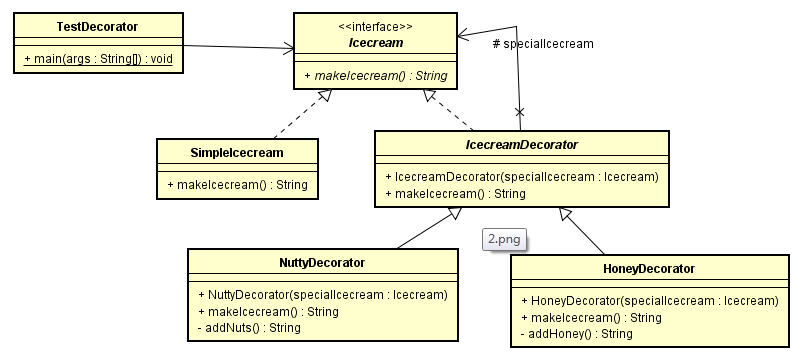
# 面向对象系统分析与设计课后作业03

**hw01.**

Following given example is an class diagram of decorator design pattern. Icecream is a classic example for decorator design pattern. You create a basic icecream and then add toppings to it as you prefer. The added toppings change the taste of the basic icecream.(下面是类图，请根据类图编写代码)。   
package iceCream;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public abstract interface IceCream {  
 public String makeIceCream();  
}

package iceCream;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public class SimpleIceCream implements IceCream{  
 @Override  
 public String makeIceCream() {  
 System.*out*.println("simple ice cream");  
 return null;  
 }  
}

package iceCream;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public class IceCreamDecorator implements IceCream{  
 private IceCream iceCream;  
  
 public IceCreamDecorator(IceCream iceCream) {  
 this.iceCream = iceCream;  
 }  
  
 public IceCreamDecorator() {  
  
 }  
  
 @Override  
 public String makeIceCream() {  
 iceCream.makeIceCream();  
 return null;  
 }  
}

package iceCream;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public class HoneyDecorator extends IceCreamDecorator{  
 private IceCreamDecorator iceCreamDecorator;  
 public HoneyDecorator(IceCreamDecorator iceCreamDecorator) {  
 super(iceCreamDecorator);  
 }  
  
 public HoneyDecorator() {  
 super();  
 }  
  
 public String addHoney() {  
 System.*out*.println("addHoney");  
 return null;  
 }  
  
 @Override  
 public String makeIceCream() {  
 addHoney();  
 System.*out*.println("making honey ice cream");  
 return null;  
 }  
}

package iceCream;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public class NuttyDecorator extends IceCreamDecorator{  
 private IceCream iceCream;  
  
 public NuttyDecorator(IceCream iceCream) {  
 super(iceCream);  
 this.iceCream = iceCream;  
 }  
  
 public NuttyDecorator() {  
 super();  
 }  
  
  
 public String addNuts() {  
 System.*out*.println("addNuts...");  
 return null;  
 }  
  
 @Override  
 public String makeIceCream() {  
 addNuts();  
 System.*out*.println("making nutty ice cream");  
 return null;  
 }  
}

package iceCream;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public class TestDecorator {  
 public static void main(String[] args) {  
 new IceCreamDecorator(new HoneyDecorator()).makeIceCream();  
 new IceCreamDecorator(new NuttyDecorator()).makeIceCream();  
 }  
}

**hw02**.

We're going to create a Shape interface and concrete classes implementing the Shape interface. We will then create an abstract decorator class ShapeDecorator implementing the Shape interface and having Shape object as its instance variable.

RedShapeDecorator is concrete class implementing ShapeDecorator.

TestShape, our demo class will use RedShapeDecorator to decorate Shape objects..  
Codes:

public class testShape {

public static void main(String[] args) {

Shape circle = new Circle();

Shape redCircle = new RedShapeDecorator(new Circle());

Shape redRectangle = new RedShapeDecorator(new Rectangle());

System.out.println("Circle with normal border");

circle.draw();

System.out.println("\nCircle of red border");

redCircle.draw();

System.out.println("\nRectangle of red border");

redRectangle.draw();

}

}

interface Shape {

void draw();

}

class Rectangle implements Shape {

@Override

public void draw() {

System.out.println("Shape: Rectangle");

}

}

class Circle implements Shape {

@Override

public void draw() {

System.out.println("Shape: Circle");

}

}

abstract class ShapeDecorator implements Shape {

protected Shape decoratedShape;

public ShapeDecorator(Shape decoratedShape){

this.decoratedShape = decoratedShape;

}

public void draw(){

decoratedShape.draw();

}

}

class RedShapeDecorator extends ShapeDecorator {

public RedShapeDecorator(Shape decoratedShape) {

super(decoratedShape);

}

@Override

public void draw() {

decoratedShape.draw();

setRedBorder(decoratedShape);

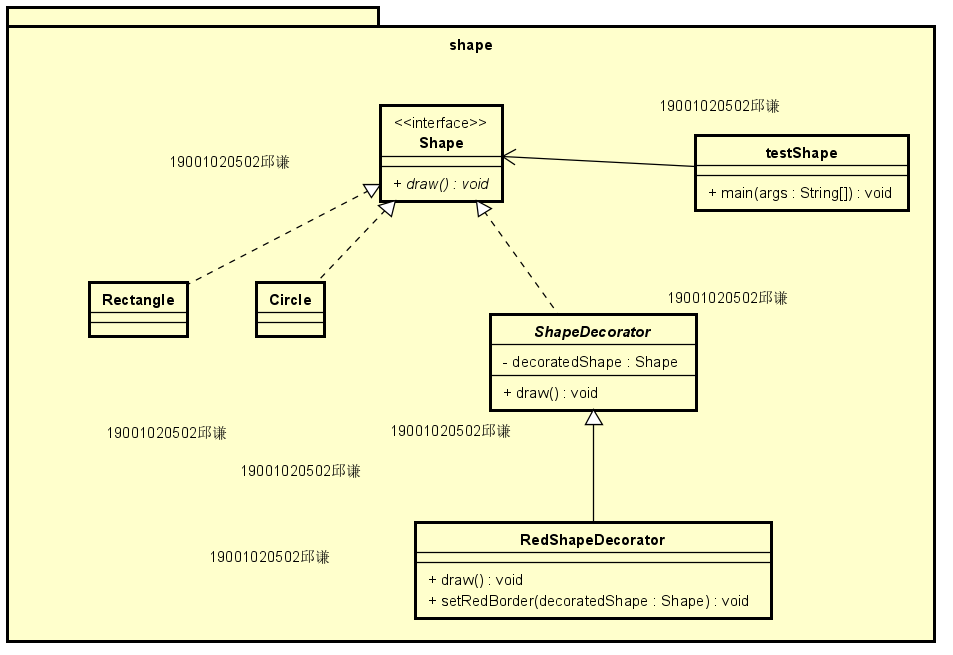
}

private void setRedBorder(Shape decoratedShape){

System.out.println("Border Color: Red");

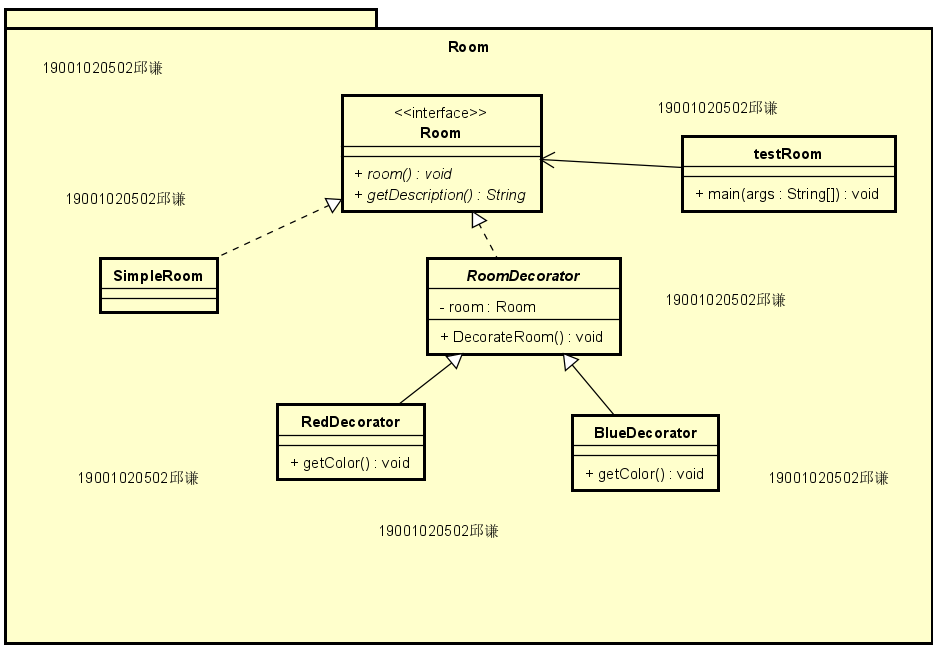
}

}



**hw03.**

To decorate a simple room with color and curtains, try to provide a solution which contains class diagrams and codes based on your class diagram.(完成类图，请根据类图编写代码）。



package room;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public interface Room {  
 public void room();  
 public String getDescription();  
}

package room;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public class SimpleRoom implements Room{  
  
 @Override  
 public void room() {  
  
 }  
  
 @Override  
 public String getDescription() {  
 return "simple room with curtain";  
 }  
}

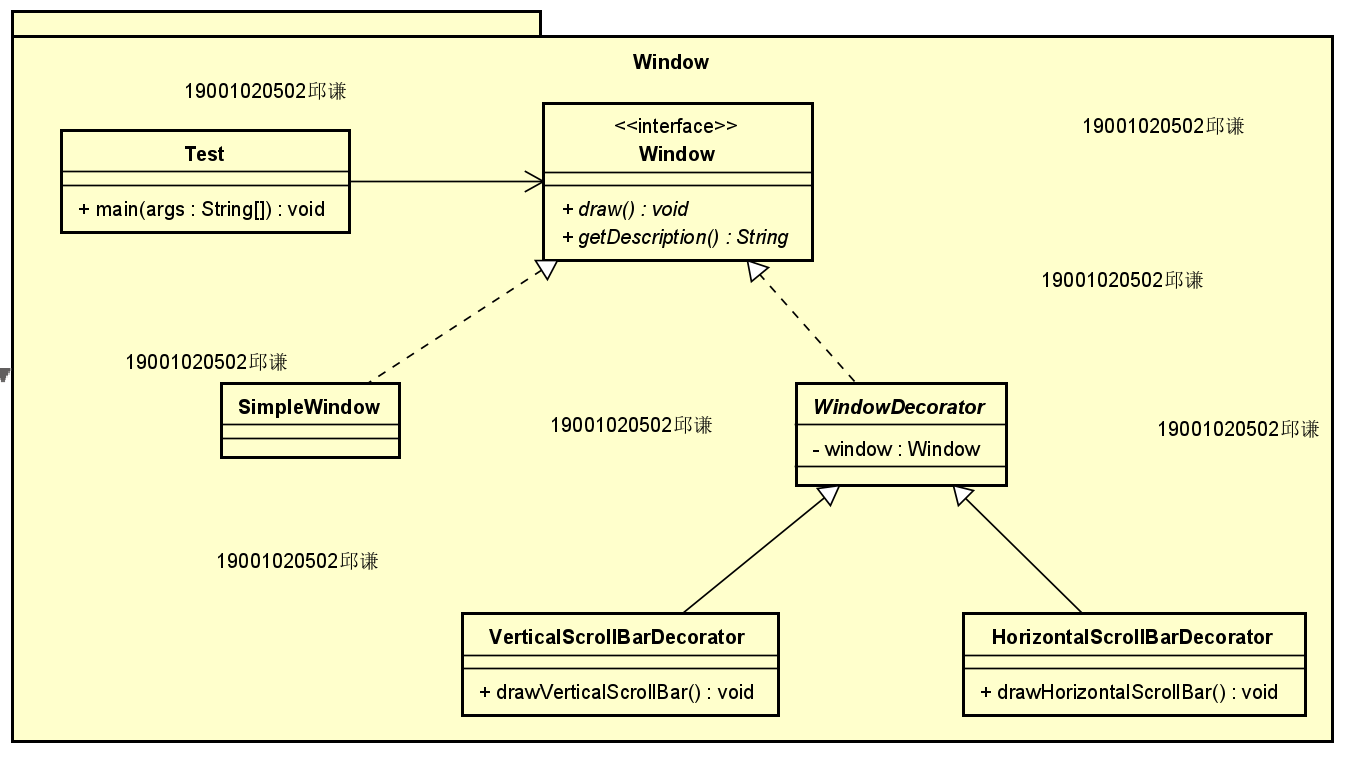
package room;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public abstract class RoomDecorator implements Room{  
 protected Room room;  
  
 public RoomDecorator(Room room) {  
 this.room = room;  
 }  
  
 @Override  
 public void room() {  
 room.room();  
 }  
  
}

package room;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public class BlueDecorator extends RoomDecorator{  
 public BlueDecorator(Room room) {  
 super(room);  
 }  
  
 @Override  
 public String getDescription() {  
 return room.getDescription() + ", with blue decorator";  
 }  
  
 @Override  
 public void room() {  
 room.room();  
 getColor();  
 }  
  
 public void getColor() {  
 System.*out*.println("blue");  
 }  
}

package room;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public class RedDecorator extends RoomDecorator{  
 public RedDecorator(Room room) {  
 super(room);  
 }  
  
 @Override  
 public String getDescription() {  
 return room.getDescription() + ", with red decorator";  
 }  
  
 @Override  
 public void room() {  
 room.room();  
 getColor();  
 }  
  
 public void getColor() {  
 System.*out*.println("red");  
 }  
}

package room;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public class Test {  
 public static void main(String[] args) {  
 Room roomDecorator = new BlueDecorator(new SimpleRoom());  
 System.out.println(roomDecorator);  
  
 Room roomDecorator2 = new RedDecorator(new SimpleRoom());  
 System.out.println(roomDecorator2);  
 }  
}

**hw04.**



package window;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public interface Window {  
 public void draw();  
 public String getDescription();  
}

package window;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public class SimpleWindow implements Window{  
 @Override  
 public void draw() {  
  
 }  
  
 @Override  
 public String getDescription() {  
 return "simple window";  
 }  
}

package window;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public abstract class WindowDecorator implements Window{  
 protected Window window;  
  
 public WindowDecorator(Window window) {  
 this.window = window;  
 }  
  
 @Override  
 public void draw() {  
 window.draw();  
 }  
}

package window;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public class HorizontalScrollBarDecorator extends WindowDecorator{  
 public HorizontalScrollBarDecorator(Window window) {  
 super(window);  
 }  
  
 @Override  
 public String getDescription() {  
 return window.getDescription() + ", including horizontal scrollbars";  
 }  
  
 @Override  
 public void draw() {  
 window.draw();  
 drawHorizontalScrollBar();  
  
 }  
  
 public void drawHorizontalScrollBar() {  
 System.*out*.println("drawHorizontalScrollBar");  
 }  
  
}

package window;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public class VerticalScrollBarDecorator extends WindowDecorator {  
  
 public VerticalScrollBarDecorator(Window window) {  
 super(window);  
 }  
  
 @Override  
 public String getDescription() {  
 return window.getDescription() + ", including vertical scrollbars";  
 }  
  
 @Override  
 public void draw() {  
 window.draw();  
 drawVerticalScrollBar();  
 }  
  
 public void drawVerticalScrollBar() {  
 System.*out*.println("drawVerticalScrollBar");  
 }  
}

package window;  
  
*/\*\*  
 \** ***@author*** *QiuQian  
 \*/*public class Test {  
 public static void main(String[] args) {  
 Window windowDecorator = new HorizontalScrollBarDecorator(new VerticalScrollBarDecorator(new SimpleWindow()));  
 System.*out*.println(windowDecorator.getDescription());  
  
 }  
}