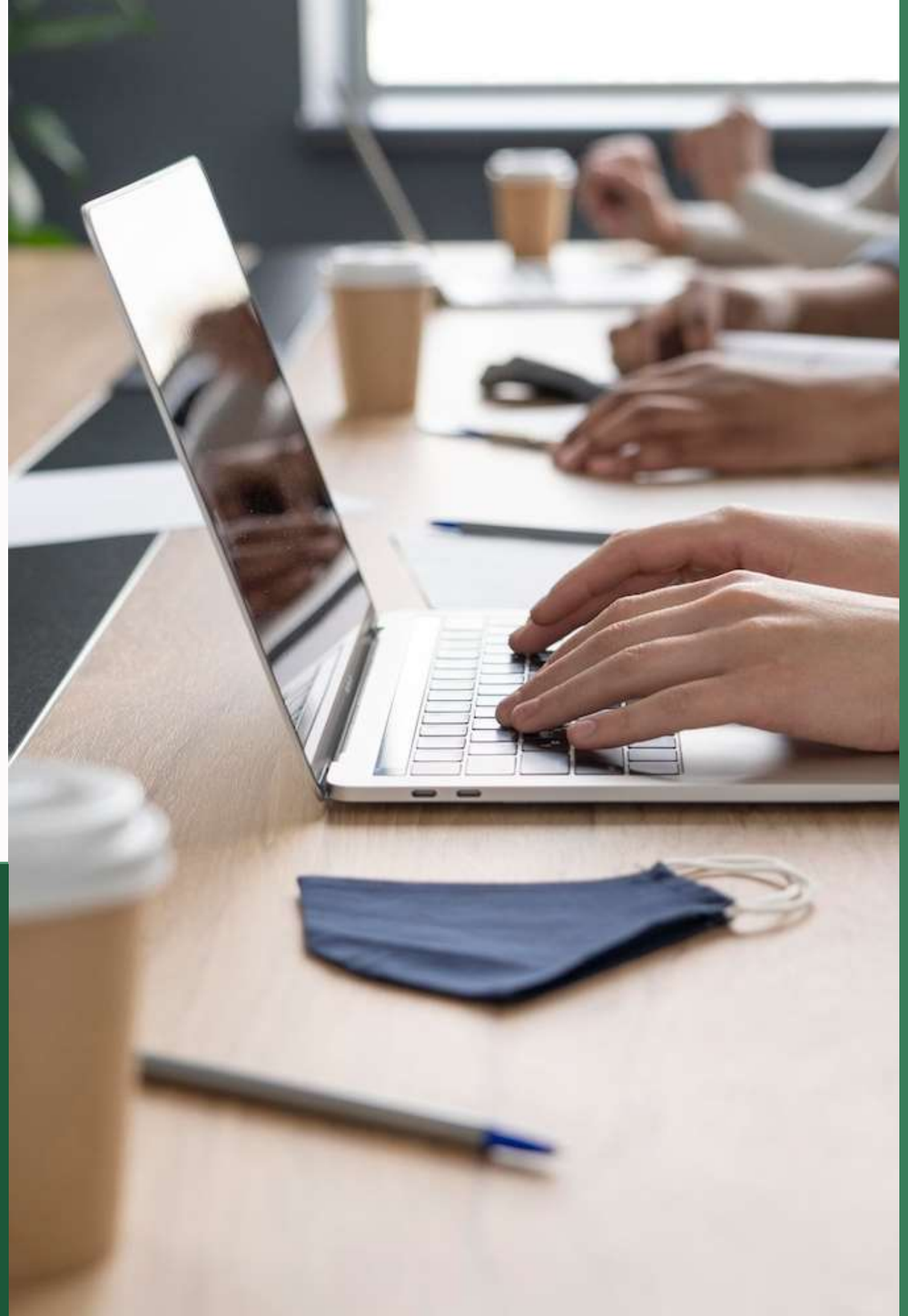
Welcome to Our Presentation



Adapter Design Pattern

Content

01 what is Adapter Design pattern

02 Components of Adapter Design Pattern

03 How Adapter Design Pattern work

04 Why do we need an Adapter Design Pattern

05 When not to use the Adapter Design Pattern

06 ULM Diagram

07 source Code

08 Advantages and Disadvantages

This presentation is presented to

Adiba Masud
Lecturer
Department of Software Engineering
Daffodil International University

Our Team

Md Rufsanjani Shanto

ID: 221-35-1064

Syeda Sadika Anjum

ID: 221-35-1066

MD. Habibul Basar

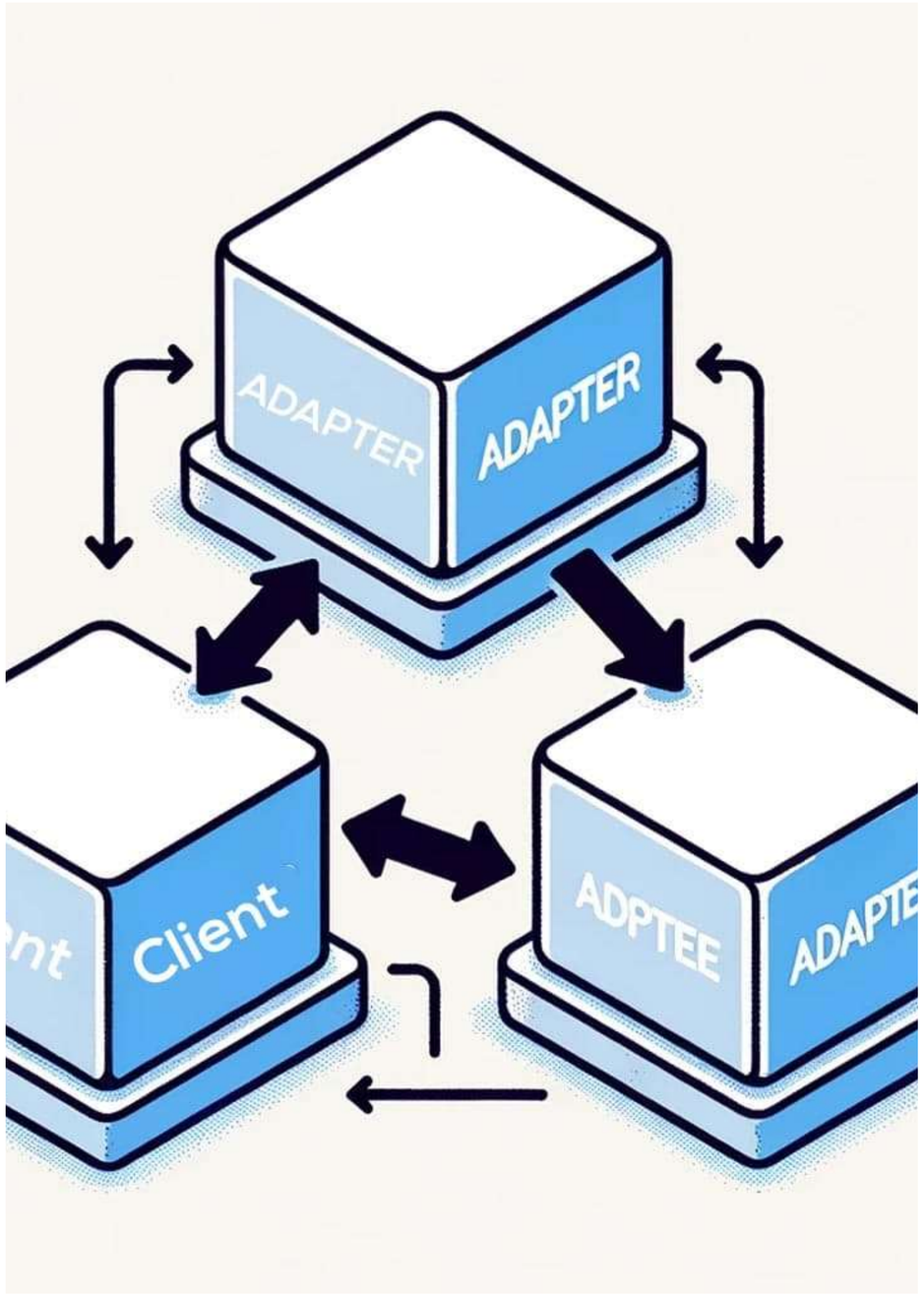
ID: 221-35-886

Razin Saleh

ID: 221-35-1055

Treebany Rani Saha

ID: 221-35-1061



What is Adapter Design pattern

Adapter design pattern is one of the structural design pattern and it is used so that two unrelated interfaces can work together.

Components of Adapter Design Pattern

- ***Target Interface***:
  It's the common interface that the client code interacts with.
- ***Adaptee:***
  It's the class or system that the client code cannot directly used due to interface mismatches.
- ***Adapter:***
  It acts as a bridge, adapting the interface of the adaptee to match the target interface.
- ***Client:***
  It's the code that benefits from the integration of the adaptee into the system through the adapter.

....

How Adapter Design Pattern work

- ***Client Request:***
  The client initiates a request by calling a method on the adapter using the target interface.
- ***Adapter Translation:***
  The adapter translates or maps the client's request into a form that the adaptee understands.
- ***Adaptee Execution:***
  The adaptee performs the actual work based on the translated the request from the adapter.
- ***Result to Client:***
  The client receives the results of the call, remaining unaware of the adapter's presence or the specific details of the adaptee.

Why do we need an Adapter Design Pattern

- *Integration of Existing Code*
- *Reuse of Existing Functionality*
- *Interoperability*
- *Client-Server Communication*
- *Third-Party Library Integration*

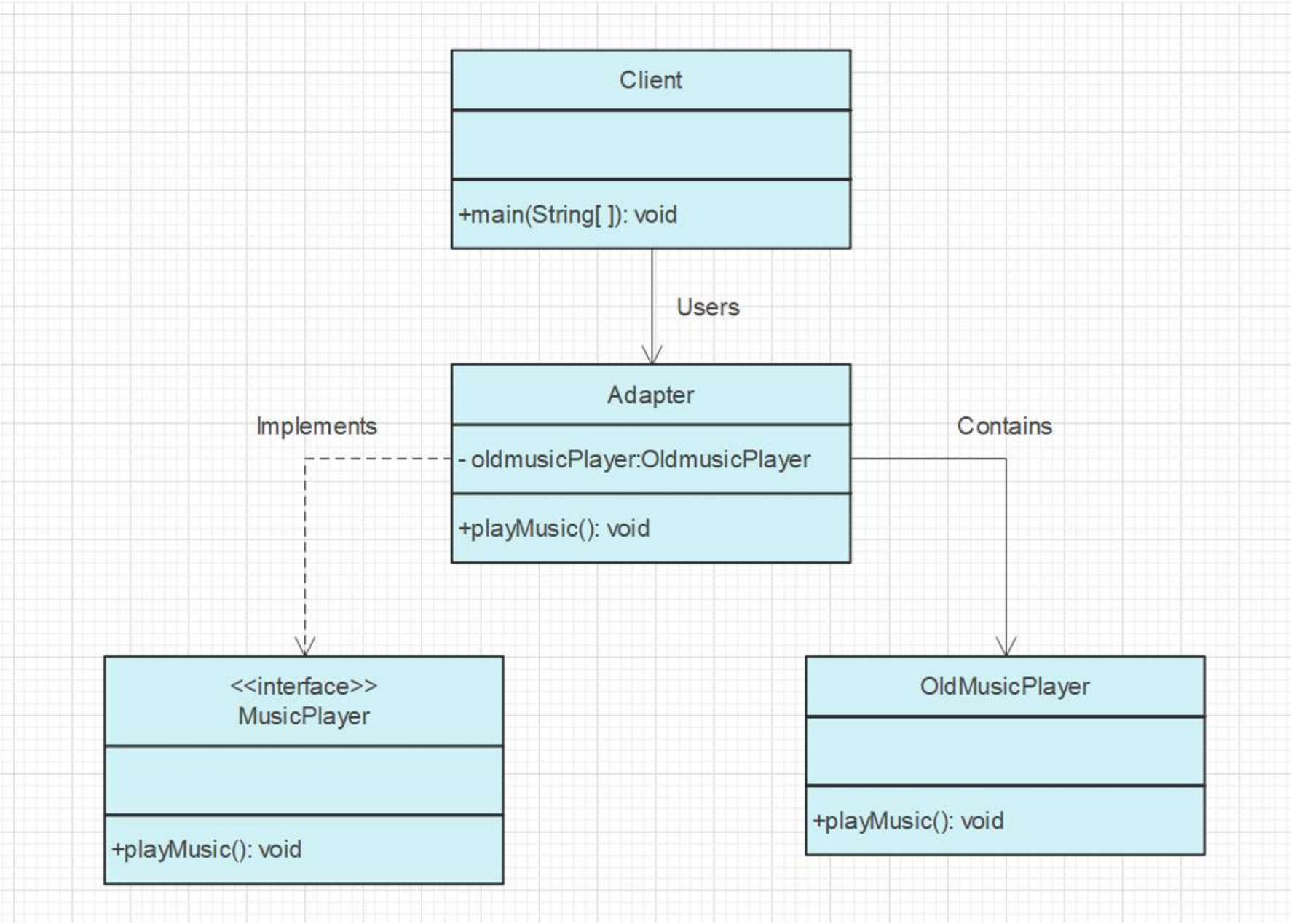
....

When not to use the Adapter Design Pattern

- *When Interfaces Are Stable*
- *When Direct Modification Is Feasible*
- *When Performance is Critical*
- *When Multiple Adapters Are Required*
- *When Adapters Introduce Ambiguity*

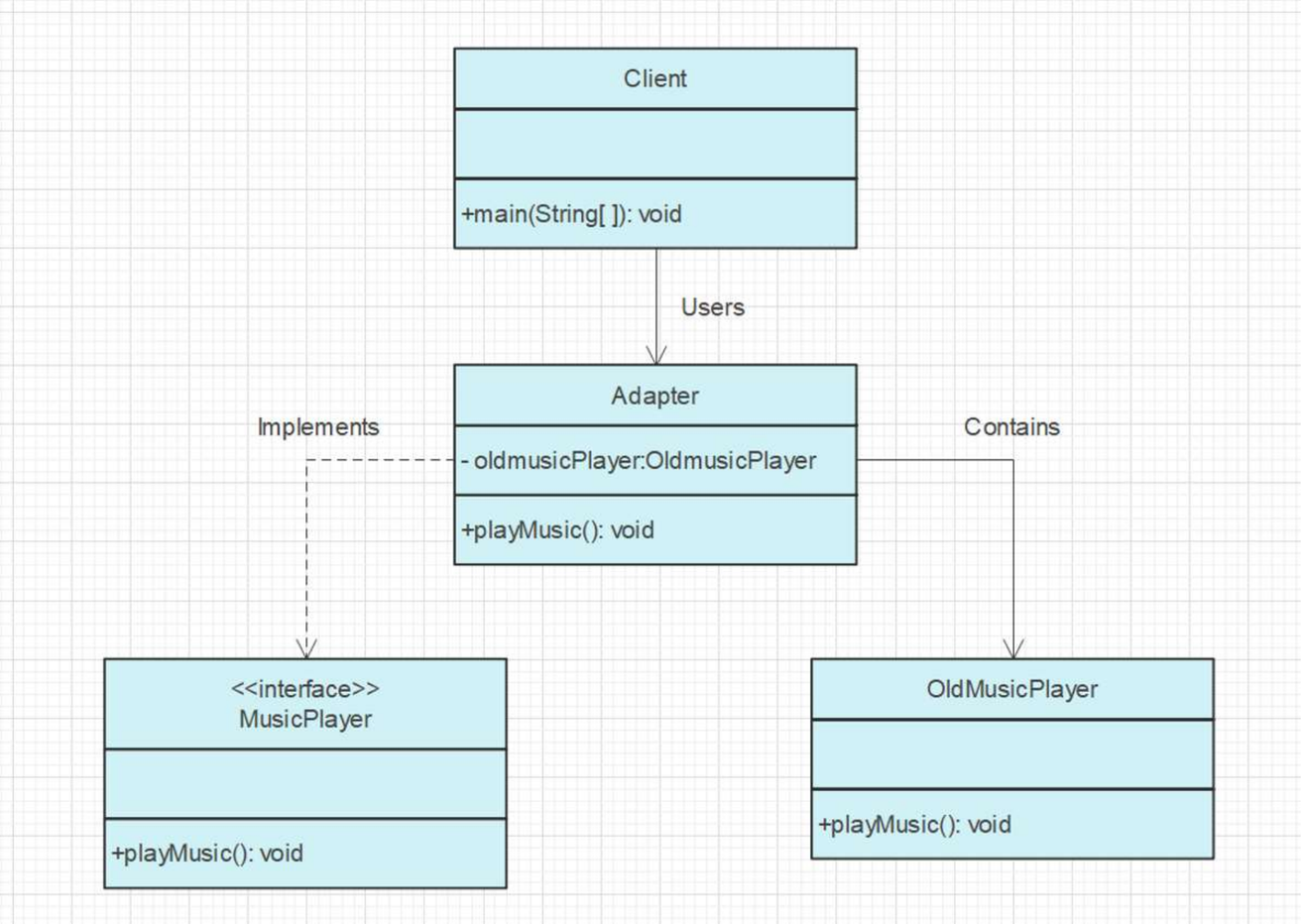
....

UML Diagram


Client
+main(String[ ]): void
Users
Adapter
- oldmusicPlayer:OldmusicPlayer
+playMusic(): void
Implements
Contains
<<interface>>
MusicPlayer
+playMusic(): void
OldMusicPlayer
+playMusic(): void

Code

```
// MusicPlayer interface
interface MusicPlayer {
  void playMusic();
}

// OldMusicPlayer class with incompatible interface
class OldMusicPlayer {
  void playOldMusic() {
    System.out.println("Playing old music...");
  }
}

// Adapter class
class Adapter implements MusicPlayer {
  private OldMusicPlayer oldMusicPlayer;

  public Adapter(OldMusicPlayer oldMusicPlayer) {
    this.oldMusicPlayer = oldMusicPlayer;
  }

  @Override
  public void playMusic() {
    oldMusicPlayer.playOldMusic();
  }
}

// Client
public class Client {
  public static void main(String[] args) {
    OldMusicPlayer oldPlayer = new OldMusicPlayer();
    MusicPlayer adapter = new Adapter(oldPlayer);
    adapter.playMusic();
  }
}
```



Advantages of the Adapter Design Pattern

- *Single Responsibility Principle (SRP)*
- *Open/Closed Principle (OCP)*
- *Compatibility and Integration*
- *Code Reuse*
- *Flexibility and Maintainability*
- *Unit Testing and Mocking*

....

Disadvantages of the Adapter Design Pattern

- *Complexity*
- *Overuse*
- *Performance Overhead*
- *Maintenance Overhead*
- *Increased Indirection*
- *Code Duplication*
- *Misuse of Legacy Code*
- *Not Suitable for All Cases*
- *Testing Complexities*

....

THANK YOU