Name: Aimable M and David R_

COSC 436: Object-Oriented Design and Programming In-class Exercise: Decorator Design Pattern

Problem:
The objective of this exercise is to implement the Decorator design pattern.

Your task:
We need to create a basic "**TextField**" class as the core class, and then create additional classes to add features to the basic class, such as adding borders, scroll bars. We don't need real GUI widgets in this exercise.

Steps:

1. Create an interface called **Widget**. This is the highest level common interface shared by all classes. Inside Widget interface, define a method **void draw().**

2. Create a core class called **TextField** that implements **Widget** interface. Add **width** and **height** as the two instance variables. Provide constructor in **TextField**, and implement the **draw()** method as printing out a sentence showing this is **TextField** with the values of **width** and **height**.

3. Create an abstract Decorator class called **Decorator** and make it implements **Widget** interface as well. Inside **Decorator** class, the most important step is to create an instance variable **Widget widget**. This is an aggregation relationship. Provide constructor for **Decorator** class. Implement **draw()** method in **Decorator**. This should be just simply calling widget.draw().

```java
// Abstract Decorator class
public abstract class Decorator implements Widget {
    protected Widget widget;

    public Decorator(Widget widget) {
        this.widget = widget;
    }

    @Override
    public void draw() {
        widget.draw();
    }
}
```

4. Then, we need to add some additional features. Create a class called **BorderDecorator** as the subclass of **Decorator**. In the **draw()** method of **BorderDecorator**, we should first call super.draw(), because this is how it delegates to the base core class. Then, write System.out.println(" BorderDecorator"); to indicate this is an additional feature created in addition to the base core class.

```java
// BorderDecorator concrete class
public class BorderDecorator extends Decorator {
    public BorderDecorator(Widget widget) {
        super(widget);
    }

    @Override
    public void draw() {
        super.draw();
        System.out.println(x:" BorderDecorator");
    }
}
```

5. Similarly, create another decorator lass called **ScrollDecorator**. Implement its **draw()** method in the similar way, but modify the additional feature to print out "ScrollDecorator".

```java
// ScrollDecorator concrete class
public class ScrollDecorator extends Decorator {
    public ScrollDecorator(Widget widget) {
        super(widget);
    }

    @Override
    public void draw() {
        super.draw();
        System.out.println(x:" ScrollDecorator");
    }
}
```

6. Finally, create a client class to show how they work. You can create objects with different features of TextField in this way:

```
Widget widget = new TextField(80, 24); // basic one widget.draw();
Widget widget2 = new ScrollDecorator(new TextField(80, 24)); // add scroll bar Widget2.draw();
Widget widget3 = new BorderDecorator(new BorderDecorator(new ScrollDecorator(new
TextField(80, 24)))); // add some borders
Widget3.draw();
```

```
PS C:\Users\aimab\OneDrive\Documents\GitHub\Object-Oriented-Design-and-Programming\exercise-12> & 'C:\Users\
-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\aimab\AppData\Roaming\Code\User\workspaceStorage\ea
e7adcb\redhat.java\jdt_ws\exercise-12_d3040a88\bin' 'Client'
Basic TextField:
TextField: width = 80, height = 24

TextField with ScrollDecorator:
TextField: width = 80, height = 24
 ScrollDecorator

TextField with multiple decorators:
TextField: width = 80, height = 24
 ScrollDecorator
 BorderDecorator
 BorderDecorator
PS C:\Users\aimab\OneDrive\Documents\GitHub\Object-Oriented-Design-and-Programming\exercise-12>
```

Upload your code to the Blackboard when you are done.