

Decorator Pattern - GUI Example - Java Sourcecode

Decorator Design Pattern is applied in Graphical User Interface toolkits to add dynamically windows behaviors.

Java Source Code Example for the Decorator Design Pattern - GUI Application

Decorator Design Pattern is applied in Graphical User Interface toolkits to add dynamically windows behaviors.

As discussed previously a persistence API can have many implementations depending on the presence or absence of a relational database, a file system, as well as on the underlying operating system.

The code below illustrates the Window interface representing the component interface.

```
package decorator;

/**
 * Window Interface
 *
 * Component window
 */
public interface Window {

    public void renderWindow();

}
```

The code below illustrates concrete window implementation:

```
package decorator;

/**
 * Window implementation
 *
 * Concrete implementation
 */
public class SimpleWindow implements Window {

    @Override
    public void renderWindow() {
        // implementation of rendering details
    }

}
```

The code below illustrates the basic decorated window implementation. Note that the decorator maintains a reference to a window object that is being decorated.

```
package decorator;
```

```

/**
 *
 */
public class DecoratedWindow implements Window{

/**
 * private reference to the window being decorated
 */
private Window privateWindowRefernce = null;

public DecoratedWindow( Window windowRefernce) {

    this.privateWindowRefernce = windowRefernce;
}

@Override
public void renderWindow() {

    privateWindowRefernce.renderWindow();

}
}

```

The code below illustrates a scrollable window which is a decorated window

package decorator;

```

/**
 * Concrete Decorator with extended state
 *
 * Scrollable window creates a window that is scrollable
 *
 */
public class ScrollableWindow extends DecoratedWindow{

/**
 * Additional State
 */
private Object scrollBarObjectRepresentation = null;

public ScrollableWindow(Window windowRefernce) {

    super(windowRefernce);
}

@Override
public void renderWindow() {

    // render scroll bar
    renderScrollBarObject();

    // render decorated window
    super.renderWindow();
}

private void renderScrollBarObject() {

    // prepare scroll bar
    scrollBarObjectRepresentation = new Object();
}
}

```

```
// render scrollbar  
  
}  
}
```

The code below illustrates a driver program for the window component. Note how the scrolling functionality has been added dynamically at runtime.

```
package decorator;  
  
public class GUIDriver {  
  
    public static void main(String[] args) {  
  
        // create a new window  
        Window window = new ConcreteWindow();  
  
        window.renderWindow();  
  
        // at some point later  
        // maybe text size becomes larger than the window  
        // thus the scrolling behavior must be added  
  
        // decorate old window  
        window = new ScrollableWindow(window);  
  
        // now window object  
        // has additional behavior / state  
  
        window.renderWindow();  
  
    }  
}
```