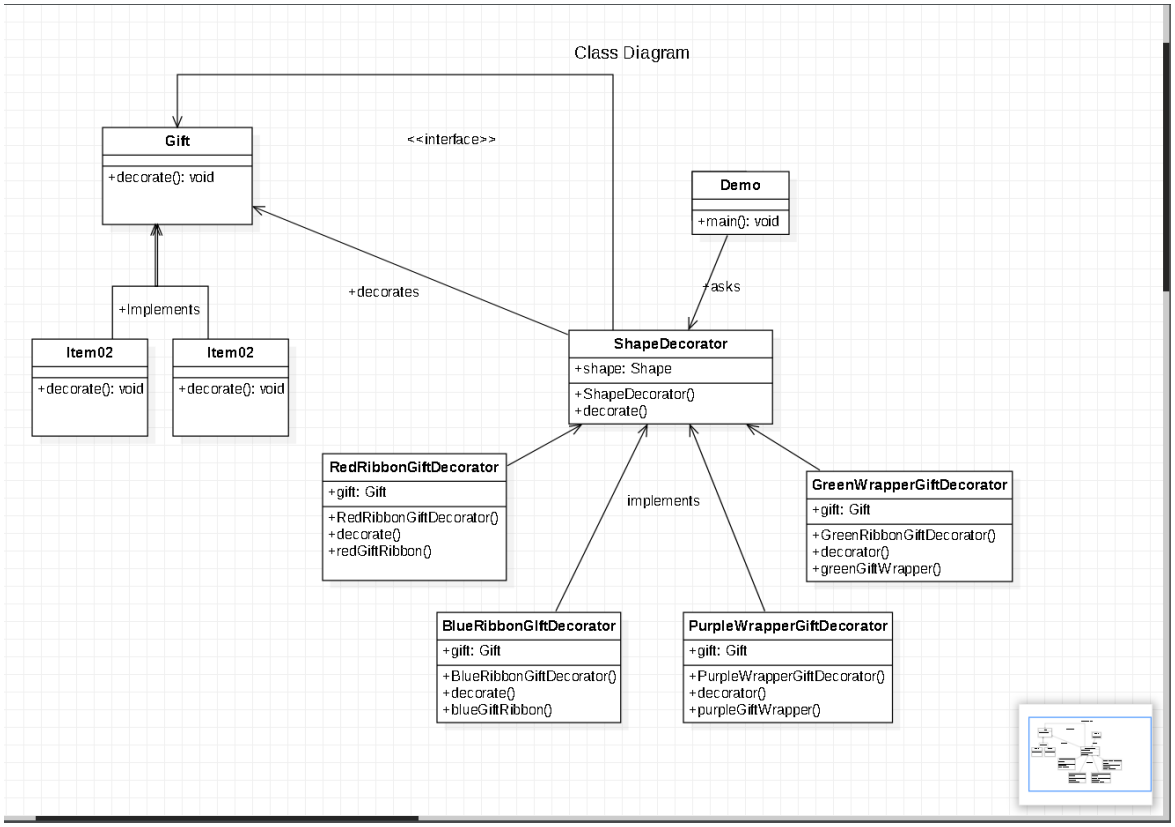
Lab Session 03

Explore Structural Design Pattern to add functionality to an object dynamically.

Exercise

1. Suppose we are selling a gift item. Once a user selects a gift item, there can be multiple ways just to decorate that gift item with a red or a blue ribbon, purple or green gift wrap, etc. Draw a class diagram for this scenario using a decorator pattern. Also implement an interface for gift items and use decorator patterns to execute different combinations possible using Java. Also attach print outs.



Gift.java

```java
package lab03Q1;

public interface Gift {
    void decorate();
}
```

GiftDecorator.java

```java
package lab03Q1;

public abstract class GiftDecorator implements Gift {
    protected Gift decoratedGift;
    public GiftDecorator(Gift decoratedGift){
    this.decoratedGift = decoratedGift;
    }
    public void decorate(){
    decoratedGift.decorate();

}
}
```

Item01.java

```java
package lab03Q1;

public class Item01 implements Gift {

    @Override
    public void decorate() {
        // TODO Auto-generated method stub
        System.out.println("Gift: Item01");
    }

}
```

Item02.java

```java
package lab03Q1;

public class Item02 implements Gift {

    @Override
    public void decorate() {
        // TODO Auto-generated method stub
        System.out.println("Gift: Item02");
    }

}
```

RedRibbonGiftDecorator.java

```java
package lab03Q1;

public class RedRibbonGiftDecorator extends GiftDecorator{
       public RedRibbonGiftDecorator(Gift decoratedGift) {
           super(decoratedGift);
           }
           @Override
           public void decorate() {
           decoratedGift.decorate();
           redRibbon(decoratedGift);
           }
           private void redRibbon(Gift decoratedGift){
               System.out.println("Ribbon color: RedRibbon");
               }
           }
```

BlueRibbonGiftDecorator.java

```java
package lab03Q1;

public class BlueRibbonGiftDecorator extends GiftDecorator{

    public BlueRibbonGiftDecorator(Gift decoratedGift) {
        super(decoratedGift);
        }
        @Override
        public void decorate() {
        decoratedGift.decorate();
        blueRibbon(decoratedGift);
        }
        private void blueRibbon(Gift decoratedGift){
            System.out.println("Ribbon color: BlueRibbon");
        }

        }
```

PurpleWrapGiftDecorator.java

```java
package lab03Q1;

public class PurpleGiftWrapGiftDecorator extends GiftDecorator{
    public PurpleGiftWrapGiftDecorator(Gift decoratedGift) {
        super(decoratedGift);
        }
        @Override
        public void decorate() {
        decoratedGift.decorate();
        purpleGiftWrap(decoratedGift);
        }
        private void purpleGiftWrap(Gift decoratedGift){
            System.out.println("Gift wrap color: purple wrap");
        }
        }
```

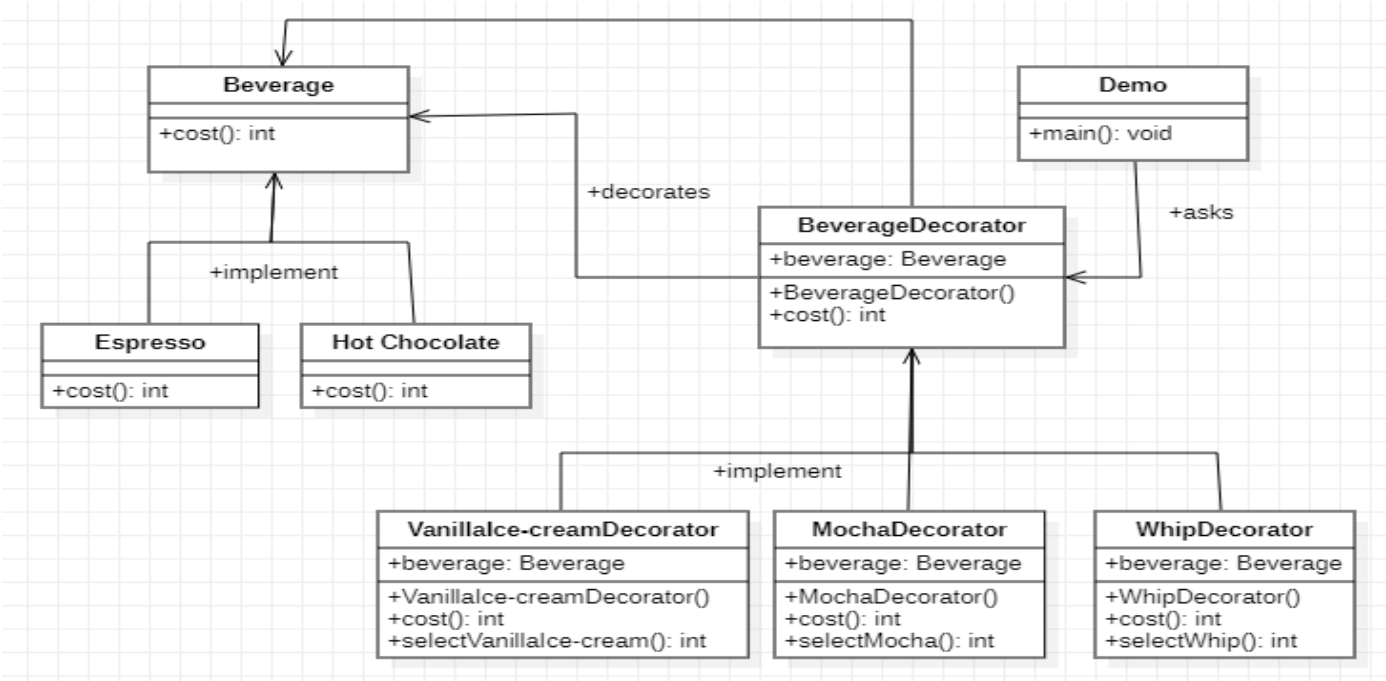
GreenWrapGiftDecorator.java

```java
package lab03Q1;

public class GreenGiftWrapGiftDecorator extends GiftDecorator {
    public GreenGiftWrapGiftDecorator(Gift decoratedGift) {
        super(decoratedGift);
        }
        @Override
        public void decorate() {
        decoratedGift.decorate();
        greenGiftWrap(decoratedGift);
        }
        private void greenGiftWrap(Gift decoratedGift){
            System.out.println("Gift wrap color: green wrap");
            }
        }
```

Output:

```
Gift item01 with no decoration
Gift: Item01

Gift item01 wrapped
Gift: Item01
Gift wrap color: purple wrap

Gift item02 Ribboned
Gift: Item02
Ribbon color: BlueRibbon

Gift item02 Wrapped
Gift: Item02
Gift wrap color: green wrap

Gift item02 Ribboned
Gift: Item02
Gift wrap color: green wrap
```

2. Suppose you are planning to start a side business of a Coffee Shop. To automate the order processing, you intend to seek help from the OO decorator pattern. Create a Beverage class and decorate it with condiments at run time and estimate the overall cost of Beverage at the end. For example, a customer wants Espresso with mocha and whip or he wants Hot Chocolate with Vanilla ice cream. Draw a class diagram for the given situation and implement it in Java. Also attach print outs.



Step 1: Beverage.java interface

Beverage.java BeverageDec...

```java
package Lab3_q2;

public interface Beverage {
    int cost();

}
```

Step 2: Create concrete classes implementing the interface Beverage

Espresso.java HotChocolate.java MochaDecor

```java
package Lab3_q2;

public class Espresso implements Beverage {

    @Override
    public int cost() {
        int price = 100;
        return price;
    }

}
```

*HotChocolate.java MochaDecorator.java WhipDe

```java
package Lab3_q2;

public class HotChocolate implements Beverage {
    @Override
    public int cost() {
        int price = 150;
        return price;
    }
}
```

Step 3: Create the abstract decorator class implementing the Beverage interface

*BeverageDecorator.java

```
1  package Lab3_q2;
2
3  public abstract class BeverageDecorator implements Beverage {
4      protected Beverage decoratedBeverage;
5      public BeverageDecorator(Beverage decoratedBeverage) {
6          this.decoratedBeverage = decoratedBeverage;
7      }
8      @Override
9      public int cost() {
10         return decoratedBeverage.cost();
11     }
12 }
```

Step 4: Create concrete class implementing the BeverageDecorator abstract class

*MochaDecorator.java

```
1  package Lab3_q2;
2
3  public class MochaDecorator extends BeverageDecorator {
4      public MochaDecorator(Beverage decoratedBeverage) {
5          super(decoratedBeverage);
6      }
7      public int cost() {
8          return decoratedBeverage.cost()+selectMocha(decoratedBeverage);
9      }
10     private int selectMocha(Beverage decoratedBeverage) {
11         int price=50;
12         System.out.println("Mocha cost:"+price);
13         return price;
14     }
15 }
```

*WhipDecorator.java

```
1  package Lab3_q2;
2
3  public class WhipDecorator extends BeverageDecorator {
4          public WhipDecorator(Beverage decoratedBeverage) {
5              super(decoratedBeverage);
6          }
7          public int cost() {
8              return decoratedBeverage.cost()+selectWhip(decoratedBeverage);
9          }
10         private int selectWhip(Beverage decoratedBeverage) {
11             int price=100;
12             System.out.println("Whip cost:"+price);
13             return price;
14         }
15     }
```

*VanillaIceCreamDecorator.java

```java
package Lab3_q2;

public class VanillaIceCreamDecorator extends BeverageDecorator {
    public VanillaIceCreamDecorator(Beverage decoratedBeverage) {
        super(decoratedBeverage);
    }
    public int cost() {
        return decoratedBeverage.cost()+selectVanillaIceCream(decoratedBeverage);
    }
    private int selectVanillaIceCream(Beverage decoratedBeverage) {
        int price=80;
        System.out.println("Vanilla Ice-cream cost:"+price);
        return price;
    }
}
```

*Demo.java

```java
package Lab3_q2;

public class Demo {
    public static void main(String[] args) {
        Beverage espresso= new Espresso();
        Beverage mochaEspresso = new MochaDecorator(new Espresso());
        Beverage whipEspresso = new WhipDecorator(new Espresso());
        Beverage mwEspresso = new WhipDecorator(new MochaDecorator(new Espresso()));
        Beverage hotChocolate= new HotChocolate();
        Beverage vanillaHotChocolate = new VanillaIceCreamDecorator(new Espresso());

        System.out.println("Espresso cost:"+espresso.cost());
        System.out.println("Espresso with Mocha cost:" + mochaEspresso.cost());
        System.out.println("\n");
        System.out.println("Espresso cost:"+espresso.cost());
        System.out.println("Espresso with Whip cost:" + whipEspresso.cost());
        System.out.println("\n");
        System.out.println("Espresso cost:"+espresso.cost());
        System.out.println("Espresso with Mocha & Whip cost:" + mwEspresso.cost());
        System.out.println("\n");
        System.out.println("Hot Chocolate cost:"+hotChocolate.cost());
        System.out.println("Hot Chocolate with Vanilla Ice-cream cost:" + vanillaHotChocolate.cost());

    }
}
```

Step 5: Create the main class demonstrating the decorator design pattern

Output:

```
Espresso cost:100
Mocha cost:50
Espresso with Mocha cost:150


Espresso cost:100
Whip cost:100
Espresso with Whip cost:200


Espresso cost:100
Mocha cost:50
Whip cost:100
Espresso with Mocha & Whip cost:250


Hot Chocolate cost:150
Vanilla Ice-cream cost:80
Hot Chocolate with Vanilla Ice-cream cost:180
```