Java Decorator Design Pattern

What is decorator design pattern in Java?

          Decorator design pattern is used to **enhance the functionality of a particular object at run-time** or dynamically.

          At the same time **other instance of same class will not be affected by this** so individual object gets the new behavior.

          Basically we wrap the original object through decorator object.

          Decorator design pattern is based on abstract classes and we derive concrete implementation from that classes,

          It’s a structural design pattern and most widely used.

Problem which is solved by Decorator Pattern:

Now the question is why this pattern has came into existence what is the problem with existing system, so the answer is if anyone wants to add some functionality to individual object or change the state of particular object at run time it is not possible what the possible is we can provide the specific behavior to all the object of that class at design time by the help of inheritance or using subclass, but **Decorator pattern** makes possible that we provide individual object of same class a specific behavior or state at run time. This doesn’t affect other object of same [Class in Java](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html).

**When to use Decorator pattern in Java**

**          When sub classing is become impractical and we need large number of** different possibilities to make independent object or we can say we have number of combination for an object.

          Secondly when we want to add functionality to individual object not to all object at run-time we use decorator design pattern.

1. **IN JDK, Decorator pattern (BufferedInputStream can decorate other streams such as FilterInputStream)**

**(**[**http://stackoverflow.com/questions/1673841/examples-of-gof-design-patterns**](http://stackoverflow.com/questions/1673841/examples-of-gof-design-patterns)**)**

**Code Example of decorator design pattern:**

<http://java67.blogspot.com/2013/07/decorator-design-pattern-in-java-real-life-example-tutorial.html>

**Sandwich.java**

**import** **java.math.BigDecimal**;

/\*\*

\* Base class for all types of Sandwich, cost method is abstract because

\* different sandwiches has different price.

\*

\* @author Javin Paul

\*/

**public** **abstract** **class** **Sandwich** {

**protected** String description = "Sandwich";

**public** String **getDescription**(){

**return** description;

}

**public** **abstract** BigDecimal **price**();

}

**WhiteBreadSandWich.java**

**import** **java.math.BigDecimal**;

/\*\*

\* A Concrete implementation of abstract Sandwich class, which represent a WhiteBread

\* Sandwich, whose price is 3.0$.

\*

\* @author Javin Paul

\*/

**public** **class** **WhiteBreadSandWich** **extends** Sandwich {

**public** **WhiteBreadSandWich**(String desc){

description = desc;

}

**@Override**

**public** BigDecimal **price**() {

**return** **new** **BigDecimal**("3.0");

}

}

**SandWichDecorator.java**

/\*\*

\* Base class for Decorators, this class inherit from Sandwich, so that

\* it can be of same type, which is required to pass decorators where

\* original object is expected. Later, this class will also come handy

\* to provide common functionalities to Decorators.

\*

\* @author

\*/

**public** **abstract** **class** **SandWichDecorator** **extends** Sandwich {

**@Override**

**public** **abstract** BigDecimal **price**();

}

**CheeseDecorator.java**

**import** **java.math.BigDecimal**;

/\*\*

\* A Decorator class, which adds cheese (new functionality) into Sandwich object.

\* This Decorator class modifies price() and getDescritption() method to implement

\* new behavior.

\*

\* @author

\*/

**public** **class** **CheeseDecorator** **extends** SandWichDecorator{

Sandwich currentSandwich;

**public** **CheeseDecorator**(Sandwich sw){

currentSandwich = sw;

}

**@Override**

**public** String **getDescription**(){

**return** currentSandwich.getDescription() + ", Cheese";

}

**@Override**

**public** BigDecimal **price**() {

**return** currentSandwich.price().add(**new** BigDecimal("0.50"));

}

}

**SandwichMaker.java**

/\*\*

\* Test class to demonstrate How Decorator Pattern in Java work together. This class

\* first creates a Sandwich and decorates it with extra cheese. This is nice example

\* of how to provide new functionalities to an object at runtime using Decorator Pattern.

\*

\* @author Javain Paul

\*/

**public** **class** **SandwichMaker** {

**public** **static** **void** **main**(String args[]){

Sandwich mySandwich = **new** WhiteBreadSandWich("White bread Sandwich");

System.out.printf("Price of %s is $%.2f %n", mySandwich.getDescription(),

mySandwich.price());

//adding extra cheese using Decorator Pattter

mySandwich = **new** CheeseDecorator(mySandwich);

System.out.printf("Price of %s is $%.2f %n", mySandwich.getDescription(),

mySandwich.price());

}

}

**Output:**

Price of White bread Sandwich is $3.**00**

Price of White bread Sandwich, Cheese is $3.**50**

To better understand concept of decorator design pattern let see a code example using Decorator Pattern in Java. You can also look inside JDK and find what are classes and packages which are using decorator pattern.

**Advantage of Decorator design Pattern in Java**

In brief we see what the main advantages of using decorator design patterns are.

1.      Decorator Pattern is flexible than inheritance because **inheritance add responsibilities at compile time and it will add at run-time.**

2.      Decorator pattern enhance or modify the object functionality

**Disadvantage**

Main disadvantage of using Decorator Pattern in Java is that the code maintenance can be a problem as it provides a lot of similar kind of small objects (each decorator).

That’s all on **decorator design pattern in Java**. To get mastery on decorator pattern I suggest looking inside JDK library itself and finding what classes are decorated, why they are decorated. Also think of scenario where inheritance is impractical and you look more flexibility and try to **use decorator pattern in Java** there.

Read more: <http://javarevisited.blogspot.com/2011/11/decorator-design-pattern-java-example.html#ixzz2y88MdXaV>

**Decorator** – add additional responsibilities dynamically to an object. Examples -

* All subclasses of java.io.InputStream, OutputStream, Reader and Writer have a constructor taking an instance of same type.
* Almost all implementations of java.util.List, Set and Map have a constructor taking an instance of same type.
* java.util.Collections, the checkedXXX(), synchronizedXXX() and unmodifiableXXX() methods.
* javax.servlet.http.HttpServletRequestWrapper and HttpServletResponseWrapper
* Display tag custom tag library proved option to decorator for rendering of tables in JSP.
* Any client side JavaScript library which has render-er or parser like Yahoo UI datatable, JQuery grid

Good explanation at the following

<http://www.journaldev.com/1540/decorator-pattern-in-java-example-tutorial>