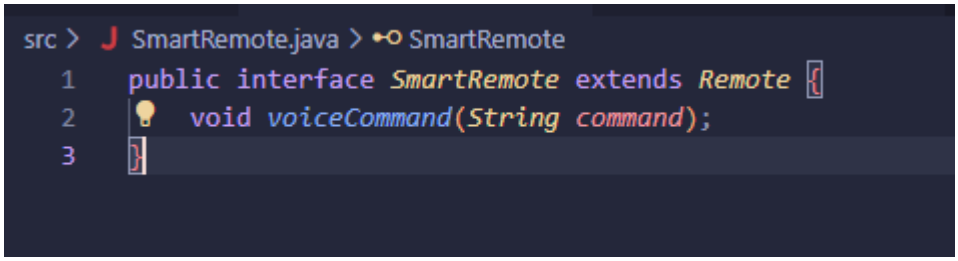
1. **Introduction:** The bridge pattern is a design pattern that separates an abstraction from its implementation. This Pattern is useful when you want to avoid permanent binding, making it possible to switch implementations at runtime.

   Selected implementation: https://refactoring.guru/design-patterns/bridge/java/example

   I choose the bridge implementation from refactoring guru. In this example, the bridge pattern is implemented by having remotes and devices that are controlled by them. While having the remotes as abstractions so that the same remotes can work with multiple devices.
   - Device interface to describe a device
   - Radio and Tv for concrete implementations
   - Remote interface for defining remote
   - BasicRemote and AdvancedRemote for more concrete implementations

2. **New Functionality:** I wanted to add more accessibility from the default tv and radio remotes with voice control support. With a method for parsing voice controls voiceCommand(String command);.

3. **Implementation:**




```java
public class VoiceRemote extends BasicRemote implements SmartRemote {

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

    @Override
    public void voiceCommand(String command) {
        System.out.println("Voice Remote: Processing voice command - '" + command + "'");

        // Parse and execute voice commands
        String lowerCommand = command.toLowerCase();

        if (lowerCommand.contains("turn on") || lowerCommand.contains("power on")) {
            if (!device.isEnabled()) {
                power();
            }
        } else if (lowerCommand.contains("turn off") || lowerCommand.contains("power off")) {
            if (device.isEnabled()) {
                power();
            }
        } else if (lowerCommand.contains("volume up")) {
            volumeUp();
        } else if (lowerCommand.contains("volume down")) {
            volumeDown();
        } else if (lowerCommand.contains("mute")) {
            device.setVolume(percent:0);
        } else if (lowerCommand.contains("channel")) {
            if (lowerCommand.contains("up") || lowerCommand.contains("next")) {
                channelUp();
            } else if (lowerCommand.contains("down") || lowerCommand.contains("previous")) {
                channelDown();
            }
        } else {
            System.out.println("Voice Remote: Command not recognized");
        }
    }

}
```

4. **Verification:** I tested the implementation through adding testing for a demo run with testVoiceDevice. This demo run demonstrates:
    - Backwards compatibility with original implementation
    - New remote functionality, new remote controls devices
    - Bridge pattern integrity, new components integrate with existing design

```
src > App.java > ...
1   public class App {
2       public static void testDevice(Device device) {
3           System.out.println(x:"Tests with basic remote.");
4           BasicRemote basicRemote = new BasicRemote(device);
5           basicRemote.power();
6           device.printStatus();
7
8           System.out.println(x:"Tests with advanced remote.");
9           AdvancedRemote advancedRemote = new AdvancedRemote(device);
10          advancedRemote.power();
11          advancedRemote.mute();
12          device.printStatus();
13      }
14
15      public static void testVoiceDevice(Device device) {
16          System.out.println(x:"Tests with Voice Remote on Smart TV.");
17
18          VoiceRemote voiceRemote = new VoiceRemote((Device) device);
19
20          voiceRemote.voiceCommand(command:"turn on");
21          device.printStatus();
22
23          voiceRemote.voiceCommand(command:"volume up");
24          device.printStatus();
25
26          voiceRemote.voiceCommand(command:"mute");
27          device.printStatus();
28
29          voiceRemote.voiceCommand(command:"channel next");
30          device.printStatus();
31
32          voiceRemote.voiceCommand(command:"turn off");
33          device.printStatus();
34
35      }
36
        Run | Debug
37      public static void main(String[] args) throws Exception {
38          System.out.println(x:"=== Testing Original Implementation ===");
39          testDevice(new Tv());
40          testDevice(new Radio());
41
42          System.out.println(x:"\n=== Testing Extended Implementation ===");
43          testVoiceDevice(new Tv());
44      }
45  }
46
```

Demo run results :

=== Testing Original Implementation ===

Tests with basic remote.

Remote: power toggle

| I'm TV set.

| I'm enabled

| Current volume is 30%

| Current channel is 1

Tests with advanced remote.

Remote: power toggle

Remote: mute

| I'm TV set.

| I'm disabled

| Current volume is 0%

| Current channel is 1

Tests with basic remote.

Remote: power toggle

| I'm radio.

| I'm enabled

| Current volume is 30%

| Current channel is 1

Tests with advanced remote.

Remote: power toggle

Remote: mute

| I'm radio.

| I'm disabled

| Current volume is 0%

| Current channel is 1

=== Testing Extended Implementation ===

Tests with Voice Remote on Smart TV.

Voice Remote: Processing voice command - 'turn on'

Remote: power toggle

| I'm TV set.

| I'm enabled

| Current volume is 30%

| Current channel is 1

Voice Remote: Processing voice command - 'volume up'

Remote: volume up

| I'm TV set.

| I'm enabled

| Current volume is 40%

| Current channel is 1

Voice Remote: Processing voice command - 'mute'

| I'm TV set.

| I'm enabled

| Current volume is 0%

| Current channel is 1

Voice Remote: Processing voice command - 'channel next'

Remote: channel up

| I'm TV set.

| I'm enabled

| Current volume is 0%

| Current channel is 2

Voice Remote: Processing voice command - 'turn off'

Remote: power toggle

| I'm TV set.

| I'm disabled

| Current volume is 0%

| Current channel is 2

5. **Conclusion:** Implementation of bridge pattern extension:
    - Added a new voiceRemote to use voice controls
    - Added voice command method to parse voice commands

    Design decisions

    - Created a simple interface for voiceRemote to make for easy simple integration and backwards compatibility while enabling new features, instead of integrating with old interface.
    - Implemented simple string parsing for voice commands for remote. Keeps the demo focused on the bridge pattern rather than complex language processing.

    Alternative options considered
    - Modifying original interface, breaks existing code.

- Complex voice processing, avoided to maintain focus on design pattern.