1. **Adapter Pattern :**

Converts the interface of a class into another interface that a client wants.

It used so that two unrelated interfaces can work together. The object that joins these unrelated interface is called an Adapter.

The Adapter Pattern is also known as "***Wrapper***".

1. **Real world example:**

Ever tried to use a your camera memory card in your laptop. You cannot use it directly simply because there is no port in laptop which accept it. You must use a compatible card reader. You put your memory card into card reader and then inject card reader into laptop. This card reader can be called adapter.

A similar example is your mobile charger or your laptop charger which can be used with any power supply without fear of the variance power supply in different locations. That is also called power “adapter”.

Adapter Design is very useful for the system integration when some other existing components have to be adapted by the existing system without source code modifications.

1. **Where to use Adapter Design Pattern?**

The main use of this pattern is when a class that you need to use doesn’t meet the requirements of an interface. e.g. If you want to read the system input through command prompt in java then given below code is common way to do it:

*BufferedReader br = new BufferedReader(new InputStreamReader(System.in));*

*System.out.print("Enter String");*

*String s = br.readLine();*

*System.out.print("Enter input: " + s);*

Now observe above code carefully.

1) System.in is static instance of InputStream declared as:

public final static InputStream in = nullInputStream();

2) BufferedReader as java docs define, reads a character stream.

public class BufferedReader extends Reader{..}

Now here is the problem. System.in provides byte stream where BufferedReader expects character stream. How they will work together?

This is the ideal situation to put a adapter in between two incompatible interfaces. InputStreamReader does exactly this thing and works adapter between System.in and BufferedReader.

## **Participants of Adapter Design Pattern:**

The classes and/or objects participating in this pattern are listed as below:

* Target (BufferedReader): It defines the application-specific interface that Client uses directly.
* Adapter (InputStreamReader): It adapts the interface Adaptee to the Target interface. It’s middle man.
* Adaptee (System.in): It defines an existing incompatible interface that needs adapting before using in application.
* Client: It is your application that works with Target interface.

**Other example implementations of Adapter Design Pattern**

**A. java.util.Arrays#asList()**

This method accepts multiple strings and return a list of input strings. Though it’s very basic usage, but it’s what a adapter does, right?

1. **java.io.OutputStreamWriter(OutputStream)**

It’s similar to above usecase we discussed in this post:

*Writer writer = new OutputStreamWriter(new FileOutputStream("c:\\data\\output.txt"));*

*writer.write("Hello World");*

## **Advantage of Adapter Pattern**

It allows two or more previously incompatible objects to interact.

It allows reusability of existing functionality.

## **Usage of Adapter pattern:**

* When an object needs to utilize an existing class with an incompatible interface.
* When you want to create a reusable class that cooperates with classes which don't have compatible interfaces.

Ref: <https://www.javatpoint.com/adapter-pattern>

<https://howtodoinjava.com/design-patterns/structural/adapter-design-pattern-in-java/>