Adapter Design Pattern

Software Design Architecture Lab # 8

Muhammad Rehan Baig

Adapter Design Pattern

1. **Adapter** pattern works as a bridge between two incompatible interfaces. This type of design pattern comes under structural pattern as this pattern combines the capability of two independent interfaces.
2. This pattern involves a single class which is responsible to join functionalities of independent or incompatible interfaces.
3. Ex: A real life example could be a case of **card reader** which acts as an adapter between memory card and a laptop. You plugin the memory card into card reader and card reader into the laptop so that memory card can be read via laptop.

Why Adapter Design Pattern

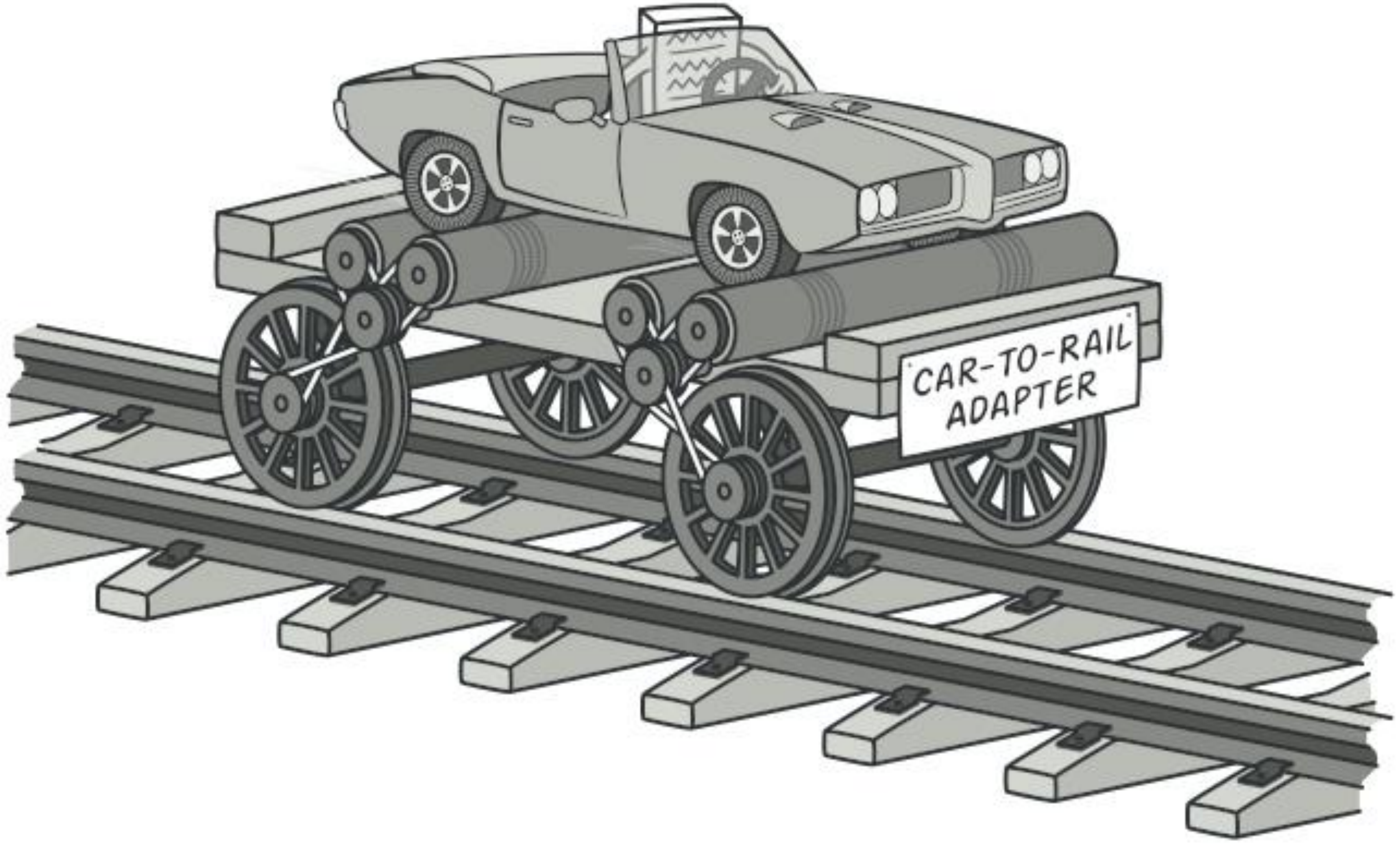
- *For making compatibility.*
- *Enhance and extend usage.*
- *Loose coupling of classes.*

When to Use this pattern?

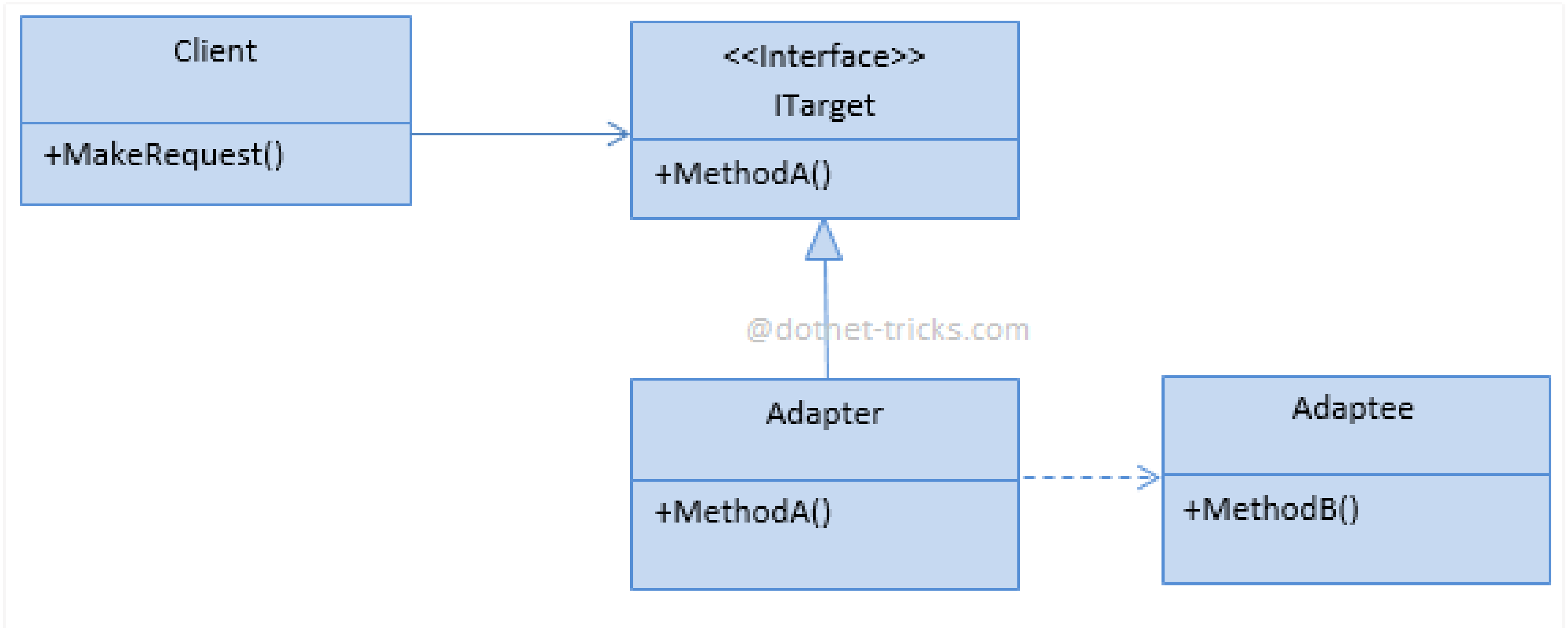
1. When we want to make compatibility between incompatible system.
2. **Adapter** pattern acts as a bridge between two incompatible interfaces. This pattern involves a single class called **adapter** which is responsible for communication between two independent or incompatible interfaces.

Example: A **card reader** acts as an **adapter** between a **memory card** and a **laptop**. You plugins the **memory card** into **card reader** and **card reader** into the **laptop** so that **memory card** can be read via **laptop**.

Example



UML Diagram



Adapter Pattern

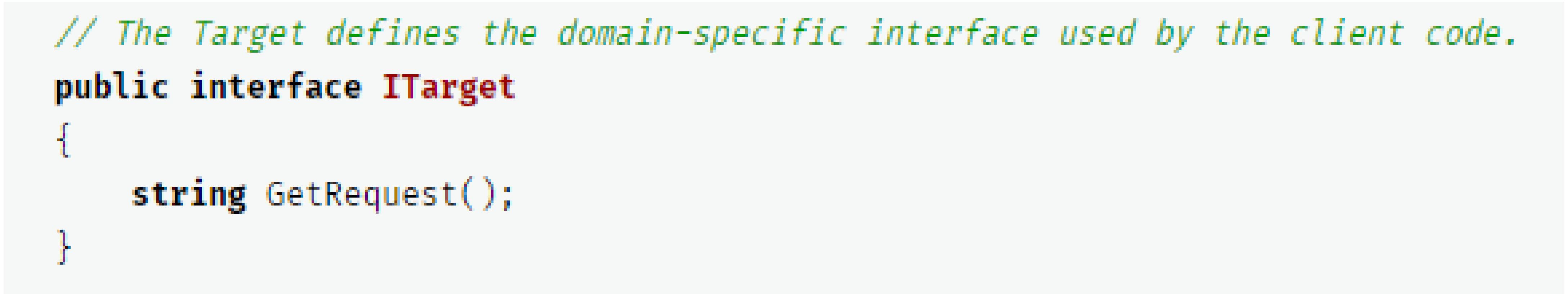
Implementation Steps

Following are the steps for implementing Adapter design pattern.

1. Client Class
2. Adapter Class
3. Adaptee Class

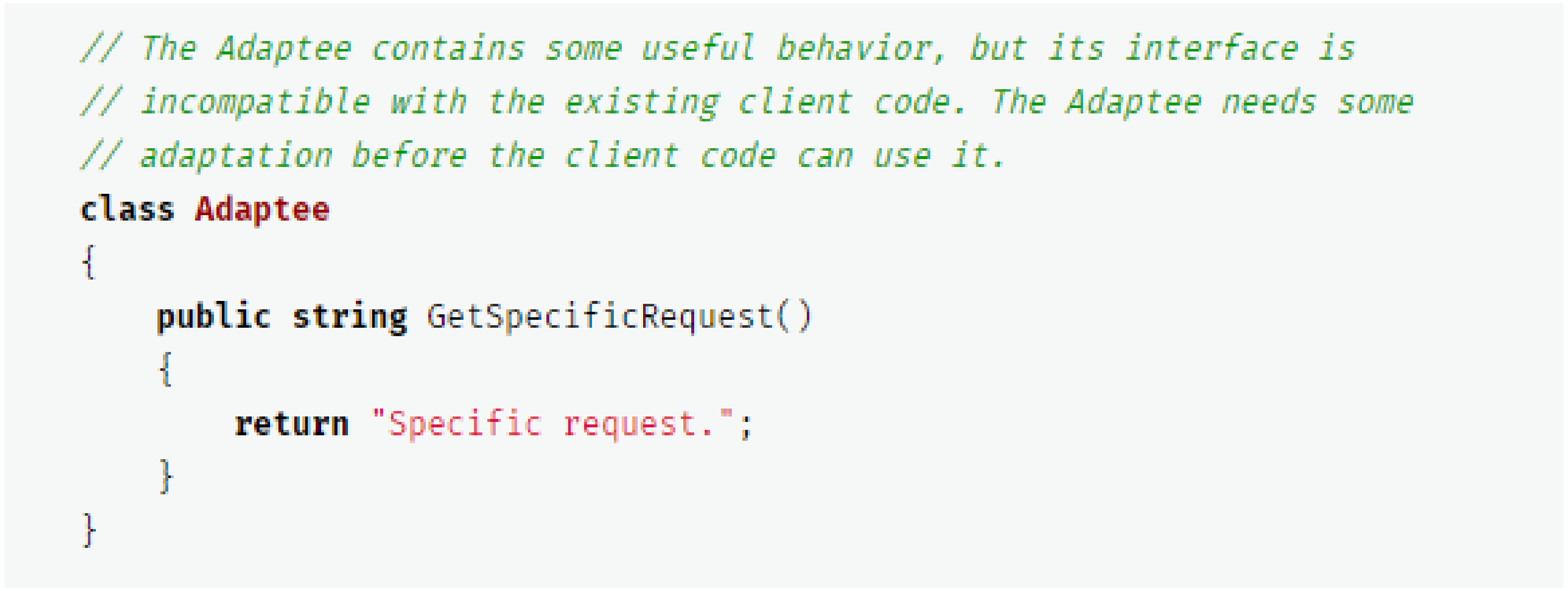
Example Step by Step

Step1



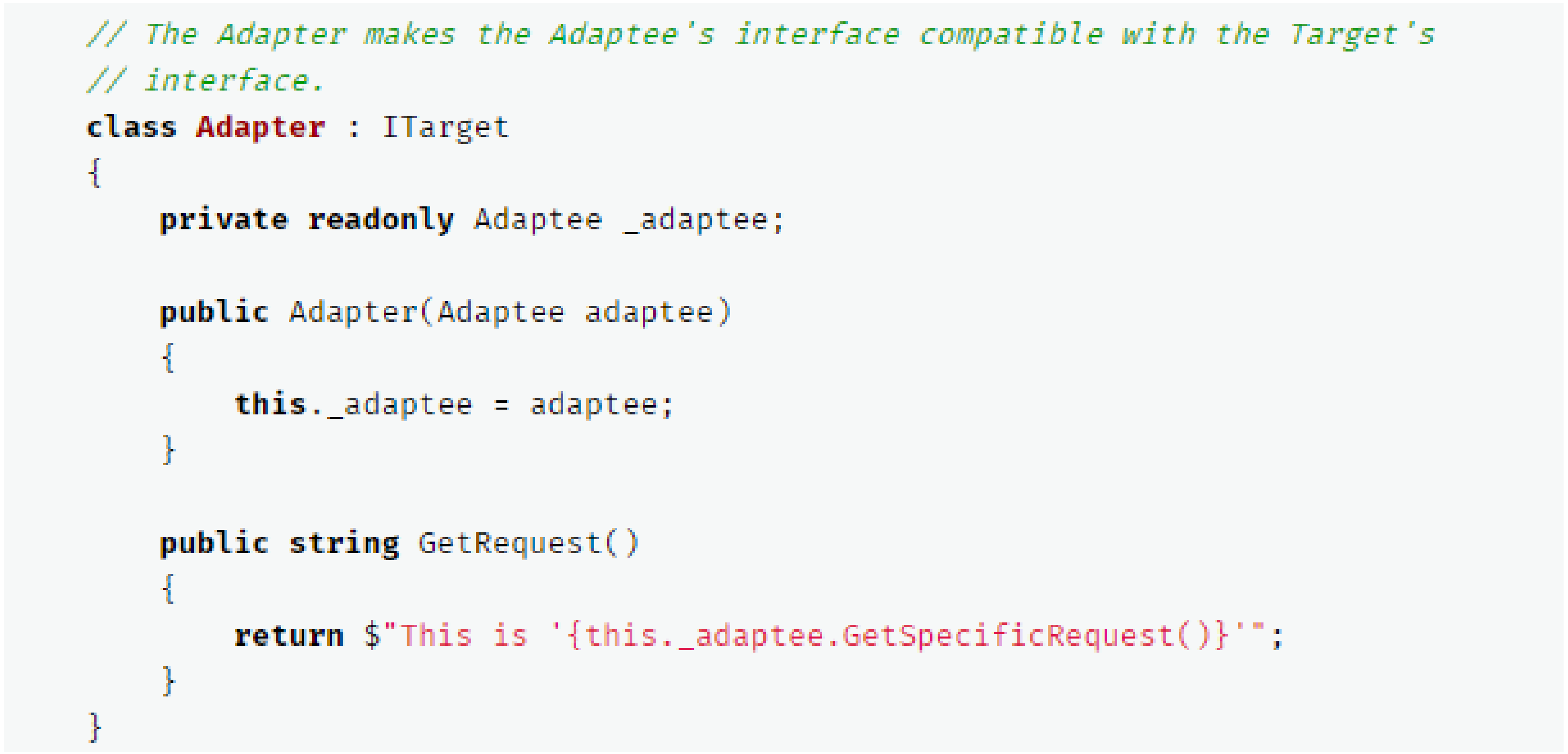
```
// The Target defines the domain-specific interface used by the client code.
public interface ITarget
{
    string GetRequest();
}
```

Step2



```
// The Adaptee contains some useful behavior, but its interface is
// incompatible with the existing client code. The Adaptee needs some
// adaptation before the client code can use it.
class Adaptee
{
    public string GetSpecificRequest()
    {
        return "Specific request.";
    }
}
```

Step 3

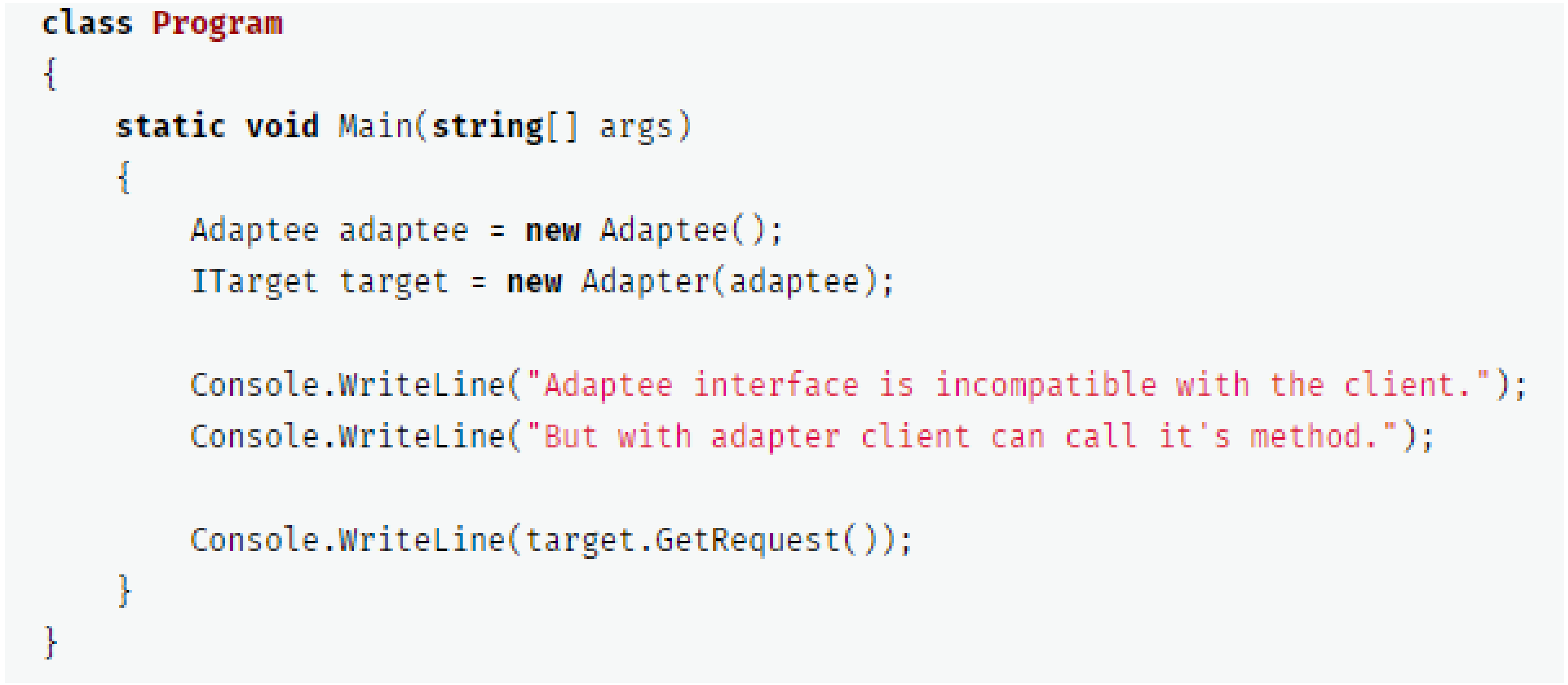


```
// The Adapter makes the Adaptee's interface compatible with the Target's
// interface.
class Adapter : ITarget
{
    private readonly Adaptee _adaptee;

    public Adapter(Adaptee adaptee)
    {
        this._adaptee = adaptee;
    }

    public string GetRequest()
    {
        return $"This is '{this._adaptee.GetSpecificRequest()}'";
    }
}
```

Consumation (Client Code)

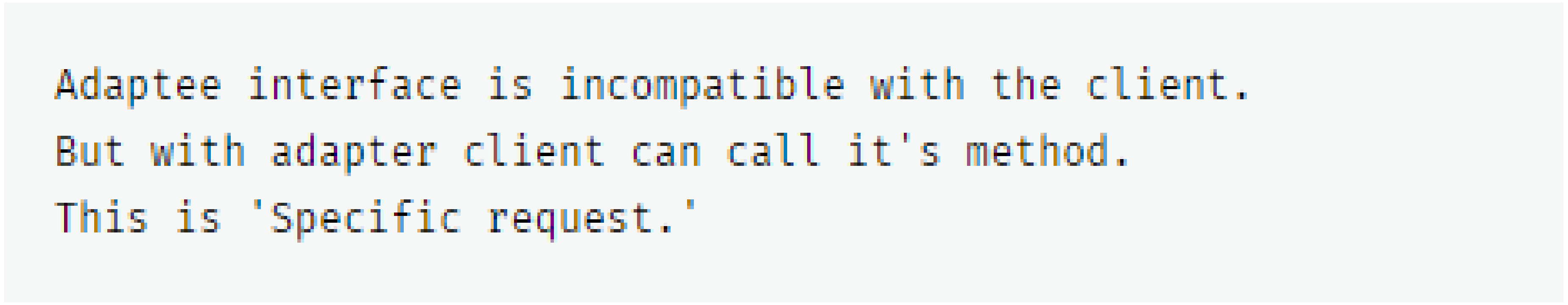


```
class Program
{
    static void Main(string[] args)
    {
        Adaptee adaptee = new Adaptee();
        ITarget target = new Adapter(adaptee);

        Console.WriteLine("Adaptee interface is incompatible with the client.");
        Console.WriteLine("But with adapter client can call it's method.");

        Console.WriteLine(target.GetRequest());
    }
}
```

Output



```
Adaptee interface is incompatible with the client.
But with adapter client can call it's method.
This is 'Specific request.'
```

Tasks

1. You are working on a project where you get data from third party webapi/service in XML format. You are creating charts from that data for creating a chart you need data in JSON format in order to make them compatiable you need to convert xml data into json but limitation is you can’t modify third party webservice/api you have to make Adapter that returns data in JSON by using adapter pattern.