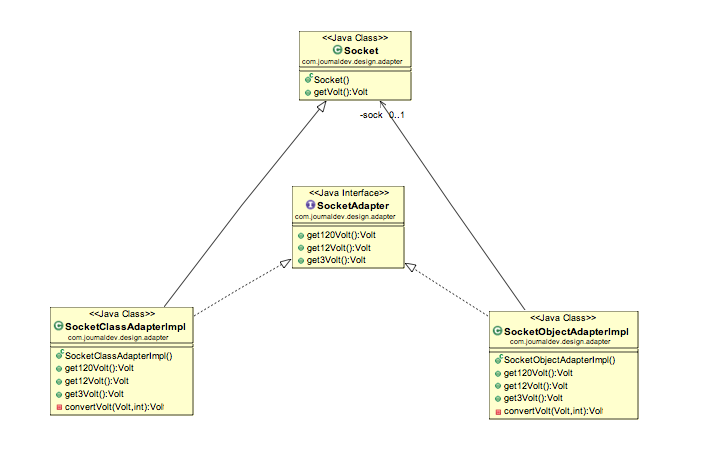
**Adapter Design Pattern in Java**



Adapter design pattern is one of the **structural design pattern** and its used so that two unrelated interfaces can work together. The object that joins these unrelated interface is called an **Adapter**.

**Adapter Design Pattern**

One of the great real life example of Adapter design pattern is mobile charger. Mobile battery needs 3 volts to charge but the normal socket produces either 120V (US) or 240V (India). So the mobile charger works as an adapter between mobile charging socket and the wall socket. We will try to implement multi-adapter using adapter design pattern in this tutorial. So first of all we will have two classes - Volt (to measure volts) and Socket (producing constant volts of 120V).

package com.journaldev.design.adapter;

public class Volt {

private int volts;

public Volt(int v){

this.volts=v;

}

public int getVolts() {

return volts;

}

public void setVolts(int volts) {

this.volts = volts;

}

}

package com.journaldev.design.adapter;

public class Socket {

public Volt getVolt(){

return new Volt(120);

}

}

Now we want to build an adapter that can produce 3 volts, 12 volts and default 120 volts. So first of all we will create an adapter interface with these methods.

package com.journaldev.design.adapter;

public interface SocketAdapter {

public Volt get120Volt();

public Volt get12Volt();

public Volt get3Volt();

}

**Two Way Adapter Pattern**

While implementing Adapter pattern, there are two approaches - class adapter and object adapter - however both these approaches produce same result.

1. **Class Adapter** - This form uses **java inheritance** and extends the source interface, in our case Socket class.
2. **Object Adapter** - This form uses **Java Composition** and adapter contains the source object.

**Adapter Design Pattern - Class Adapter**

Here is the **class adapter** approach implementation of our adapter.

package com.journaldev.design.adapter;

//Using inheritance for adapter pattern

public class SocketClassAdapterImpl extends Socket implements SocketAdapter{

@Override

public Volt get120Volt() {

return getVolt();

}

@Override

public Volt get12Volt() {

Volt v= getVolt();

return convertVolt(v,10);

}

@Override

public Volt get3Volt() {

Volt v= getVolt();

return convertVolt(v,40);

}

private Volt convertVolt(Volt v, int i) {

return new Volt(v.getVolts()/i);

}

}

**Adapter Design Pattern - Object Adapter Implementation**

Here is the **Object adapter** implementation of our adapter.

package com.journaldev.design.adapter;

public class SocketObjectAdapterImpl implements SocketAdapter{

//Using Composition for adapter pattern

private Socket sock = new Socket();

@Override

public Volt get120Volt() {

return sock.getVolt();

}

@Override

public Volt get12Volt() {

Volt v= sock.getVolt();

return convertVolt(v,10);

}

@Override

public Volt get3Volt() {

Volt v= sock.getVolt();

return convertVolt(v,40);

}

private Volt convertVolt(Volt v, int i) {

return new Volt(v.getVolts()/i);

}

}

Notice that both the adapter implementations are almost same and they implement the SocketAdapter interface. The adapter interface can also be an **abstract class**. Here is a test program to consume our adapter design pattern implementation.

package com.journaldev.design.test;

import com.journaldev.design.adapter.SocketAdapter;

import com.journaldev.design.adapter.SocketClassAdapterImpl;

import com.journaldev.design.adapter.SocketObjectAdapterImpl;

import com.journaldev.design.adapter.Volt;

public class AdapterPatternTest {

public static void main(String[] args) {

testClassAdapter();

testObjectAdapter();

}

private static void testObjectAdapter() {

SocketAdapter sockAdapter = new SocketObjectAdapterImpl();

Volt v3 = getVolt(sockAdapter,3);

Volt v12 = getVolt(sockAdapter,12);

Volt v120 = getVolt(sockAdapter,120);

System.out.println("v3 volts using Object Adapter="+v3.getVolts());

System.out.println("v12 volts using Object Adapter="+v12.getVolts());

System.out.println("v120 volts using Object Adapter="+v120.getVolts());

}

private static void testClassAdapter() {

SocketAdapter sockAdapter = new SocketClassAdapterImpl();

Volt v3 = getVolt(sockAdapter,3);

Volt v12 = getVolt(sockAdapter,12);

Volt v120 = getVolt(sockAdapter,120);

System.out.println("v3 volts using Class Adapter="+v3.getVolts());

System.out.println("v12 volts using Class Adapter="+v12.getVolts());

System.out.println("v120 volts using Class Adapter="+v120.getVolts());

}

private static Volt getVolt(SocketAdapter sockAdapter, int i) {

switch (i){

case 3: return sockAdapter.get3Volt();

case 12: return sockAdapter.get12Volt();

case 120: return sockAdapter.get120Volt();

default: return sockAdapter.get120Volt();

}

}

}

When we run above test program, we get following output.

v3 volts using Class Adapter=3

v12 volts using Class Adapter=12

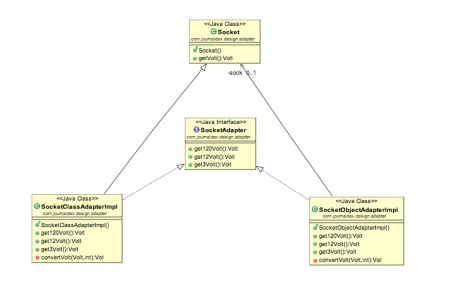
v120 volts using Class Adapter=120

v3 volts using Object Adapter=3

v12 volts using Object Adapter=12

v120 volts using Object Adapter=120

**Adapter Design Pattern Class Diagram**

[](https://journaldev.nyc3.digitaloceanspaces.com/2013/07/adapter-pattern-java-class-diagram.png)

**Adapter Design Pattern Example in JDK**

Some of the adapter design pattern example I could easily find in JDK classes are;

* java.util.Arrays#asList()
* java.io.InputStreamReader(InputStream) (returns a Reader)
* java.io.OutputStreamWriter(OutputStream) (returns a Writer)

That’s all for adapter design pattern in java.