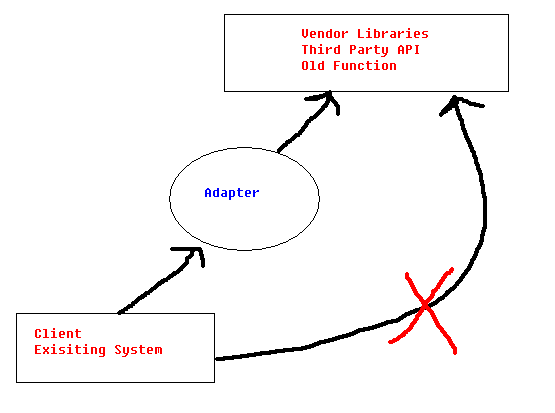
ADAPTOR DESIGN PATTERN

Adapter , Adapter , Adapter ~ Actually Adapter design pattern can consider as a simple conversion program / class. It usually used to make two incompatible interfaces or classes to work together. Please review below Adapter design pattern image draw by me, a bit ugly :)

[](http://www.mkyong.com/wp-content/uploads/2008/09/adapter-graphic.bmp)

**Case Study**

Company A developed a program in Java to display all of the products’ detail in console. This Java program is very simple and it take an Iterator collection and iterate it to display product one by one.

However Company A outsource Product back end system to a vendor called Vendor B. Vendor B came out a system which will return all products as Enumeration collection.

Arg…wait… return Product as Enumeration? But current Company A system is design to accept product as Iterator. Company A design are all base on Iterator collection to function, now Vendor B provide an old and obsolete function (Enumeration), what should we do?

Above scenarios sound similar to everyone right? Yes it always happened to me as well. So .. who going to change it? To be frankly, i can guarantee Vendor B 100% will not change it for you.

However company A also cannot afford to redesign it because it invested so much time and manpower effort already.

Here Adapter come in place. What we need to do is create an adapter class which can convert Enumeration to Iterator. Understand? Study below code may make sense to you.

**Code Study**

**This is an “EnumProduct” class provided by vendor, it will return Product as Enumeration by calling getProduct () function.**

**package** com.mkyong.adapter;

**import** java.util.Enumeration;

**import** java.util.Vector;

**public** **class** EnumProduct

{

**private** Vector product;

**public** EnumProduct(){

product = **new** Vector();

setProduct("ProductA");

setProduct("ProductB");

setProduct("ProductC");

}

**public** **void** setProduct(String s){

product.add(s);

}

**public** Enumeration getProduct(){

Enumeration<String> eProduct = product.elements();

**return** eProduct;

}

}

**Here is Company A “Product” program which is use to display all products on console screen.**

**package** com.mkyong.adapter;

**import** java.util.Enumeration;

**import** java.util.Vector;

**import** java.util.Iterator;

**public** **class** Product

{

*//cant change here for some reason*

**public** **void** displayProduct(Iterator iterator){

**for** (; iterator.hasNext();)

System.out.println(iterator.next());

}

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

Product product = **new** Product();

*//product.displayProduct(); //display it*

}

}

**Company A use displayProduct(Iterator iterator) function to display all of the products information in console. Company A do not want to change and insist want accept Iterator as parameter.**

**Here Adapter Design Pattern can make it work without any changes involve in current system and vendor’s class as well, what Company A need to do is create an Adapter class to convert Enumeration to Iterator like below**

**package** com.mkyong.adapter;

**import** java.util.Iterator;

**import** java.util.Enumeration;

**public** **class** EnumToIteratorAdapter **implements** Iterator

{

Enumeration enumA;

**public** EnumToIteratorAdapter(Enumeration e){

enumA = e;

}

**public** **boolean** hasNext(){

**return** enumA.hasMoreElements();

}

**public** Object next(){

**return** enumA.nextElement();

}

**public** **void** remove(){

**throw** **new** UnsupportedOperationException();

}

}

**Ok, now revisit Company A “Product” class, it can accept Enumeration now!!!**

**package** com.mkyong.adapter;

**import** java.util.Enumeration;

**import** java.util.Vector;

**import** java.util.Iterator;

**public** **class** Product

{

**public** **void** displayProduct(Iterator iterator){

**for** (; iterator.hasNext();)

System.out.println(iterator.next());

}

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

Product product = **new** Product();

EnumProduct enumProduct = **new** EnumProduct();

EnumToIteratorAdapter enumToIteratorAdapter = **new** EnumToIteratorAdapter(enumProduct.getProduct());

product.displayProduct(enumToIteratorAdapter);

}

}

**We didn’t changed anything in either Product Class nor EnumProduct class. We just create a new Adapter class to convert Enumration to Iterator class. Done.**

### Adapter Pattern Example in JDK

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

Another example:

Consider that we have a third party library that provides print string functionality through PrintString class.  
This is our **Adaptee**. I know this is silly assumption but lets go with it for now.  
  
**PrintString.java(Adaptee) :**

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/472801/Adapter-Design-Pattern-in-Java)

package org.arpit.javapostsforlearning.designpatterns;

public class PrintString {

public void print(String s)

{

System.out.println(s);

}

}

Client deals with ArrayList<String> but not with string. We have provided a PrintableList interface that expects the client input. This is our **target**.

#### PrintableList.java(Target)

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/472801/Adapter-Design-Pattern-in-Java)

package org.arpit.javapostsforlearning.designpatterns;

import java.util.ArrayList;

public interface PrintableList {

void printList(ArrayList<String> list);

}

Let's assume we can not change it now. Finally we have PrintableListAdapter class which will implement PrintableList interface and will deal with our adaptee class.

#### PrintableListAdapter.java(Adapter)

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/472801/Adapter-Design-Pattern-in-Java)

package org.arpit.javapostsforlearning.designpatterns;

import java.util.ArrayList;

public class PrintableListAdapter implements PrintableList{

public void printList(ArrayList<String> list) {

*//Converting ArrayList<String> to String so that we can pass String to*

*// adaptee class*

String listString = "";

for (String s : list)

{

listString += s + "\t";

}

*// instantiating adaptee class*

PrintString printString=new PrintString();

ps.print(listString);

}

}

#### AdapterDesignPatternMain.java

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/472801/Adapter-Design-Pattern-in-Java)

package org.arpit.javapostsforlearning.designpatterns;

import java.util.ArrayList;

public class AdapterDesignPatternMain {

public static void main(String[] args)

{

ArrayList<String> list=new ArrayList<String>();

list.add("one");

list.add("two");

list.add("three");

PrintableList pl=new PrintableListAdapter();

pl.printList(list);

}

}

#### Output

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/472801/Adapter-Design-Pattern-in-Java)

one two three

[**Adapter**](http://en.wikipedia.org/wiki/Adapter_pattern) **(recognizeable by creational methods taking an instance of different abstract/interface type and returning an implementation of own/another abstract/interface type which decorates/overrides the given instance)**

* [java.util.Arrays#asList()](http://docs.oracle.com/javase/6/docs/api/java/util/Arrays.html#asList%28T...%29)
* [java.io.InputStreamReader(InputStream)](http://docs.oracle.com/javase/6/docs/api/java/io/InputStreamReader.html#InputStreamReader%28java.io.InputStream%29) (returns a Reader)
* [java.io.OutputStreamWriter(OutputStream)](http://docs.oracle.com/javase/6/docs/api/java/io/OutputStreamWriter.html#OutputStreamWriter%28java.io.OutputStream%29) (returns a Writer)
* [javax.xml.bind.annotation.adapters.XmlAdapter#marshal()](http://docs.oracle.com/javase/6/docs/api/javax/xml/bind/annotation/adapters/XmlAdapter.html#marshal%28BoundType%29) and [#unmarshal()](http://docs.oracle.com/javase/6/docs/api/javax/xml/bind/annotation/adapters/XmlAdapter.html#unmarshal%28ValueType%29)