	Course Name: Design Patterns/Thinking LAB	**EXPERIMENT NO. 7**	
	Course Code: 20CP210P **Faculty: Dr. Ketan Sabale**	**Branch: CSE**	**Semester: IV**
Submitted by: Rhythm Shah **Roll no:** 22BCP071			

Objective: To familiarize students with standard Structural design patterns.
Experiment: Explain the Adapter design pattern and write a program using any object-oriented programming language to demonstrate the working of Adapter design pattern.

Theory:

Adapter is a structural design pattern that allows objects with incompatible interfaces to collaborate.

Adapter pattern works as a bridge between two incompatible interfaces. This type of design pattern comes under structural pattern as this pattern combines the capability of two independent interfaces.

Adapters can not only convert data into various formats but can also help objects with different interfaces collaborate. Here's how it works:

1. The adapter gets an interface, compatible with one of the existing objects.
2. Using this interface, the existing object can safely call the adapter's methods.
3. Upon receiving a call, the adapter passes the request to the second object, but in a format and order that the second object expects.

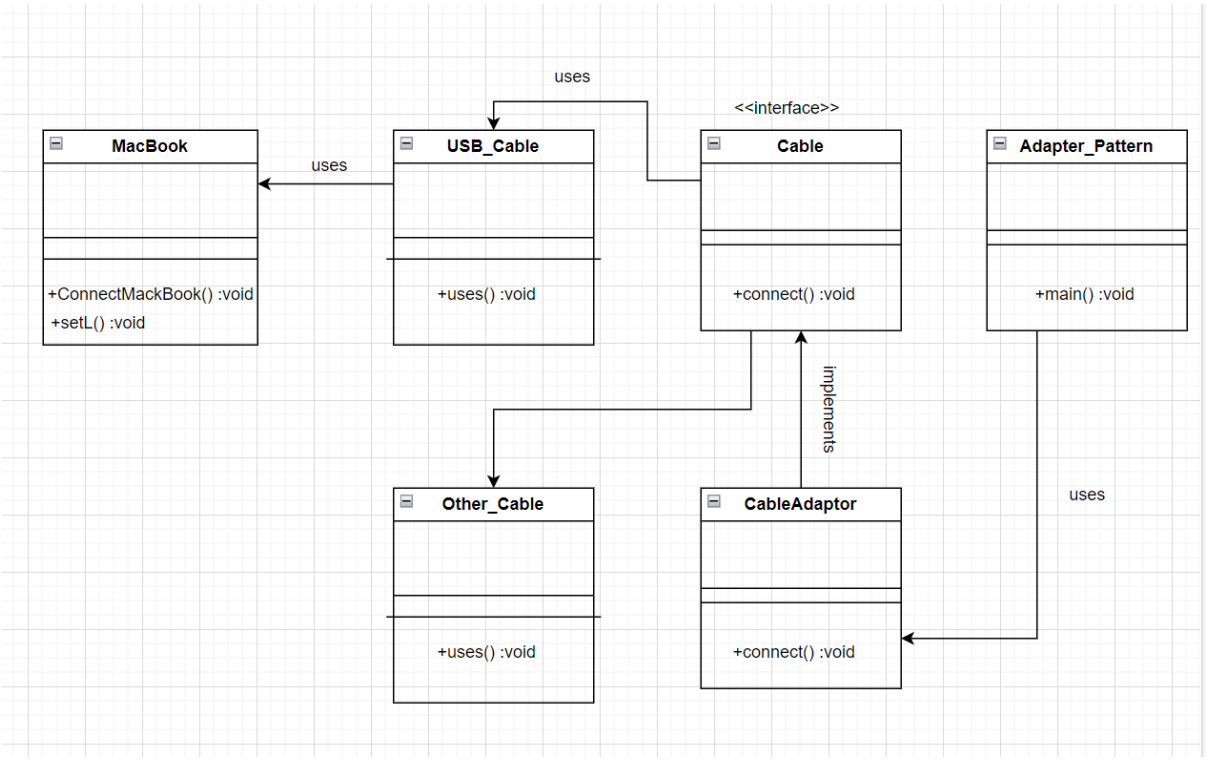
Implementation:

I have created an interface Cable which is implemented by class Cable Adaptor. A class MacBook with ab class of USB_Cable and a main method to call. There is problem in connecting usb_cable with MacBook and more connection problems therefore an Adaptor class is made which allows objects with incompatible interface to operate.

So, cable adaptor provides an interface therefore USB_Cable can connect with MacBook through adaptor. For that a connect method is made in interface and cable adaptor uses this method.

This was the implementation for Adapter Design Pattern.

UML Diagram:



Code:

```
interface Cable
{
   void connect (String str);
}

class Cable Adaptor implements Cable
{
   USB_Cable UC = new USB_Cable();
   public void connect(String str)
   {
      UC.uses(str);
   }
}

class MacBook{

   private Cable L;

   public Cable getL()
   {
      return L;
   }

   public void setL(Cable L)
   {
      this.L = L;
   }

   public void ConnectMacBook(String str)
   {
      L.connect(str);
   }
}

class USB_Cable
{
   public void uses(String str)
   {
      System.out.println(str);
   }
}

class Other_Cable
{
   public void uses(String str)
   {
```

```
        System.out.println("Different cable needes to connect");
    }
}

public class adapter_pattern {
    public static void main(String[] args)
    {
        Cable L = new CableAdaptor();
        MacBook MB = new MacBook();
        MB.setL(L);
        MB.ConnectMacBook("I am having problem in connecting USB Cable with
MackBook");
    }
}
```

Output: