Homework-3

SDA

Zunaira Abdul Aziz
BSSE23058
Section A

Overview

Starbuzz Coffee is a simple Java application that demonstrates the Decorator Design Pattern. It allows users to create different types of beverages and add various condiments, calculating the total cost for each order. The application showcases object-oriented programming principles, including abstraction and inheritance.

Features

Create different types of beverages (e.g., Dark Roast, Espresso, Decaf, House Blend).

Add condiments (e.g., Milk, Mocha, Soy, Whip) to beverages.

Calculate and display the total cost of the beverage with added condiments.

Display a detailed description of the ordered beverage.

Code Structure

The project consists of the following classes:

Beverage: An abstract class representing the base beverage.

CondimentDecorator: An abstract class extending Beverage, representing condiments.

DarkRoast, Decaf, Espresso, HouseBlend: Concrete classes that extend Beverage, each representing a specific type of coffee.

Milk, Mocha, Soy, Whip: Concrete classes that extend CondimentDecorator, representing different condiments.

StarbuzzCoffee: The main class containing the main method to run the application..

Code

```
public abstract class Beverage {// Beverage is an abstract getDescription is
already class with the two methods
                                // implemented for us, but we getDescription()
and cost().
    String description = "Unknown Beverage";

    public String getDescription() {
        return description;
    }

    public abstract double cost();
}
```

```
public abstract class CondimentDecorator extends Beverage {// we need to be
interchangeable with a Beverage, so we
                                                            // extend the
Beverage class.
    @Override
    public abstract String getDescription();// going to require that the
condiment decorators all reimplement the
                                            // getDescription() method.
}

public class StarbuzzCoffee {
    public static void main(String[] args) {
        Beverage beverage1 = new DarkRoast();
        beverage1 = new Milk(beverage1);
        beverage1 = new Mocha(beverage1);
        System.out.println(beverage1.getDescription() + " $" +
beverage1.cost());

        Beverage beverage2 = new HouseBlend();
        beverage2 = new Soy(beverage2);
        beverage2 = new Whip(beverage2);
        System.out.println(beverage2.getDescription() + " $" +
beverage2.cost());

        Beverage beverage3 = new Espresso();
        beverage3 = new Whip(beverage3);
        System.out.println(beverage3.getDescription() + " $" +
beverage3.cost());

        Beverage beverage4 = new Decaf();
        beverage4 = new Mocha(beverage4);
        System.out.println(beverage4.getDescription() + " $" +
beverage4.cost());
    }
}
```

```
public class DarkRoast extends Beverage {
    public DarkRoast() { // set the appropriate description, “House Blend
Coffee,” and then return the
                         // correct cost:
        description = "Dark Roast COffee";
    }

    @Override
    public double cost() {
        return 1.00;
    }
}

public class Decaf extends Beverage {
    public Decaf() { // set the appropriate description, “House Blend Coffee,”
and then return the
                    // correct cost:
        description = "Decaf COffee";
    }

    @Override
    public double cost() {
        return 1.50;
    }
}

public class Espresso extends Beverage {

    public Espresso() {// To take care of the description, we set this in the
constructor for the
                       // class. Remember the description instance variable is
inherited from Beverage.
        description = "Espresso";
    }

    @Override
    public double cost() {
        return 2.00;
    }
}

public class HouseBlend extends Beverage {
    public HouseBlend() { // set the appropriate description, “House Blend
Coffee,” and then return the
                          // correct cost:
        description = "House Blend Coffee";
    }
```

```
    @Override
    public double cost() {
        return 0.50;
    }
}

public class Milk extends CondimentDecorator {
    Beverage beverage; // An instance variable to hold the beverage we are
wrapping.

    public Milk(Beverage beverage) {
        this.beverage = beverage; // A way to set this instance variable to
the object we are wrapping. Here,
                                  // we're going to to pass the beverage we're
wrapping to the decorator's
                                  // constructor.
    }

    @Override
    public String getDescription() {
        return beverage.getDescription() + ", Milk";
    }

    @Override
    public double cost() {
        return .20 + beverage.cost();
    }// we need to compute the cost of our beverage with Mocha. First, we
delegate
     // the call to the object we're decorating, so that it can compute the
cost;
}

public class Mocha extends CondimentDecorator {
    Beverage beverage; // An instance variable to hold the beverage we are
wrapping.

    public Mocha(Beverage beverage) {
        this.beverage = beverage; // A way to set this instance variable to
the object we are wrapping. Here,
                                  // we're going to to pass the beverage we're
wrapping to the decorator's
                                  // constructor.
    }

    @Override
    public String getDescription() {
        return beverage.getDescription() + ", Mocha";
    }
```

```
    @Override
    public double cost() {
        return .20 + beverage.cost();
    }// we need to compute the cost of our beverage with Mocha. First, we
delegate
     // the call to the object we're decorating, so that it can compute the
cost;
     // then, we add the cost of Mocha to the result.
}

public class Soy extends CondimentDecorator {
    Beverage beverage; // An instance variable to hold the beverage we are
wrapping.

    public Soy(Beverage beverage) {
        this.beverage = beverage; // A way to set this instance variable to
the object we are wrapping. Here,
                                  // we're going to to pass the beverage we're
wrapping to the decorator's
                                  // constructor.
    }

    @Override
    public String getDescription() {
        return beverage.getDescription() + ", Soy";
    }

    @Override
    public double cost() {
        return .20 + beverage.cost();
    }// we need to compute the cost of our beverage with Mocha. First, we
delegate
     // the call to the object we're decorating, so that it can compute the
cost;
}

public class Whip extends CondimentDecorator {
    Beverage beverage; // An instance variable to hold the beverage we are
wrapping.

    public Whip(Beverage beverage) {
        this.beverage = beverage; // A way to set this instance variable to
the object we are wrapping. Here,
                                  // we're going to to pass the beverage we're
wrapping to the decorator's
                                  // constructor.
    }
```

```java
    @Override
    public String getDescription() {
        return beverage.getDescription() + ", Whip";
    }

    @Override
    public double cost() {
        return .20 + beverage.cost();
    }// we need to compute the cost of our beverage with Mocha. First, we
delegate
     // the call to the object we're decorating, so that it can compute the
cost;
}
```

```
zuni_2004@DESKTOP-16K3I1B:/mnt/d/University/SDA/SDA Homework/Homework-3$ javac *.java
zuni_2004@DESKTOP-16K3I1B:/mnt/d/University/SDA/SDA Homework/Homework-3$ java StarbuzzCoffee
Dark Roast COffee, Milk, Mocha $1.4
House Blend Coffee, Soy, Whip $0.8999999999999999
Espresso, Whip $2.2
Decaf COffee, Mocha $1.7
zuni_2004@DESKTOP-16K3I1B:/mnt/d/University/SDA/SDA Homework/Homework-3$
```