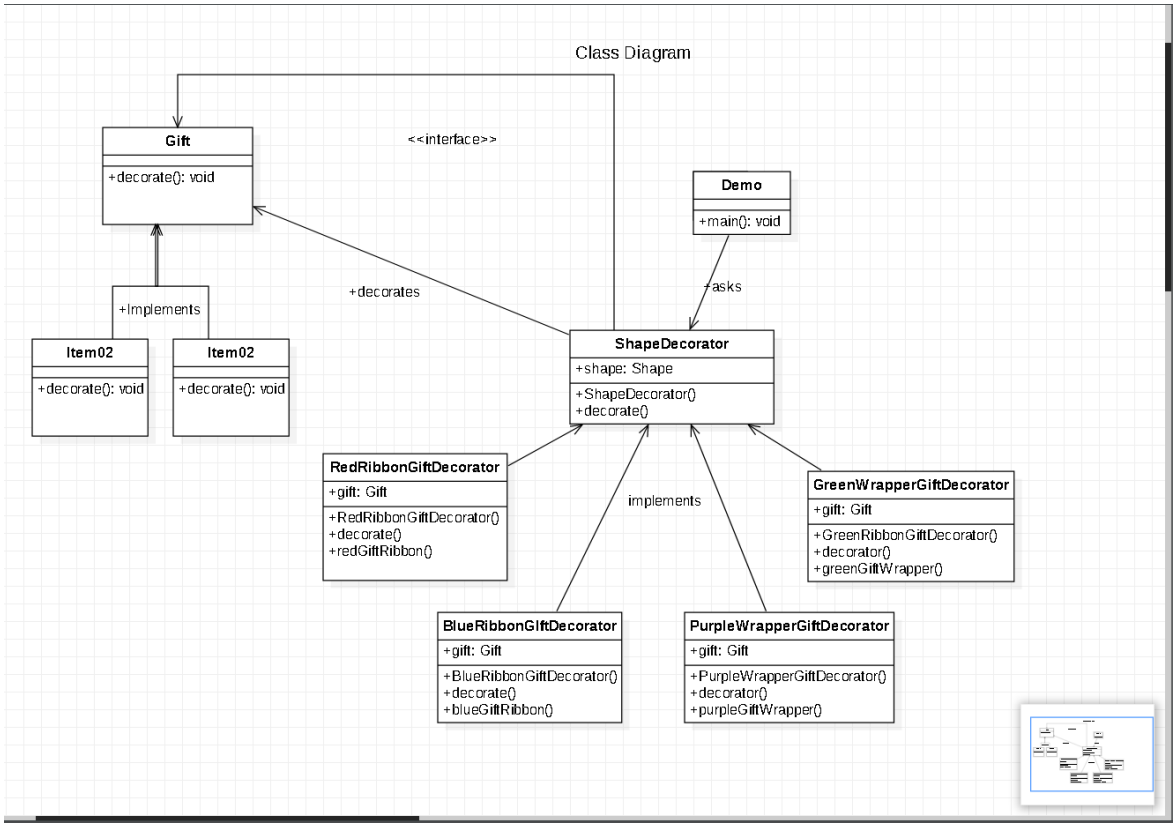
Lab Session 03

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");
            }
        }

```