1. **Introduction:** A brief introduction to the bridge pattern and the selected implementation. Provide a link to the original implementation and describe how the Bridge pattern is used in the implementation.

Bridge:

This is a structural design pattern where extending the class is changing from extending to compositing. The 2 main components are the abstract class and the implementation class; the client only knows the abstract class, which will call its concrete implementation class.

This is the example, https://refactoring.guru/design-patterns/bridge/java/example.
In this example, The Bridge pattern is used to solve the problem of extending the Device into 2 or more dimensentional class by create 2 class, Abstract class, Device and Implementation class, Remote. By passing the the Remote class to the Device class as a parameter, it create a bridge between the Device class and Remote class. In the example, there are two 2 devices created in the beginning, each is then passed a Remote and each device is able to call the remote method easily through the remote variable inside the device class.

2. **New Functionality:** Describe the new functionality you added to the implementation. Explain the motivation behind the new functionality.

New Functionality:

I add the new class SmartRemote class. The class has extra method call voiceControl(String command). Based on the command, the remote will call the method of its parent, such as channelUp(), channelDown(). The motivation behind the method is to demonstrate that even with a new class implemented from the Remote class and extend from Advanced Remote class, the Device class does not affected by new class, and the client just need to call the Device class, yet still able to make use of the new method of new class, SmartRemote Therefore, extending the Implementation class will not affect the code base of client, where it know only the Abstract class.

3. **Implementation:** Provide the code for the new functionality. You should include the new classes and methods you added to the implementation.

```
4.
   public class SmartRemote extends AdvancedRemote {

       public SmartRemote(Device device) {
           super(device);
       }

       public void voiceControl(String command){
           System.out.println("voiceControl activated");
           doCommand(command);
       }

       private void doCommand(String command){
           switch(command.toLowerCase()){
               case "mute":
                   mute();
                   break;
               case "power":
                   power();
                   break;
               case "volumndown":
                   volumeDown();
                   break;
               case "volumnup":
                   volumeUp();
                   break;
               case "channelup":
                   channelUp();
                   break;
               case "channeldown":
                   channelDown();
                   break;
               default:
           }
       }
   }
```

5. **Verification:** Describe how you verified that the new functionality works as expected. You can provide code snippets, screenshots, or any other relevant information.

```
6. public static void main(String[] args) {
        testDevice(new Tv());
        testDevice(new Radio());
        testDevice(new SmartTv());
    }

    public static void testDevice(Device device) {
        System.out.println("Tests with basic remote.");
        BasicRemote basicRemote = new BasicRemote(device);
        basicRemote.power();
        device.printStatus();

        System.out.println("Tests with advanced remote.");
        AdvancedRemote advancedRemote = new AdvancedRemote(device);
        advancedRemote.power();
        advancedRemote.mute();
        device.printStatus();

        System.out.println("Tests with smart remote.");
        SmartRemote smartRemote = new SmartRemote(device);
        smartRemote.power();
        smartRemote.mute();
        smartRemote.voiceControl("channelup");
        if (device instanceof  SmartTv){
            ((SmartTv)device).browseInternet();
        }else {
            System.out.println("This device does not implement Internet
    browsing");
        }
    }
```

Here, I call the smartRemote.voiceControl() on each Device, Tv, Radio, SmartTv.

With TV

```
-----------------------------------
        | I'm TV set.
        | I'm disabled
        | Current volume is 0%
        | Current channel is 1
        -----------------------------------

voiceControl activated
Remote: channel up
```

With Radio

```
With Radio:
-----------------------------------
        | I'm radio.
        | I'm disabled
        | Current volume is 0%
        | Current channel is 1
        -----------------------------------

Remote: mute
voiceControl activated
Remote: channel up
```

With SmartTV

```
-----------------------------------
        | I'm smart TV set.
        | I'm disabled
        | Current volume is 0%
        | Current channel is 1
        -----------------------------------

voiceControl activated
Remote: channel up
```

The program able to run without any error when calling the voiceControl() method on each device.

7. **Conclusion:** Summarize the work done and the results obtained. Discuss any design decisions you had to make and any alternative options you considered.

Summary, I have implement the SmartRemote class as I want to extend the Remote class to have more functionality. And as I extend the Implementation class of the Bridge design Pattern, it does not cause any change or require change in the client code.

I have use inheritance when creating SmartRemote class as believe this class has more method than its parent class, AdvanceRemote