Centro Universitário UniBH
Ciência da Computação
Práticas de Programação
Professor: Lucas Schmidt

Aula Prática - 6 Padrões Estruturais - Adapter

Intenção

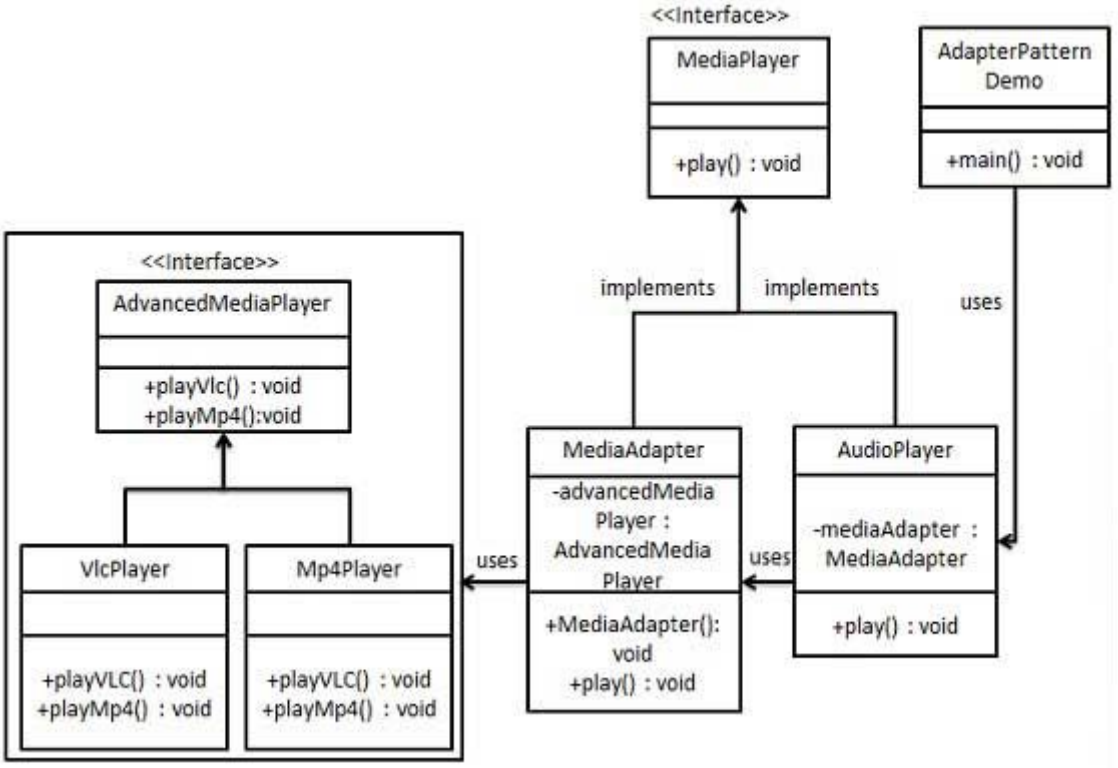
Converter a interface de uma classe em outra interface esperada pelo cliente. Permite que classes com interfaces incompatíveis possam colaborar.

Usar este padrão quando…

- Você quer usar uma classe já pronta que possui uma interface diferente da que você precisa;
- Você quer criar uma classe reutilizável já prevendo que a situação acima ocorrerá no futuro.

Vantagens e desvantagens

- Adapter de Classe:
  - Não funciona bem quando se quer adaptar uma hierarquia de classes;
  - Permite que o adaptador sobrescreva algumas funções do adaptado.
- Adapter de Objeto:
  - Permite o uso de um único adaptador para uma hierarquia de classes adaptadas;
  - É mais difícil sobrescrever funções do adaptado.



Passo 1
Crie interfaces para o Media Player e o 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);
}
```

Passo 2
Crie classes concretas implementando a interface AdvancedMediaPlayer.

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);
   }
}
```

Passo 3

Crie uma classe de adaptador implementando a interface do MediaPlayer.

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);
      }
   }
}
```

Passo 4

Crie uma classe concreta implementando a interface do MediaPlayer.

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");
      }
   }
}
```

Passo 5

Use o AudioPlayer para reproduzir diferentes tipos de formatos de áudio.

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");
   }
}
```

Passo 6

Teste sua implementação!