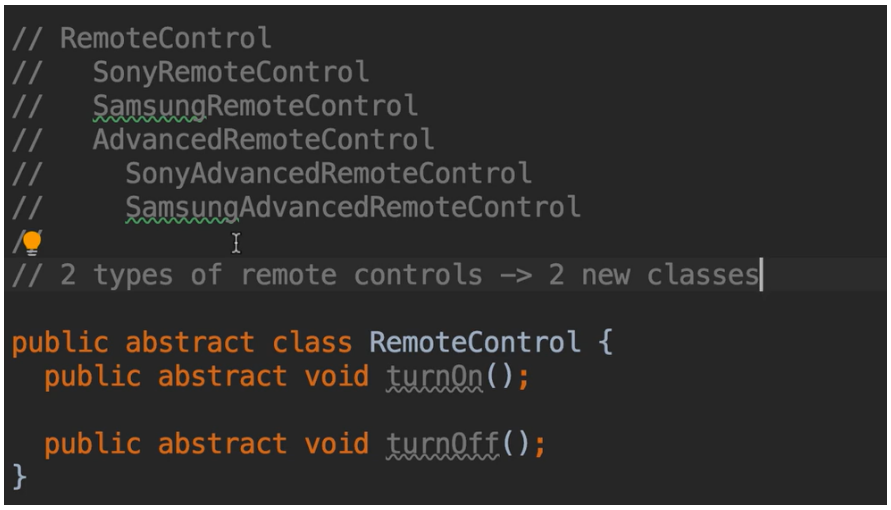
Bridge Pattern

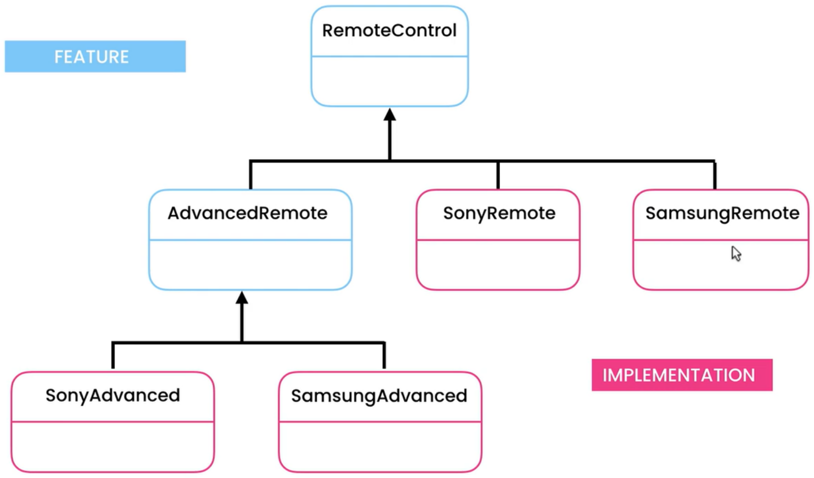
Allows us to develop a simple yet flexible hierarchy

Example Problem - We are developing a Remote Control, we also have an advanced remote control with extra features such as View Guide. Both are abstract classes and the Advanced class extends the Remote Control Class. there are then concrete implementation such as Sony Remote Control and Sony Advanced which utilise the manufacturers own libraries to speak with the hardware. Quickly the hierarchy can become compleax as we add concrete implementations and diferent type of remote controls :

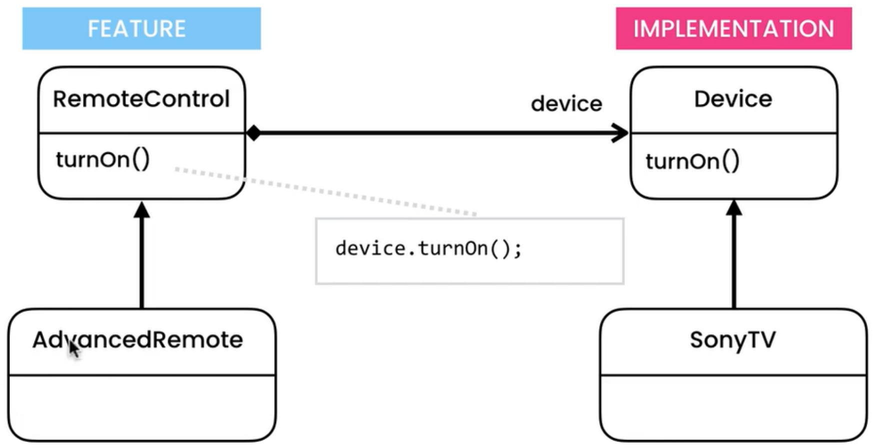


```
// RemoteControl
//   SonyRemoteControl
//   SamsungRemoteControl
//   AdvancedRemoteControl
//     SonyAdvancedRemoteControl
//     SamsungAdvancedRemoteControl
//
// 2 types of remote controls -> 2 new classes

public abstract class RemoteControl {
  public abstract void turnOn();

  public abstract void turnOff();
}
```

The Bridge Pattern helps solve this issue, by splitting the two dimension that our hierarch is growing along (features and implementations) :



Becomes :



By splitting the dimensions and adding a bridge.

```
public class RemoteControl {
  protected Device device;

 public RemoteControl(Device
device) {
    this.device = device;
 }

  public void turnOn() {
    device.turnOn();
 }

  public void turnOff() {
    device.turnOff();
 }
}
```

```
public interface
Device {
   void turnOn();
  void turnOff();
  void setChannel(
int number);
}
```

```
public class AdvancedRemoteControl ext
ends RemoteControl {
  public AdvancedRemoteControl(Device
device) {
    super(device);
 }

  public void setChannel(int number) {
    device.setChannel(number);
 }
}
```

```
public class SamsungTV
implements Device {
  @Override
  public void turnOn()
{
    System.out.println(
"Samsung: turnOn");
 }

  @Override
  public void turnOff()
{
    System.out.println(
"Samsung: turnOff");
 }

  @Override
  public void setChanne
l(int number) {
    System.out.println(
"Samsung: setChannel");
 }
}
```

```
public class SonyTV imp
lements Device {
  @Override
  public void turnOn()
{
    System.out.println(
"Sony: turnOn");
 }

  @Override
  public void turnOff()
{
    System.out.println(
"Sony: turnOff");
 }

  @Override
  public void setChanne
l(int number) {
    System.out.println(
"Sony: setChannel");
 }
}
```

```
public class Main {
    public static void main(String[] args) {
    var remoteConrol = new AdvanceRemoteControl(new SonyTV());
   }

}
```