**Adapter Pattern**

**Overview**

This is a great pattern to describe and great pattern to use. As name says ‘adapter pattern’ works just as adapter works. You buy a mobile phone in Australia and travel to India, you would find that wall socket uses circular outlets instead of standard outlet with vertical outlets. You would need an adaptor to convert type of charger you use to type of charger outlet would support.

Freeman Eric., Freeman Elisa. (2004) define adapter pattern as a pattern in code that converts an interface of a class into another interface that client expects. Adapter lets classes work together which otherwise would not work because of difference in interface structure when two interfaces are involved or two classes or more are involved.

Adapter pattern would have an adapter and adaptee, instead of using composition to adapt the adaptee, the adapter subclasses the Adaptee and target classes. So , to make class A compatible to class B, a class C would interact with class B and make request compatible.

* Two incompatible classes/interfaces are created
* An adapter is third class/interface which comes in middle to transform value between C to A or A to C , as a separate object/interface B.

**Benefits**

The main benefit of using this pattern is to connect to dissimilar pattern/class/interfaces together with a bridged connection which simplifies requests or changes request in a way that suits to need of a class. Client object doesn’t need to know underlying logic of adapter class that handles transforming the request. Class C would bridge the gap between A and B in spite of different types in interfaces used or data used.

**Risks of such refactor**

Risk of such refactor would be that you will add more complexity by adding another interface/class which increases code size and confuse developers even more.

**Problem/Scenario**

We would consider an example where its more practical to scenario of work in software industry. In this example, I have considered a case where backend would send response to front end in form of XML. But, its obvious nowadays in development that most of front end layout only accepts or may mostly accept JSON payload.

**Benefit of Refactor**

- Adapter pattern can make XML response change to JSON payload without changing relevant classes with extra class.

- If XML data is required then we can easily take adapter class and continue to use regular classes pass right data.

**Before Refactor**

Before refactor, it would be just either converting XML data to JSON straightaway on the class that has XML data.

**After Refactor**

After refactor, an additional adapter class would pick XML file and convert to JSON data and if front end needs that in JSON, it would pick up from adapter class otherwise if XML is needed for some reason, it will be picked up as well interchangeably.

Before/after link

<https://github.com/sppanday/S120-PRT583-Group-A/tree/feat/design-pattern/s260598-PandaySurendra/Sprint-4-Deliverables/Task030_adapter_pattern/Task030_adapter_pattern/Task030_adapter_pattern>

**UML Diagram**

Please find it inside relevant project UML in pattern folder

/AdapterPattern.png

**Reference**

Freeman Eric., Freeman Elisa. 2004. 1st ed. O’Reily Media Inc. 1995, Sebasttopo, CA