

HTMLCSSJavascript

LEARN JAVA DESIGN PATTERNS

problem solving approach

Design Patterns Tutorial

Design Patterns - Home

Design Patterns - Overview

Design Patterns - Factory Pattern

Abstract Factory Pattern

Design Patterns - Singleton Pattern

Design Patterns - Builder Pattern

Design Patterns - Prototype Pattern

Design Patterns - Adapter Pattern

Design Patterns - Bridge Pattern

Design Patterns - Filter Pattern

Design Patterns - Composite Pattern

Design Patterns - Decorator Pattern

Design Patterns - Facade Pattern

Design Patterns - Flyweight Pattern

Design Patterns - Proxy Pattern

Chain of Responsibility Pattern

Design Patterns - Command Pattern

Design Patterns - Interpreter Pattern

Design Patterns - Iterator Pattern

Design Patterns - Mediator Pattern

Design Patterns - Memento Pattern

Design Patterns - Observer Pattern

Design Patterns - State Pattern

Design Patterns - Null Object Pattern

Design Patterns - Strategy Pattern

Design Patterns - Template Pattern

Design Patterns - Visitor Pattern

Design Patterns - MVC Pattern

Business Delegate Pattern

Composite Entity Pattern

Data Access Object Pattern

Front Controller Pattern

Intercepting Filter Pattern

Service Locator Pattern

Transfer Object Pattern

Design Patterns Resources

Design Patterns - Questions/Answers

Design Patterns - Quick Guide

Design Patterns - Useful Resources

Design Patterns - Discussion

Selected Reading

UPSC IAS Exams Notes

Developer's Best Practices

Questions and Answers

Effective Resume Writing

HR Interview Questions

Computer Glossary

Who is Who

Development

Computer Programming

DevOps

Exams Syllabus

Latest Technologies

Management Tutorials

Misc tutorials

Python Technologies

Selected Reading

Telecom Tutorials

Sports Tutorials

Interview Questions

Academic Tutorials

Computer Science

Digital Marketing

Famous Monuments

Machine Learning

Mathematics Tutorials

Mobile Development

SAP Tutorials

Software Quality

UPSC IAS Exams

XML Technologies

Big Data & Analytics

Databases

Engineering Tutorials

GATE Exams

Mainframe Development

Microsoft Technologies

Java Technologies

Programming Scripts

Soft Skills

Web Development

Multi-Language

# Adapter Pattern

The Adapter Pattern is a structural design pattern that allows classes to work together that would otherwise be incompatible. It acts as a bridge between two incompatible interfaces.

It is used when:

- There is a target interface that defines a set of methods.
- There is a client that depends on the target interface.
- There is an existing class that implements a different interface.
- The client needs to use the existing class, but it cannot because the interfaces are incompatible.

The Adapter Pattern solves this by creating an adapter class that implements the target interface and delegates the calls to the existing class.

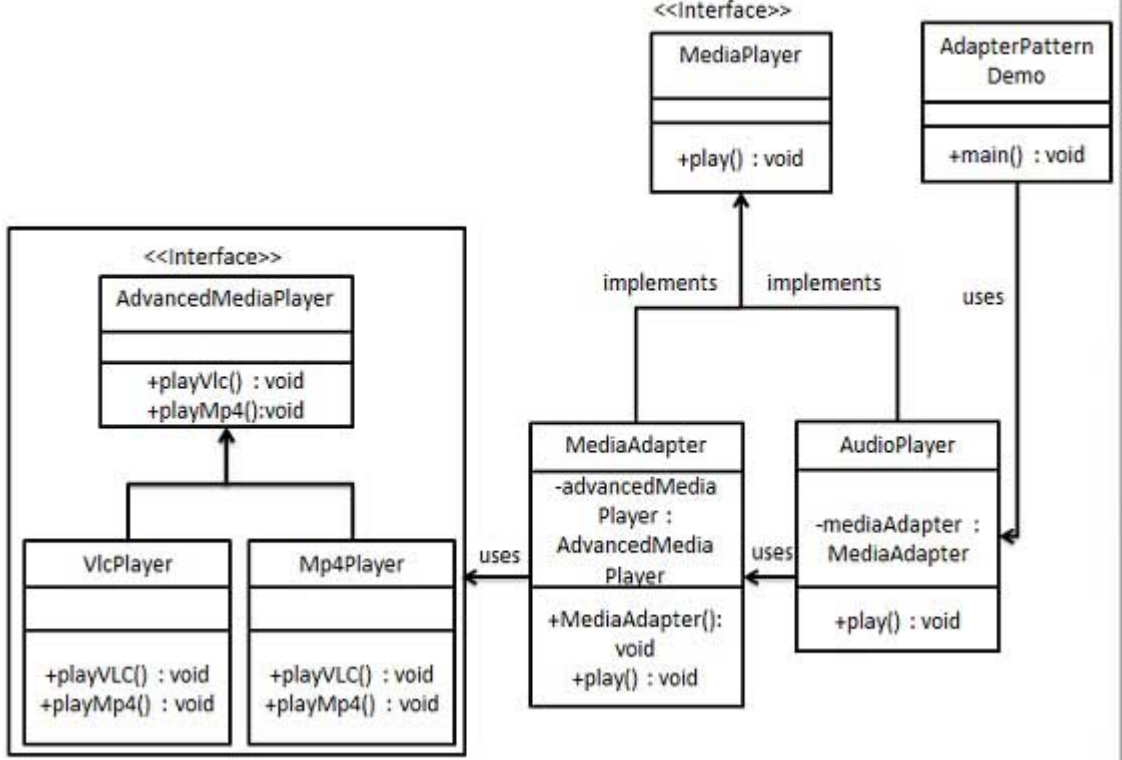
## Implementation

We have a MediaPlayer interface and a concrete class AudioPlayer implementing the MediaPlayer interface. AudioPlayer can play mp3 format audio files by default.

We are having another interface AdvancedMediaPlayer and concrete classes implementing the AdvancedMediaPlayer interface. These classes can play vlc and mp4 format files.

We want to make AudioPlayer to play other formats as well. To attain this, we have created an adapter class MediaAdapter which implements the MediaPlayer interface and uses AdvancedMediaPlayer objects to play the required format.

AudioPlayer uses the adapter class MediaAdapter passing it the desired audio type without knowing the actual class which can play the desired format. AdapterPatternDemo, our demo class will use AudioPlayer class to play various formats.



## Step 1

Create interfaces for Media Player and Advanced Media Player.

MediaPlayer.java

```
public interface MediaPlayer {
    public void play(String audioType, String fileName);
}
```

AdvancedMediaPlayer.java

```
public interface AdvancedMediaPlayer {
    public void playVlc(String fileName);
    public void playMp4(String fileName);
}
```

## Step 2

Create concrete classes implementing the AdvancedMediaPlayer interface.

VlcPlayer.java

```
public class VlcPlayer implements AdvancedMediaPlayer{
    @Override
    public void playVlc(String fileName) {
        System.out.println("Playing vlc file. Name: " + fileName);
    }

    @Override
    public void playMp4(String fileName) {
        //do nothing
    }
}
```

Mp4Player.java

```
public class Mp4Player implements AdvancedMediaPlayer{

    @Override
    public void playVlc(String fileName) {
        //do nothing
    }

    @Override
    public void playMp4(String fileName) {
        System.out.println("Playing mp4 file. Name: " + fileName);
    }
}
```

## Step 3

Create adapter class implementing the MediaPlayer interface.

MediaAdapter.java

```
public class MediaAdapter implements MediaPlayer {

    AdvancedMediaPlayer advancedMusicPlayer;

    public MediaAdapter(String audioType){

        if(audioType.equalsIgnoreCase("vlc") ){
            advancedMusicPlayer = new VlcPlayer();
        }else if (audioType.equalsIgnoreCase("mp4")){
            advancedMusicPlayer = new Mp4Player();
        }
    }

    @Override
    public void play(String audioType, String fileName) {

        if(audioType.equalsIgnoreCase("vlc"){
            advancedMusicPlayer.playVlc(fileName);
        }
        else if(audioType.equalsIgnoreCase("mp4")){
            advancedMusicPlayer.playMp4(fileName);
        }
    }
}
```

## Step 4

Create concrete class implementing the MediaPlayer interface.

AudioPlayer.java

```
public class AudioPlayer implements MediaPlayer {
    MediaAdapter mediaAdapter;

    @Override
    public void play(String audioType, String fileName) {

        //inbuilt support to play mp3 music files
        if(audioType.equalsIgnoreCase("mp3")){
            System.out.println("Playing mp3 file. Name: " + fileName);
        }

        //mediaAdapter is providing support to play other file formats
        else if(audioType.equalsIgnoreCase("vlc") || audioType.equalsIgnoreCase("mp4")){
            mediaAdapter = new MediaAdapter(audioType);
            mediaAdapter.play(audioType, fileName);
        }

        else{
            System.out.println("Invalid media. " + audioType + " format not supported");
        }
    }
}
```

## Step 5

Use the AudioPlayer to play different types of audio formats.

AdapterPatternDemo.java

```
public class AdapterPatternDemo {
    public static void main(String[] args) {
        AudioPlayer audioPlayer = new AudioPlayer();

        audioPlayer.play("mp3", "beyond the horizon.mp3");
        audioPlayer.play("mp4", "alone.mp4");
        audioPlayer.play("vlc", "far far away.vlc");
        audioPlayer.play("avi", "mind me.avi");
    }
}
```

## Step 6

Verify the output.

```
Playing mp3 file. Name: beyond the horizon.mp3
Playing mp4 file. Name: alone.mp4
Playing vlc file. Name: far far away.vlc
Invalid media. avi format not supported
```

Kickstart Your Career

Get certified by completing the course

Get Started

Print Page

PreviousNext

Advertisements

tutorialspoint

Tutorials Point is a leading Ed Tech company striving to provide the best learning material on technical and non-technical subjects.

GET IT ON Google Play

Download on the App Store

About us

Company

Our Team

Careers

Jobs

Become a Teacher

Affiliates

Contact Us

Terms

Terms of use

Privacy Policy

Refund Policy

Cookies Policy

FAQ's

Our Products

Free Library

Articles

Coding Ground

Certifications

Courses

eBooks

Corporate Training

Free Web Graphics

Contact Us

Tutorials Point India Private Limited, Incor9 Building, Kavuri Hills, Madhapur, Hyderabad, Telangana - 500081, INDIA

FacebookInstagramTwitterYouTubeLinkedIn