آزمایشگاه مهندسی نرم‌افزار

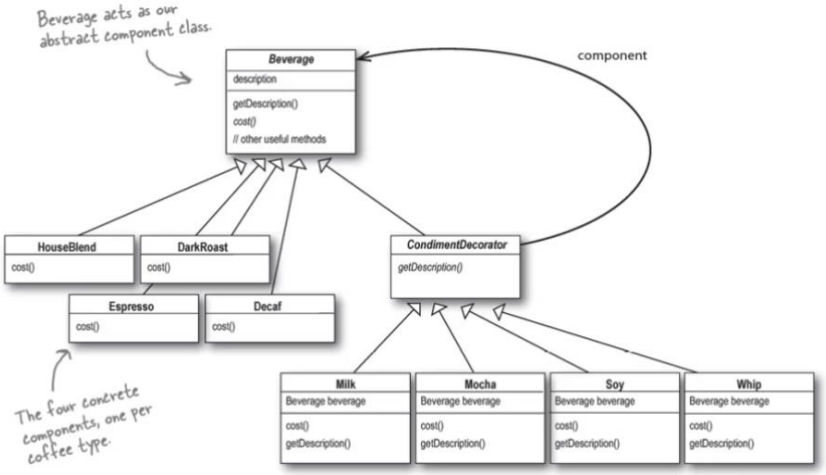
اعضای گروه :

آروین سمیعی ۹۵۱۰۵۶۴۶

علی سلمانی ۹۵۱۰۱۷۰۶

ترم اول ۰۰-۹۹

آزمایش دوم



(a) Solution Class Diagram

(b) Price List

Figure: Problem

الگوی decorator:

تست‌ها که از قبل به ما داده شده است. پس اولین مرحله‌ی TDD انجام شده‌است.

طبق نمودار بالا عمل میکنم که در آن condiment ها decorator هستند.

ابتدا اینترفیس Beverage را ایجاد میکنم که دارای دو متد getDescription و cost است.

```java
package decorator;

public interface Beverage {

    String getDescription();
    double cost();
}
```

سپس کلاس‌های قهوه را ایجاد میکنم:

```java
package decorator;

public class DarkRoast  implements Beverage{
    public String getDescription() { return "Delicious DarkRoast"; }

    public double cost() { return 0.99; }
}
```

```java
package decorator;

public class Decaf  implements Beverage{
    public String getDescription() { return "Delicious Decaf"; }

    public double cost() { return 1.99; }
}
```

```java
package decorator;

public class Espresso  implements Beverage{
    public String getDescription() { return "Delicious Espresso"; }

    public double cost() { return 1.99; }
}
```

```java
package decorator;

public class HouseBlend implements Beverage{
    public String getDescription() { return "Delicious HouseBlend"; }

    public double cost() { return 0.89; }
}
```

سپس کلاس‌های decorator را ایجاد میکنم. اول یک کلاس abstract به اسم CondimentDecorator ایجاد میکنم که سوپرکلاس تمام decorator هاست.

```java
package decorator;

public abstract class condimentDecorator implements Beverage{

    Beverage beverage;

    public condimentDecorator(Beverage beverage) { this.beverage = beverage; }

    public Beverage getBeverage() { return beverage; }

    public String getDescription(){
        return beverage.getDescription();
    }

    public double cost() { return beverage.cost(); }

}
```

حال انواع flavor ها را ایجاد میکنم:

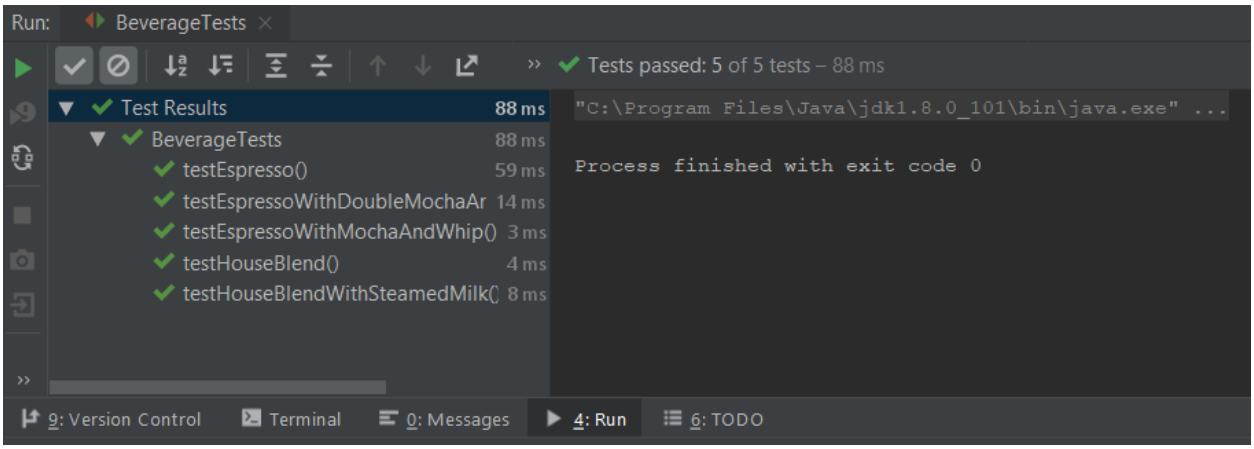
```java
package decorator;

public class Mocha extends condimentDecorator {
    public Mocha(Beverage beverage) { super(beverage); }

    @Override
    public String getDescription() { return super.getDescription() + " " +  this.addedGetDescription(); }

    private String addedGetDescription() { return "with mocha"; }

    @Override
    public double cost() { return super.cost() + addedCost(); }

    private double addedCost() { return  0.2; }
}
```

```java
package decorator;

public class Soy extends condimentDecorator {
    public Soy(Beverage beverage) {
        super(beverage);
    }

    @Override
    public String getDescription() { return super.getDescription() + " " +  this.addedGetDescription(); }

    private String addedGetDescription() { return "with soy"; }

    @Override
    public double cost() { return super.cost() + addedCost(); }

    private double addedCost() { return  0.15; }
}
```

```java
package decorator;

public class SteamedMilk extends condimentDecorator {
    public SteamedMilk(Beverage beverage) { super(beverage); }

    @Override
    public String getDescription() { return super.getDescription() + " " +  this.addedGetDescription(); }

    private String addedGetDescription() { return "with milk"; }

    @Override
    public double cost() { return super.cost() + addedCost(); }

    private double addedCost() { return  0.1; }
}
```

```java
package decorator;

public class Whip extends condimentDecorator {
    public Whip(Beverage beverage) { super(beverage); }

    @Override
    public String getDescription() { return super.getDescription() + " " +  this.addedGetDescription(); }

    private String addedGetDescription() { return "with whip"; }

    @Override
    public double cost() { return super.cost() + addedCost(); }

    private double addedCost() { return  0.1; }
}
```

که در هرکدام از آن‌ها یک پس پردازش انجام می‌شود و به رفتارهای سایر اجزا addedGetDescription و addedCost را اضافه میکند.



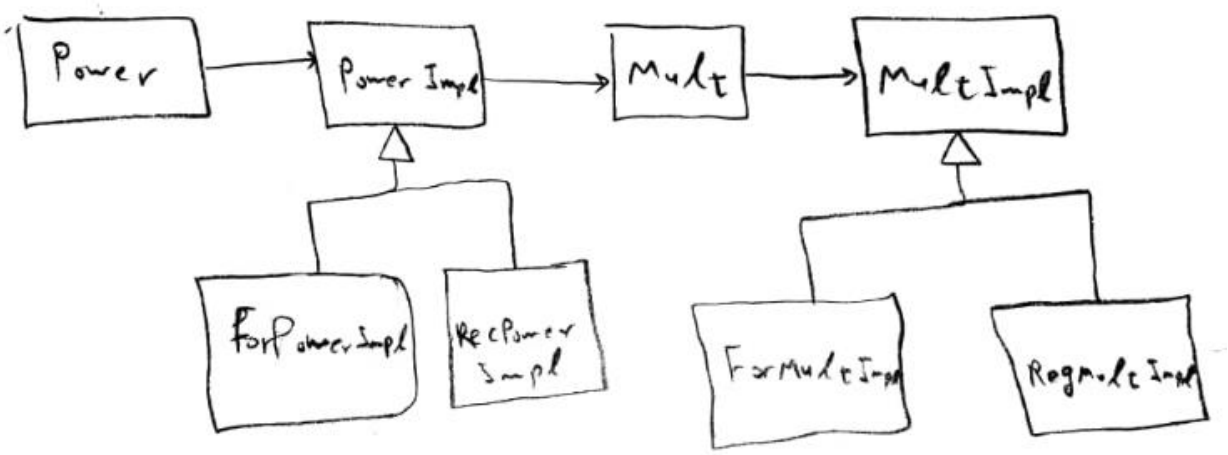
تست‌ها را اجرا میکنم و همگی پاس می‌شوند. پس مرحله‌ی دوم TDD هم انجام شد. نیاز به ریفکتورینگ هم ندارد پس کار انجام شده است.

الگوی bridge:

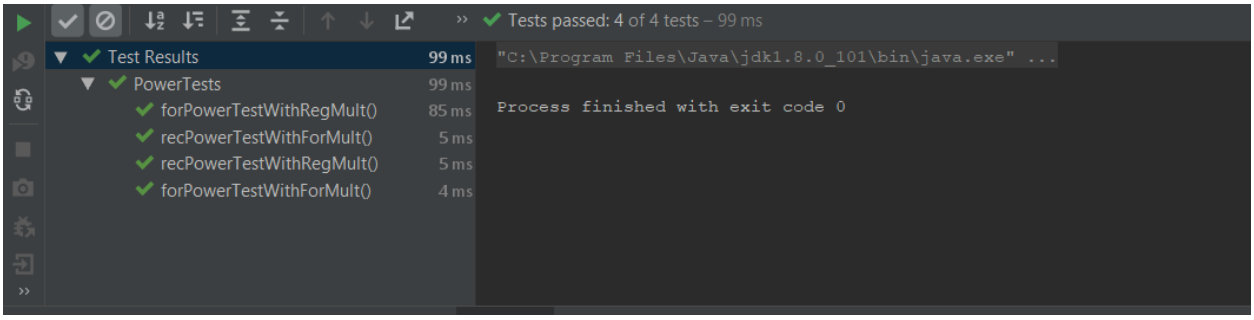
ابتدا تست‌ها را مینویسم:

```java
package tests;

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

public class PowerTests {

    @Test
    public void recPowerTestWithForMult(){
        Power power = new Power(new RecPowerImpl(new Mult(new ForMultImpl())));
        int result = power.power(2, 3);
        assertEquals( expected: 8, result);
    }

    @Test
    public void forPowerTestWithForMult(){
        Power power = new Power(new RecPowerImpl(new Mult(new ForMultImpl())));
        int result = power.power(2, 3);
        assertEquals( expected: 8, result);
    }

    @Test
    public void recPowerTestWithRegMult(){
        Power power = new Power(new RecPowerImpl(new Mult(new RegMultImpl())));
        int result = power.power(2, 3);
        assertEquals( expected: 8, result);
    }

    @Test
    public void forPowerTestWithRegMult(){
        Power power = new Power(new RecPowerImpl(new Mult(new RegMultImpl())));
        int result = power.power(2, 3);
        assertEquals( expected: 8, result);
    }

}
```

سپس طبق نمودار زیر کلاس‌ها را طراحی میکنم. پیاده‌سازی آنها در پوشه‌ی جداگانه‌ای قرار داده شده است.



سپس تست‌ها را ران میکنم و همگی پاس می‌شوند:



پس مرحله‌ی دوم TDD هم اجرا شد. کد نیاز به refactoring ندارد پس کار تمام میشود