machine Mechanical Arm Moving to Coke...
Vending Machine Mechanical Arm Moving to Pepsi...
Please wait...Dispensing Pepsi...
Successfully dispensed Pepsi.
Giving $0.75 for change
Total change given: $0.75

Cheetos chosen.
Inserted Total: $2.0
Not enough money inserted. Price for Cheetos is: $2.25
Inserted Total: $2.25
Required amount of $2.25 fulfilled. Vending Machine is now ready to dispense.
Vending Machine Mechanical Arm Moving to Coke...
Vending Machine Mechanical Arm Moving to Pepsi...
Vending Machine Mechanical Arm Moving to Cheetos...
Please wait...Dispensing Cheetos...
Successfully dispensed Cheetos.
Cannot issue refund.
Total change given: $0.0

Funyuns is not a valid option!
Doritos chosen.
Inserted Total: $10.0
Required amount of $2.25 fulfilled. Vending Machine is now ready to dispense.
Vending Machine Mechanical Arm Moving to Coke...
Vending Machine Mechanical Arm Moving to Pepsi...
Vending Machine Mechanical Arm Moving to Cheetos...
Vending Machine Mechanical Arm Moving to Doritos...
Please wait...Dispensing Doritos...
Successfully dispensed Doritos.
Giving $7.75 for change
Total change given: $7.75


Process finished with exit code 0
```

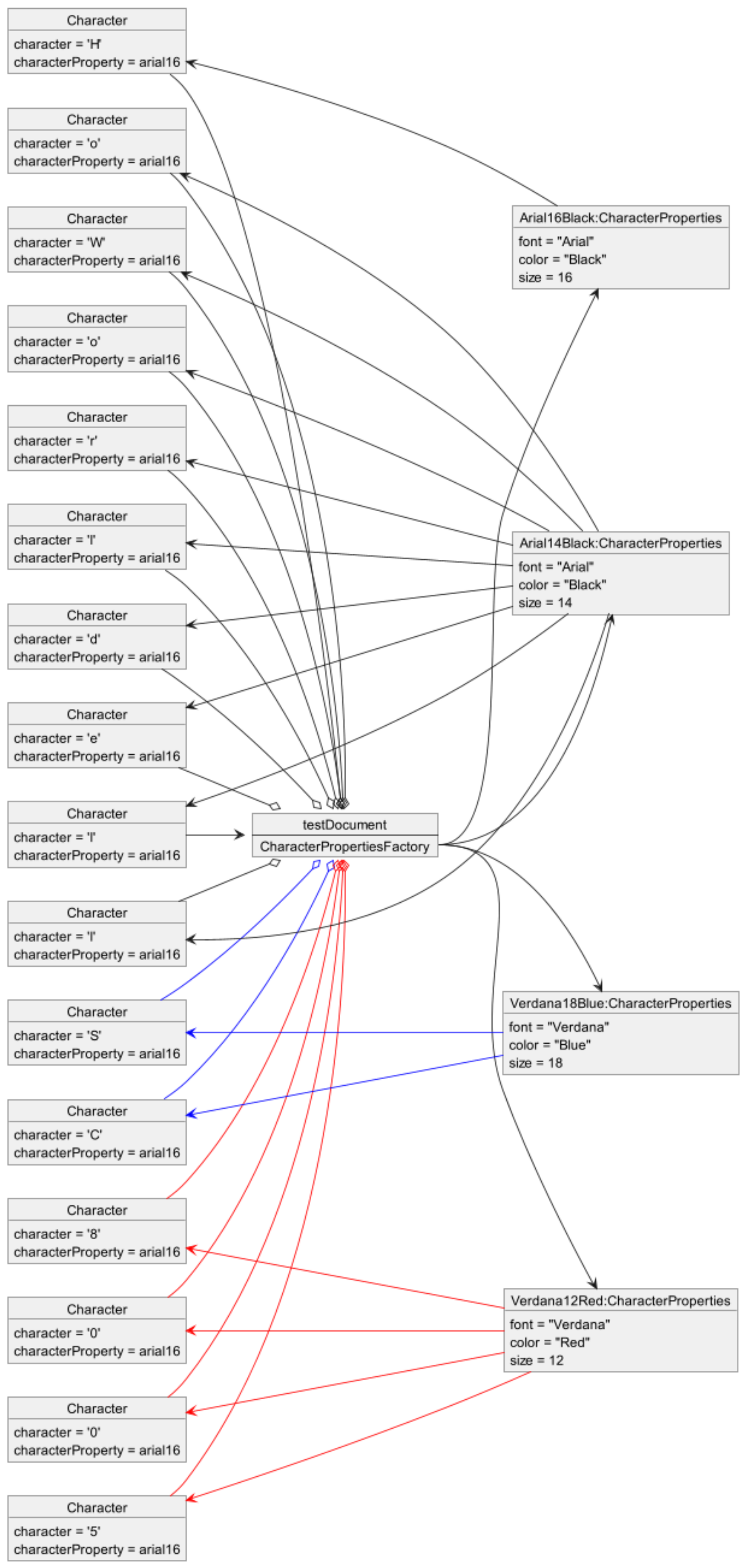
UML Section

Class Diagram

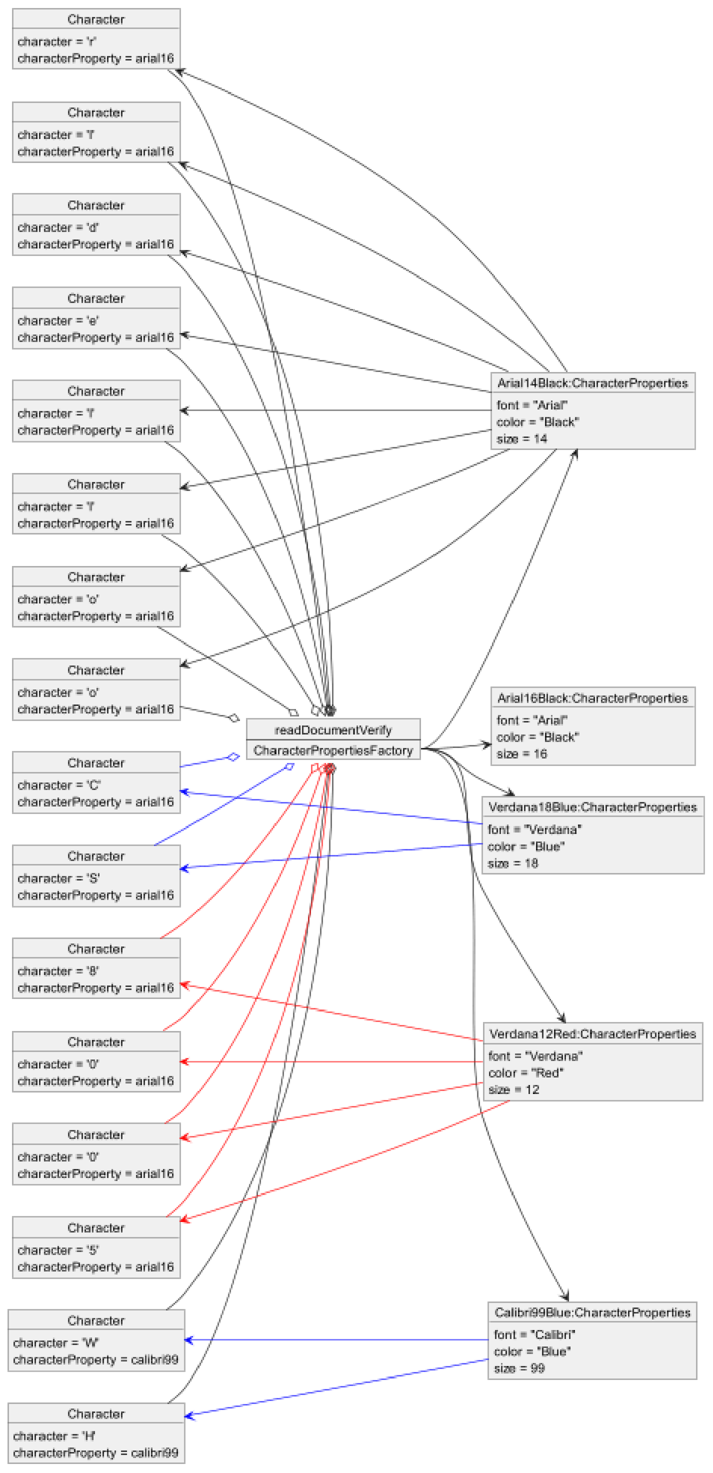


Document
characters: List<Character>
propertiesFactory: CharacterPropertiesFactory
Document():
loadFromFile(String): void
editCharacterProperties(int, String, String, int): void
saveToFile(String): void
addCharacter(char, String, String, int): void
Driver
Driver():
main(String[]): void
Character
character: char
characterProperty: CharacterProperties
Character(char, CharacterProperties):
setCharacter(char): void
getCharacter(): char
getColor(): String
getFont(): String
getSize(): int
setCharacterProperty(CharacterProperties): void
CharacterProperties
font: String
color: String
size: int
CharacterProperties(String, String, int):
getFont(): String
getColor(): String
getSize(): int
CharacterPropertiesFactory
flyweights: Map<String, CharacterProperties>
CharacterPropertiesFactory():
sizeOfMap(): int
setAndRetrieveFlyweightCharacterProperties(String, String, int): CharacterProperties
1
1
0..*
1
1
1..*
1

Part 1 Object diagram


Character
character = 'H'
characterProperty = arial16
Character
character = 'o'
characterProperty = arial16
Character
character = 'W'
characterProperty = arial16
Character
character = 'o'
characterProperty = arial16
Character
character = 'r'
characterProperty = arial16
Character
character = 'l'
characterProperty = arial16
Character
character = 'd'
characterProperty = arial16
Character
character = 'e'
characterProperty = arial16
Character
character = 'l'
characterProperty = arial16
Character
character = 'l'
characterProperty = arial16
Character
character = 'S'
characterProperty = arial16
Character
character = 'C'
characterProperty = arial16
Character
character = '8'
characterProperty = arial16
Character
character = '0'
characterProperty = arial16
Character
character = '0'
characterProperty = arial16
Character
character = '5'
characterProperty = arial16
Arial16Black:CharacterProperties
font = "Arial"
color = "Black"
size = 16
Arial14Black:CharacterProperties
font = "Arial"
color = "Black"
size = 14
testDocument
CharacterPropertiesFactory
Verdana18Blue:CharacterProperties
font = "Verdana"
color = "Blue"
size = 18
Verdana12Red:CharacterProperties
font = "Verdana"
color = "Red"
size = 12

Part 2 Object Diagram


Character
character = 'r'
characterProperty = arial16
Character
character = 'l'
characterProperty = arial16
Character
character = 'd'
characterProperty = arial16
Character
character = 'e'
characterProperty = arial16
Character
character = 'l'
characterProperty = arial16
Character
character = 'l'
characterProperty = arial16
Character
character = 'o'
characterProperty = arial16
Character
character = 'o'
characterProperty = arial16
Character
character = 'C'
characterProperty = arial16
Character
character = 'S'
characterProperty = arial16
Character
character = '8'
characterProperty = arial16
Character
character = '0'
characterProperty = arial16
Character
character = '0'
characterProperty = arial16
Character
character = '5'
characterProperty = arial16
Character
character = 'W'
characterProperty = calibri99
Character
character = 'H'
characterProperty = calibri99
readDocumentVerify
CharacterPropertiesFactory
Arial14Black:CharacterProperties
font = "Arial"
color = "Black"
size = 14
Arial16Black:CharacterProperties
font = "Arial"
color = "Black"
size = 16
Verdana18Blue:CharacterProperties
font = "Verdana"
color = "Blue"
size = 18
Verdana12Red:CharacterProperties
font = "Verdana"
color = "Red"
size = 12
Calibri99Blue:CharacterProperties
font = "Calibri"
color = "Blue"
size = 99