C# Design Patterns: Adapter

APPLYING THE ADAPTER PATTERN

Steve Smith
FORCE MULTIPLIER FOR DEV TEAMS
@ardalis | ardalis.com | weeklydevtips.com

Objectives

What problems does adapter solve?

How is the adapter pattern structured?

Apply the pattern in real code

Recognize related patterns

Problem:
Incompatible interfaces between a client and a service-provider.

Electrical outlets provide electricity

United States' outlets have a specific interface and voltage

Other countries have differing interfaces

Appliances with one kind of plug may not be able to (safely) use incompatible outlets

An adapter is used to allow devices with incompatible interfaces to work together

A specific adapter works between two specific interfaces

Adapters convert the interface of one class into an interface a client expects.

Demo

The Problem

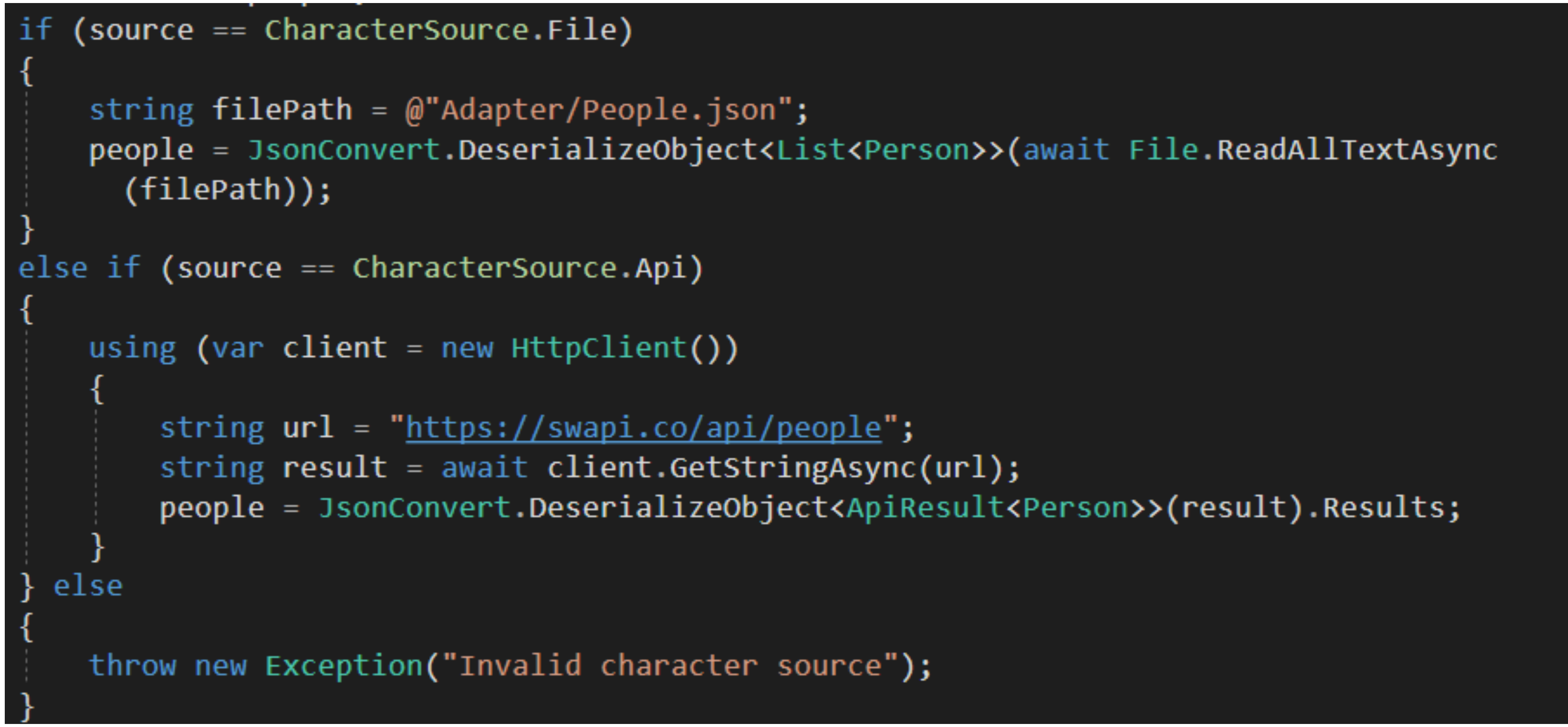
Working with differing interfaces

Related Principles

Some principles suggest the use of an Adapter as the solution in certain cases.

Polymorphism

Avoid complex conditional logic by using polymorphism



```
if (source == CharacterSource.File)
{
    string filePath = @"Adapter/People.json";
    people = JsonConvert.DeserializeObject<List<Person>>(await File.ReadAllTextAsync
      (filePath));
}
else if (source == CharacterSource.Api)
{
    using (var client = new HttpClient())
    {
        string url = "https://swapi.co/api/people";
        string result = await client.GetStringAsync(url);
        people = JsonConvert.DeserializeObject<ApiResult<Person>>(result).Results;
    }
} else
{
    throw new Exception("Invalid character source");
}
```

Applicable SOLID Principles

Single Responsibility Principle

Interface Segregation Principle

Open/Closed Principle

Small, focused interfaces are easiest to wrap with Adapters

Once client depends on adapter abstraction, it no longer needs to be changed

Adapters

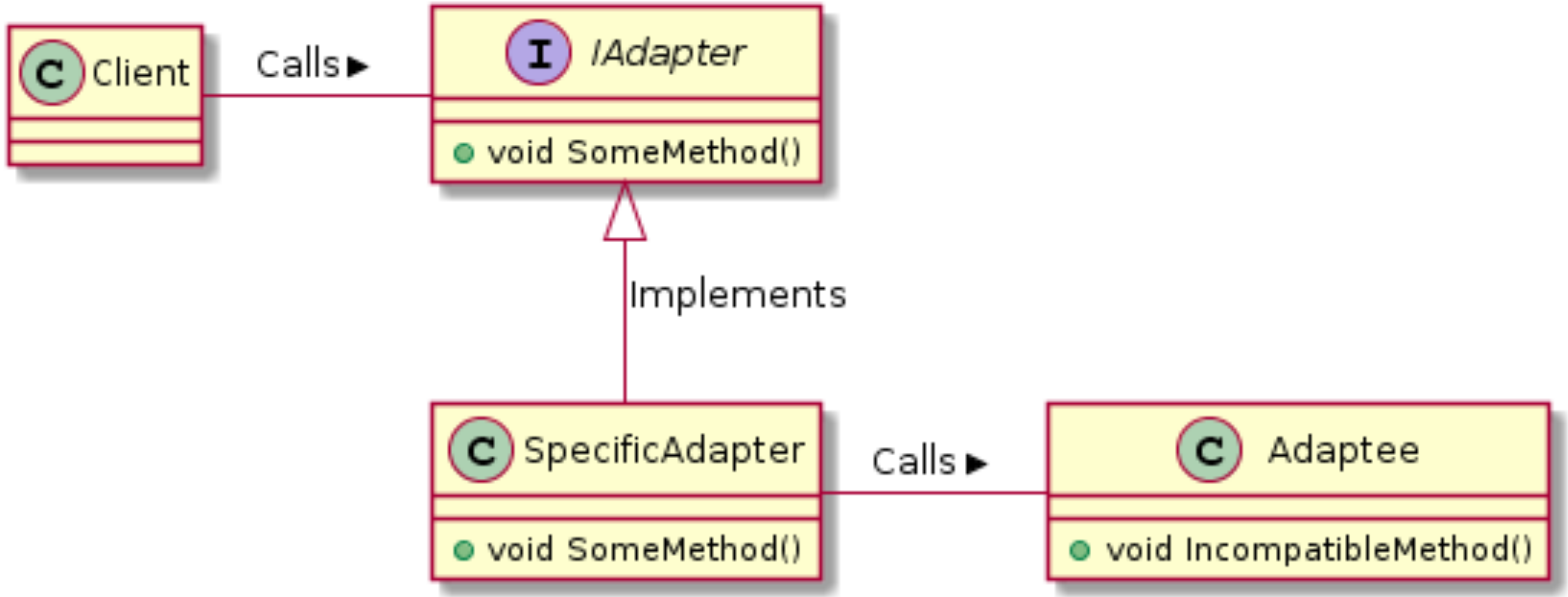
Two Kinds of Adapters

Object Adapters	Class Adapters
Hold an instance of the Adaptee	Inherit from the Adaptee
Implement or Inherit the Adapter type	Inherit from the Adapter type
Use composition and single inheritance	Require multiple inheritance

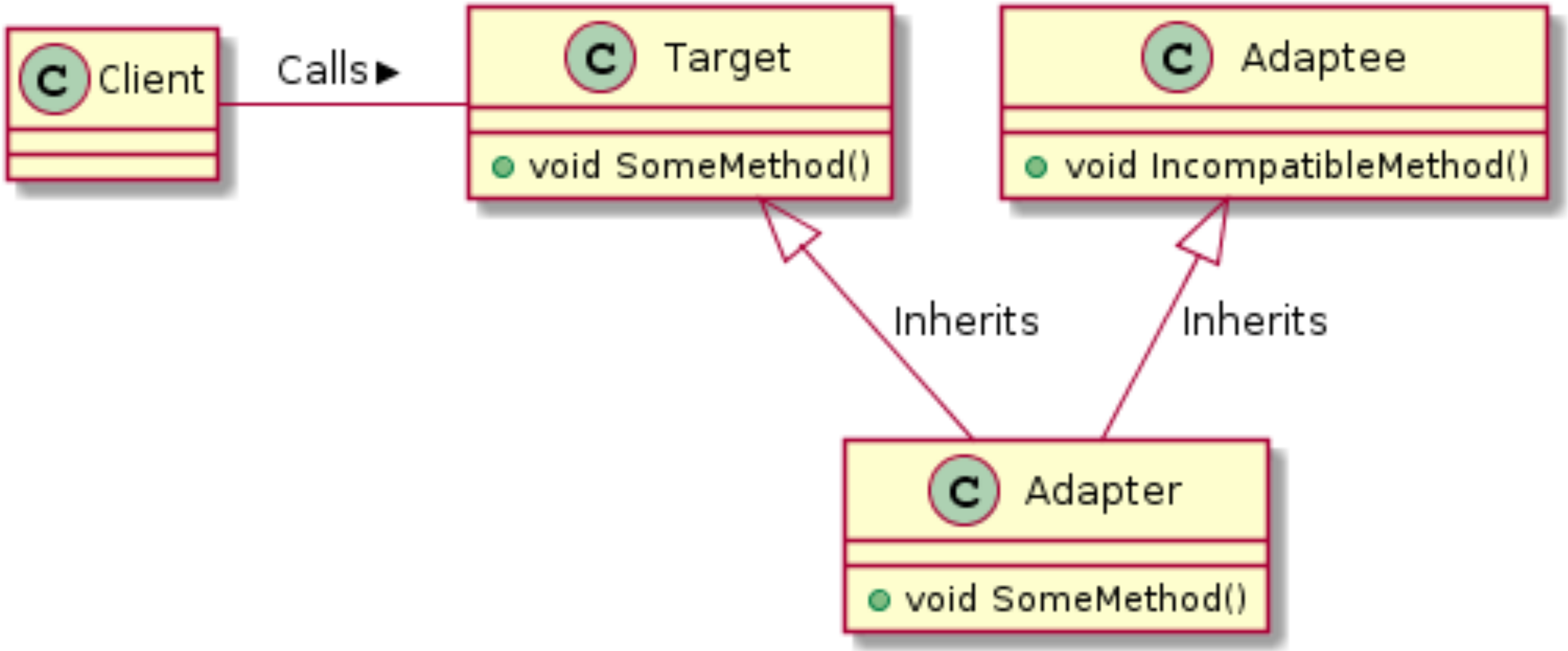
Two Kinds of Adapters

Object Adapters	Class Adapters
Hold an instance of the Adaptee	Inherit from the Adaptee
Implement or Inherit the Adapter type	Implement the Adapter interface
Use composition and single inheritance	~~Require multiple inheritance~~

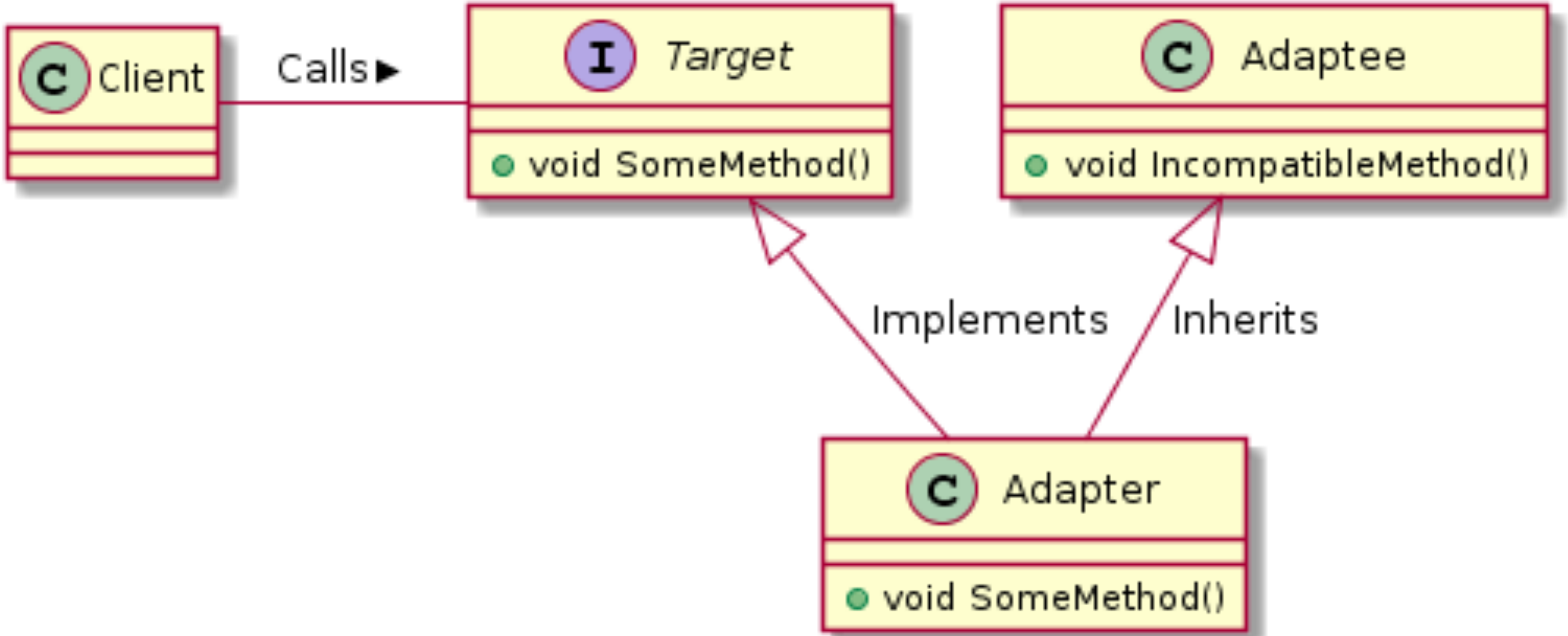
Object Adapter



Class Adapter (multiple inheritance)



Class Adapter (interface / single inheritance)



Demo

Introducing an Adapter

Related Patterns

Decorator

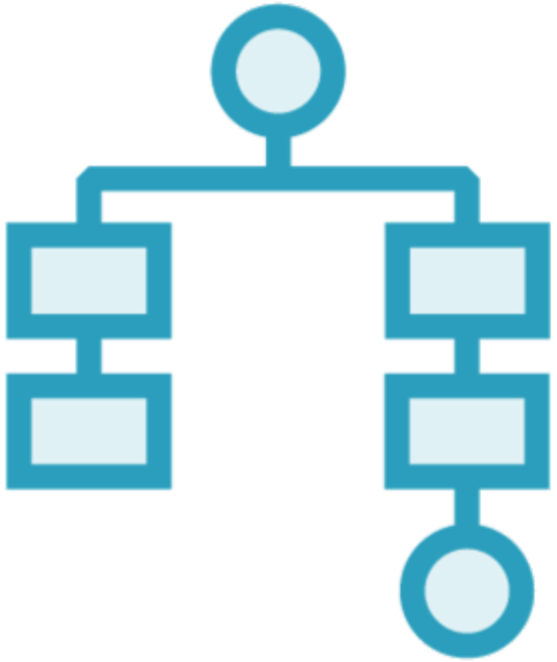
Bridge

Proxy

Repository

Strategy

Facade



Adapting Results

Create a common result type

Inherit from this type and wrap specific types

A specific result type

Incompatible with Person

```
public class Character
{
    [Newtonsoft.Json.JsonProperty("name")]
    public string FullName { get; set; }
    public string Gender { get; set; }
    [Newtonsoft.Json.JsonProperty("hair_color")]
    public string Hair { get; set; }
}
```

Base / Wrapper Type

Make abstract or virtual as required

```
public class Person
{
    public virtual string Name { get; set;  }
    public virtual string Gender { get; set; }
    [Newtonsoft.Json.JsonProperty("hair_color")]
    public virtual string HairColor { get; set; }
}
```

Implementation-Specific Wrapper

public class CharacterToPersonAdapter : Person

```
private readonly Character _character;
public CharacterToPersonAdapter(Character character)
{
    _character = character ?? throw new ArgumentNullException(nameof(character));
}

public override string Name
{
    get => _character.FullName;
    set => _character.FullName = value;
}

public override string HairColor
{
    get => _character.Hair;
    set => _character.Hair = value;
}
```

Use the Result Wrapper from Adapter Service

LINQ .Select works well for this

```
// CharacterFileSourceAdapter
public async Task<IEnumerable<Person>> GetCharacters() =>
    (await _characterFileSource
        .GetCharactersFromFile(_fileName))
        .Select(character => new CharacterToPersonAdapter(character));
```

Key Takeaways

An adapter converts an incompatible interface into a compatible one

In C#, the adapter pattern uses composition and is known as an object adapter

Adapters are similar to many other design patterns

Adapters can work with service providers but can also wrap result types

Latest sample code:
https://github.com/ardalis/DesignPatternsInCSharp

Adapter Design Pattern

APPLYING THE ADAPTER PATTERN

Steve Smith
FORCE MULTIPLIER FOR DEV TEAMS

@ardalis | ardalis.com | weeklydevtips.com