# Overview:

* The adapter design pattern converts an interface of a class into another interface that clients expect:
  + Works as a bridge between two incompatible interfaces.
    - “Adapter” does the conversion.
  + Let’s classes work together that could not otherwise.
  + Also known as a “wrapper”.
* Comes under the structural pattern classification as this pattern combines the capability of two independent interfaces.
* The adapter acts to decouple the client from the implemented interface.
  + Encapsulates any future changes.
  + Client does not need to be modified each time it needs to operate against a different interface.
* Full of good OO design principles:
  + Use of object composition to wrap the adapter with an altered interface.
    - Can use an adapter with any subclass of the adapter.
  + Binds the client to an interface, not an implementation.

# Real World Examples:

* Simple AC adapters:
  + Only change the shape of the outlet so that it matches your plug.
  + Pass the AC current straight through.
* Complex AC adapters:
  + May need to step the power up or down to match your devices needs.
* Card reader:
  + Acts as an adapter between memory card and a laptop:
    - Plug the memory card into the card reader.
    - Plug the card reader into the laptop.
    - The memory card can now be read via laptop.

# Software Examples:

* You have an existing software system:
  + You need to add a new vendor class library.
  + The new vendor designed their interfaces differently than the last vendor.
* You do not want to solve the problem by changing your existing code and you cannot change the vendor’s code.
  + Write a class that adapts the new vendor interface into the one you are.
* Adapter acts as the middleman by receiving requests from the client and converting them into requests that make sense on the vendor classes.
* Old-world Enumerators:
  + Early collection types (Vector, Stack, Hashtable, etc.) implement a method (elements ()) which returns an Enumeration.
  + The Enumeration interface allows you to step through the elements of a collection.
    - Do not need to know the specifics of how they are managed in the collection.
* New-world Iterators:
  + Newer Collection classes use an Iterator interface.
    - Like Enumeration:
      * Allows you to iterate through a set of items in a collection.
      * Allows you to remove items.
* Both of these examples use adapters.

# When to use the Adapter Pattern:

* When you want to use an existing class, and its interface does not match the one you need.
* When you want to create a reusable class that cooperates with unrelated or unforeseen classes.
* When you need to use several existing subclasses, but it is impractical to adapt their interface by sub-classing every one.
  + An object adapter can adapt the interface of its parent class.

# Implementation:

# Participants:

* Target:
  + Defines the domain-specific interface that Client uses.
* Client:
  + Collaborates with objects conforming to the Target interface.
* Adaptee:
  + Defines an existing interface that needs adapting.
* Adapter:
  + Adapts the interface of Adaptee to the Target interface.
  + Involves a single class which is responsible to join functionalities of independent or incompatible interfaces.

# Client usage:

* Client makes a request to the adapter by calling a method on it using the target interface.
* The adapter translates the request into one or more calls on the adaptee using adaptee interface.
* The client receives the results of the call and never knows there is an adapter doing the translation.
* Essentially, clients call operations on an Adapter instance:
  + In turn, the adapter calls Adaptee operations that carry out the request.

# Class Adapter implementation:

* We have defined the adapter pattern, however, there are actually two kinds of adapters:
  + Object adapters:
    - What we have been studying up to this point in the class.
    - Relies on one object (the adapting object) containing another (the adapted object) (composition).
  + Class adapters:
    - Another way to implement the adapter pattern (uses multiple inheritance).
* A class adapter is less of a focus in Java because you need multiple inheritance to implement it:
  + Not possible in Java.
* That does not mean you might not encounter a need for class adapters down the road:
  + Can utilize interfaces as a workaround the lack of support for multiple inheritance in Java.
* A class adapter is very similar to an Object adapter:
  + The class adapter will subclass the Target and the Adaptee.
  + The object adapter will use composition to pass requests to an Adaptee
  + Composition versus Inheritance.
* A Class Adapter works by creating a new class which subclasses publicly from an abstract class define its interface:
  + Subclasses privately from our existing class to access its implementation.
  + Each wrapped methods calls its associated, privately inherited method.
* A class adapter adapts Adaptee to Target by committing to a concrete Adaptee class.
  + Will not work when we want to adapt a class and all its subclasses.
* A class adapter lets the Adapter override some of the Adaptee’s behavior since Adapter is a subclass of Adaptee.

# Summary:

* Adapters vary in the amount of work they do to adapt Adaptee to the Target interface:
  + Simple interface conversion (changing the names of operations to supporting an entirely different set of operations).
  + The amount of work the Adapter does depends on how similar the Target interface is to the Adaptee’s.
* Adapter frees you from worrying about existing interfaces.
* If I have a class that does what I need, I know that I can always use the adapter pattern to give it the correct interface.
* An advantage of the object adapter is that it can adapt an adaptee class and any of its subclasses (because of composition)
  + A class adapter is committed to one specific adaptee class.
* An advantage of a class adapter is that is that it does not need to re-implement its entire adaptee:
  + Can override the behavior of an adaptee if it needs to because it is just sub-classing.
* The object adapter is more flexible:
  + Composition is also preferred over inheritance.
  + Write very little code to delegate to the adaptee.
* A class adapter is more efficient:
  + There is only one, not an adapter and an adaptee.
* An object adapter can add new behavior and everything works with the adaptee class and all of its subclasses.