16 Structural: Bridge Pattern for a Media-Player Framework

**Scenario** You are architecting a media framework that must support **multiple player types** (Audio, Video, LiveStream) and **multiple file formats / codecs** (MP3, MP4, FLAC, AVI). Management wants to mix-and-match any player with any format—at runtime—without a combinatorial explosion of subclasses or recompilation.

Your task is to apply the **Bridge pattern** so that:

* Each new player or format can be introduced in isolation.
* Clients can swap formats on an existing player object at runtime.

#### **Tasks**

1 **Design rationale** (analysis/bridge\_choice.md)

* Show the inheritance explosion that would result from naïve subclassing (AudioMp3Player, AudioMp4Player, …).
* Explain how Bridge eliminates that by decoupling “player” (abstraction) from “format decoder” (implementation).

2 **Code implementation** (src/main/java/clean/)

* **Implementation hierarchy**
  + MediaFormat (interface) + concrete Mp3Format, Mp4Format.
* **Abstraction hierarchy**
  + MediaPlayer (abstract class holding a MediaFormat)
  + Refined abstractions AudioPlayer, VideoPlayer.
* **Dynamic switch demo**
  + In BridgeDemo.java create an AudioPlayer with Mp3Format, play a track, swap to Mp4Format, play again.
* Ensure each play() prints player type then format message.

3 **Unit tests** (src/test/java/clean/)

* CrossCombinationTest – verify every player can accept every format without ClassCastException.
* RuntimeSwitchTest – give a player a new MediaFormat object and assert second play uses the new format.

4 **Reflection** (reflection.md)

* Benefit matrix: decoupling, scalability, runtime flexibility.
* Cost matrix: added abstraction, slight call-stack overhead.
* How to add a FlacFormat and an OnlineRadioPlayer tomorrow with zero edits to existing classes.

#### **Deliverables**

analysis/bridge\_choice.md

src/main/java/clean/format/MediaFormat.java

src/main/java/clean/format/Mp3Format.java

src/main/java/clean/format/Mp4Format.java

src/main/java/clean/player/MediaPlayer.java

src/main/java/clean/player/AudioPlayer.java

src/main/java/clean/player/VideoPlayer.java

src/main/java/clean/BridgeDemo.java

src/test/java/clean/CrossCombinationTest.java

src/test/java/clean/RuntimeSwitchTest.java

reflection.md

README.md

## **Detailed Solution**

### **1 Implementation Code**

/\* format/MediaFormat.java \*/

package clean.format;

public interface MediaFormat {

void play(String filename);

}

/\* format/Mp3Format.java \*/

package clean.format;

public class Mp3Format implements MediaFormat {

public void play(String filename){

System.out.println("Playing MP3 file: "+filename);

}

}

/\* format/Mp4Format.java \*/

package clean.format;

public class Mp4Format implements MediaFormat {

public void play(String filename){

System.out.println("Playing MP4 file: "+filename);

}

}

/\* player/MediaPlayer.java \*/

package clean.player;

import clean.format.MediaFormat;

public abstract class MediaPlayer {

protected MediaFormat format;

protected MediaPlayer(MediaFormat f){ this.format = f; }

public void setFormat(MediaFormat f){ this.format = f; } // runtime swap

public abstract void play(String filename);

}

/\* player/AudioPlayer.java \*/

package clean.player;

import clean.format.MediaFormat;

public class AudioPlayer extends MediaPlayer {

public AudioPlayer(MediaFormat f){ super(f); }

@Override public void play(String filename){

System.out.println("Audio Player in action…");

format.play(filename);

}

}

/\* player/VideoPlayer.java \*/

package clean.player;

import clean.format.MediaFormat;

public class VideoPlayer extends MediaPlayer {

public VideoPlayer(MediaFormat f){ super(f); }

@Override public void play(String filename){

System.out.println("Video Player in action…");

format.play(filename);

}

}

/\* BridgeDemo.java \*/

package clean;

import clean.format.\*;

import clean.player.\*;

public class BridgeDemo {

public static void main(String[] args){

MediaFormat mp3 = new Mp3Format();

MediaFormat mp4 = new Mp4Format();

MediaPlayer audio = new AudioPlayer(mp3);

audio.play("song.mp3");

// switch format on the same player instance

audio.setFormat(mp4);

audio.play("lecture.mp4");

MediaPlayer video = new VideoPlayer(mp4);

video.play("movie.mp4");

}

}

**Console output**

Audio Player in action…

Playing MP3 file: song.mp3

Audio Player in action…

Playing MP4 file: lecture.mp4

Video Player in action…

Playing MP4 file: movie.mp4

### **2 Test Highlights**

/\* CrossCombinationTest \*/

@ParameterizedTest

@MethodSource("allCombos")

void everyPlayerPlaysEveryFormat(MediaPlayer player,String file){

assertDoesNotThrow(() -> player.play(file));

}

static Stream<Arguments> allCombos(){

MediaFormat mp3 = new Mp3Format();

MediaFormat mp4 = new Mp4Format();

return Stream.of(

Arguments.of(new AudioPlayer(mp3),"a.mp3"),

Arguments.of(new AudioPlayer(mp4),"a.mp4"),

Arguments.of(new VideoPlayer(mp3),"v.mp3"),

Arguments.of(new VideoPlayer(mp4),"v.mp4"));

}

/\* RuntimeSwitchTest \*/

@Test void swapFormatAtRuntime(){

MediaPlayer audio = new AudioPlayer(new Mp3Format());

audio.play("one.mp3");

audio.setFormat(new Mp4Format());

assertDoesNotThrow(() -> audio.play("two.mp4"));

}

### **3 Reflection (excerpt)**

• Abstraction hierarchy (AudioPlayer, VideoPlayer) and implementation hierarchy

(Mp3Format, Mp4Format) vary independently → open/closed.

• New format FlacFormat adds one class; all players gain it automatically.

• Overhead: one extra indirection (player → format), negligible for I/O-bound tasks.

The Bridge pattern delivers a **flexible, scalable** media framework where new player types or codecs appear as single, isolated classes, without rewiring the rest of the system.