***Façade***

*Description:*

Facades are all around us in the real world. Operating systems are one such example - you don't see all the inner workings of your computer, but the OS provides a simplified interface to use the machine. Buildings also have a facade - the exterior of the building. Wikipedia gives us a nice link between software architecture and standard architecture:

This pattern involves a single class which provides simplified methods required by client and delegates calls to methods of existing system classes.

* Provide a unified interface to set of interfaces in a subsystem.
* Facade defines a high-level interface that makes the subsystem easier to use.
* The Facade can be used to hide the inner workings of a third party library, or some legacy code.
* All that the client needs to do is interact with the Facade, and not the subsystem that it is encompassing.

*Program Example:*

**package** DesignPatternFacade;

**public** **interface** Design

{

**public** **abstract** **void** designFun ();

}

**package** DesignPatternFacade;

**public** **class** Model1 **implements** Design{

**public** **void** designFun()

{

System.*out*.println("Model Design");

}

}

**package** DesignPatternFacade;

**public** **class** Model2 **implements** Design

{

**public** **void** designFun()

{

System.*out*.println("Model2 Design");

}

}

**package** DesignPatternFacade;

**public** **class** Model3 **implements** Design{

**public** **void** designFun()

{

System.*out*.println("Model3 Design");

}

}

**package** DesignPatternFacade;

**public** **class** designProducer

{

**private** Design Model1;

**private** Design Model2;

**private** Design Model3;

**public** designProducer() {

Model1 = **new** Model1();

Model2 = **new** Model2();

Model3 = **new** Model3();

}

**public** **void** getModel1Design()

{

Model1.designFun();

}

**public** **void** getModel2Design()

{

Model2.designFun();

}

**public** **void** getModel3Design()

{

Model3.designFun();

}

}

**package** DesignPatternFacade;

**public** **class** FacadeClient {

**public** **static** **void** main (String arg[])

{

designProducer dp = **new** designProducer();

dp.getModel1Design();

dp.getModel2Design();

dp.getModel3Design();

}

Output:

Model1 Design

Model2 Design

Model3 Design

***Factory***

*Description:*

In java, the factory pattern is used to create instances of different classes of same type

This pattern introduces loose coupling between classes which is the most important principle one should consider and apply while designing the application architecture.

Factory pattern is most suitable where there is some complex object creation steps are involved. To ensure that these steps are centralized and not exposed to composing classes, factory pattern should be used

Factory pattern, for creating instances for your classes, Factory, as name suggest, is a place to create some different products which are somehow similar in features yet divided in categories.

*Program:*

**package** DesignPatternFactory;

**public** **interface** ProdItem {

**public** **void** itemlist();

}

**package** DesignPatternFactory;

**import** java.util.\*;

**public** **class** product1 **implements** ProdItem{

**public** **void** itemlist() {

System.*out*.println("Product 1 item list");

List<String> str = **new** ArrayList<String>();

str.add("Mirror");

str.add("Cables");

str.add("Handle");

Iterator itr = str.iterator();

**for** (String s: str)

{

System.*out*.println(s);

}

}

}

**package** DesignPatternFactory;

**public** **class** product2 **implements** ProdItem {

**public** **void** itemlist() {

System.*out*.println("Product 2 item list");

}

}

**package** DesignPatternFactory;

**public** **class** product3 **implements** ProdItem

{

**public** **void** itemlist()

{

System.*out*.println("Product 3 item list");

}

}

**package** DesignPatternFactory;

**public** **class** ProdFactory {

**public** ProdItem getItem(String product){

**if**(product == **null**){

**return** **null**;

}

**if**(product.equalsIgnoreCase("product1"))

{

**return** **new** product1();

}

**else** **if**(product.equalsIgnoreCase("product2")){

**return** **new** product2();

}

**else** **if**(product.equalsIgnoreCase("product3")){

**return** **new** product3();

}

**return** **null**;

}

}

**package** DesignPatternFactory;

**public** **class** ProductList {

**public** **static** **void** main (String arg[])

{

ProdFactory items = **new** ProdFactory();

ProdItem item1 = items.getItem("product1");

item1.itemlist();

ProdItem item2 = items.getItem("product2");

item2.itemlist();

ProdItem item3 = items.getItem("product3");

item3.itemlist();

}

}

Output:

Product 1 item list

Mirror

Cables

Handle

Product 2 item list

Product 3 item list