20 Challenge 3- Bridge Pattern for a Pluggable Streaming Platform

**Scenario** Your new streaming service must combine **many video containers** (MP4, MKV, AVI, WebM…) with **many delivery protocols** (HTTP-progressive, HLS, RTMP, DASH…). You need to add a container or protocol in the future with **zero changes** to existing classes.

The classic inheritance approach (Mp4OverHttpPlayer, Mp4OverHlsPlayer, MkvOverHttpPlayer …) explodes combinatorially.  
 Apply the **Bridge pattern** so that:

* A *format* abstraction (MP4, MKV…) can pair with any *protocol* implementation (HTTP, RTMP…).
* Either side can be extended independently.
* A client can even swap protocols at runtime.

#### **Tasks**

1. **Design rationale** (analysis/bridge\_streaming\_need.md)  
    *Illustrate the class-explosion problem with a quick diagram, then outline how the Bridge decouples “format” from “protocol.”*

**Implementation** (src/main/java/streaming/)  
 *Implementation hierarchy* protocol/

StreamingProtocol.java ← implementation interface

HttpProtocol.java

HlsProtocol.java

RtmpProtocol.java

*Abstraction hierarchy* format/

VideoFormat.java ← abstract class, holds StreamingProtocol

Mp4Format.java

MkvFormat.java

1. *Requirements*
   * VideoFormat exposes play(String title) and setProtocol(StreamingProtocol p) for runtime swaps.
   * Each concrete protocol prints "Streaming {title} via {PROTO}".
   * Each concrete format prints "Preparing <FORMAT> container" then delegates to its StreamingProtocol.
2. **Demo** (BridgeDemo.java)  
    *Create an Mp4Format over HttpProtocol, play a movie, switch to HlsProtocol, play again; create an MkvFormat over RtmpProtocol, play a clip.*
3. **Unit tests** (src/test/java/streaming/)  
   * EveryComboTest – iterate all combinations (2 formats × 3 protocols), ensure no exception.
   * RuntimeSwapTest – verify setProtocol really changes the underlying protocol call.
4. **Reflection** (reflection.md)  
    \*Summarise Bridge benefits, call-stack overhead, and how to add a new DashProtocol tomorrow.

#### **Deliverables**

analysis/bridge\_streaming\_need.md

src/main/java/streaming/protocol/StreamingProtocol.java

src/main/java/streaming/protocol/HttpProtocol.java

src/main/java/streaming/protocol/HlsProtocol.java

src/main/java/streaming/protocol/RtmpProtocol.java

src/main/java/streaming/format/VideoFormat.java

src/main/java/streaming/format/Mp4Format.java

src/main/java/streaming/format/MkvFormat.java

src/main/java/streaming/BridgeDemo.java

src/test/java/streaming/EveryComboTest.java

src/test/java/streaming/RuntimeSwapTest.java

reflection.md

README.md

## **Detailed Solution**

### **1 Implementation Code**

/\* protocol/StreamingProtocol.java \*/

package streaming.protocol;

public interface StreamingProtocol {

void stream(String title);

}

/\* protocol/HttpProtocol.java \*/

package streaming.protocol;

public class HttpProtocol implements StreamingProtocol {

public void stream(String title){

System.out.println("Streaming \"" + title + "\" via HTTP-progressive");

}

}

/\* protocol/HlsProtocol.java \*/

package streaming.protocol;

public class HlsProtocol implements StreamingProtocol {

public void stream(String title){

System.out.println("Streaming \"" + title + "\" via HLS");

}

}

/\* protocol/RtmpProtocol.java \*/

package streaming.protocol;

public class RtmpProtocol implements StreamingProtocol {

public void stream(String title){

System.out.println("Streaming \"" + title + "\" via RTMP");

}

}

/\* format/VideoFormat.java \*/

package streaming.format;

import streaming.protocol.StreamingProtocol;

/\*\* Abstraction – holds a protocol implementor. \*/

public abstract class VideoFormat {

protected StreamingProtocol protocol;

protected VideoFormat(StreamingProtocol p){ this.protocol = p; }

public void setProtocol(StreamingProtocol p){ this.protocol = p; }

public abstract void play(String title);

}

/\* format/Mp4Format.java \*/

package streaming.format;

import streaming.protocol.StreamingProtocol;

public class Mp4Format extends VideoFormat {

public Mp4Format(StreamingProtocol p){ super(p); }

@Override public void play(String title){

System.out.println("Preparing MP4 container");

protocol.stream(title);

}

}

/\* format/MkvFormat.java \*/

package streaming.format;

import streaming.protocol.StreamingProtocol;

public class MkvFormat extends VideoFormat {

public MkvFormat(StreamingProtocol p){ super(p); }

@Override public void play(String title){

System.out.println("Preparing MKV container");

protocol.stream(title);

}

}

/\* BridgeDemo.java \*/

package streaming;

import streaming.format.\*;

import streaming.protocol.\*;

public class BridgeDemo {

public static void main(String[] args){

VideoFormat mp4 = new Mp4Format(new HttpProtocol());

mp4.play("Inception");

mp4.setProtocol(new HlsProtocol()); // runtime switch

mp4.play("Interstellar");

VideoFormat mkv = new MkvFormat(new RtmpProtocol());

mkv.play("Tutorial.mkv");

}

}

**Console Output**

Preparing MP4 container

Streaming "Inception" via HTTP-progressive

Preparing MP4 container

Streaming "Interstellar" via HLS

Preparing MKV container

Streaming "Tutorial.mkv" via RTMP

### **2 Test Highlights (JUnit 5)**

/\* EveryComboTest \*/

@ParameterizedTest

@MethodSource("combos")

void playerWorks(VideoFormat format,String title){

assertDoesNotThrow(() -> format.play(title));

}

static Stream<Arguments> combos(){

return Stream.of(

new Mp4Format(new HttpProtocol()),

new Mp4Format(new HlsProtocol()),

new Mp4Format(new RtmpProtocol()),

new MkvFormat(new HttpProtocol()),

new MkvFormat(new HlsProtocol()),

new MkvFormat(new RtmpProtocol())

).map(f -> Arguments.of(f,"demo"));

}

/\* RuntimeSwapTest \*/

@Test void swapProtocol(){

VideoFormat mp4 = new Mp4Format(new HttpProtocol());

ByteArrayOutputStream out=new ByteArrayOutputStream();

System.setOut(new PrintStream(out));

mp4.play("x"); // HTTP

mp4.setProtocol(new HlsProtocol());

mp4.play("y"); // HLS

String s=out.toString();

assertTrue(s.contains("HTTP") && s.contains("HLS"));

}

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

* **Decoupling** – Adding DashProtocol or AviFormat requires one class each; all combinations work immediately.
* **Runtime flexibility** – setProtocol() lets an adaptive-bitrate algorithm switch protocols on the fly.
* **Cost** – One extra indirection (format → protocol), negligible versus network latency.

The **Bridge pattern** perfectly matches a matrix of video containers × protocols, keeping the codebase extensible and combinatorially lean.