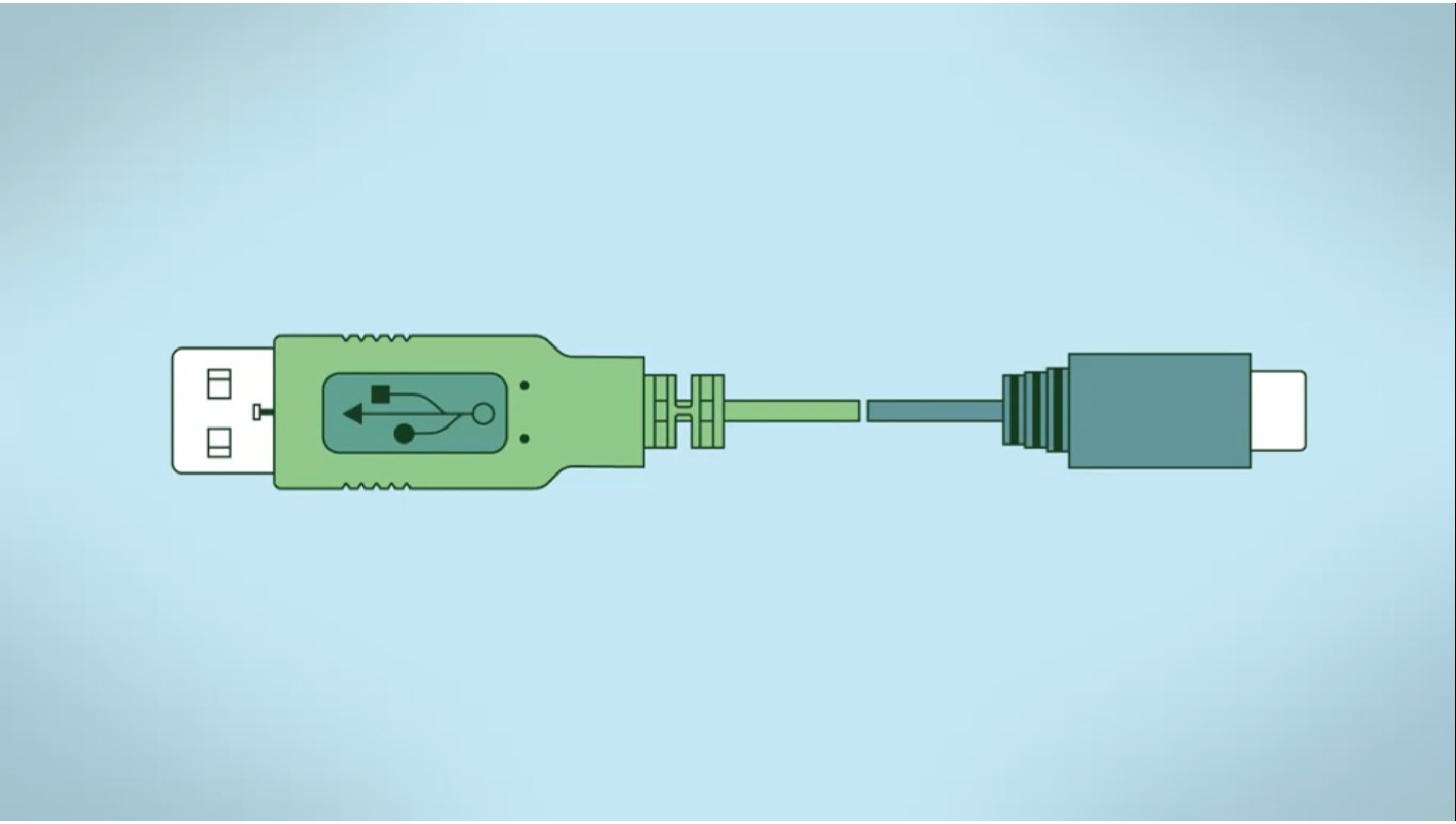
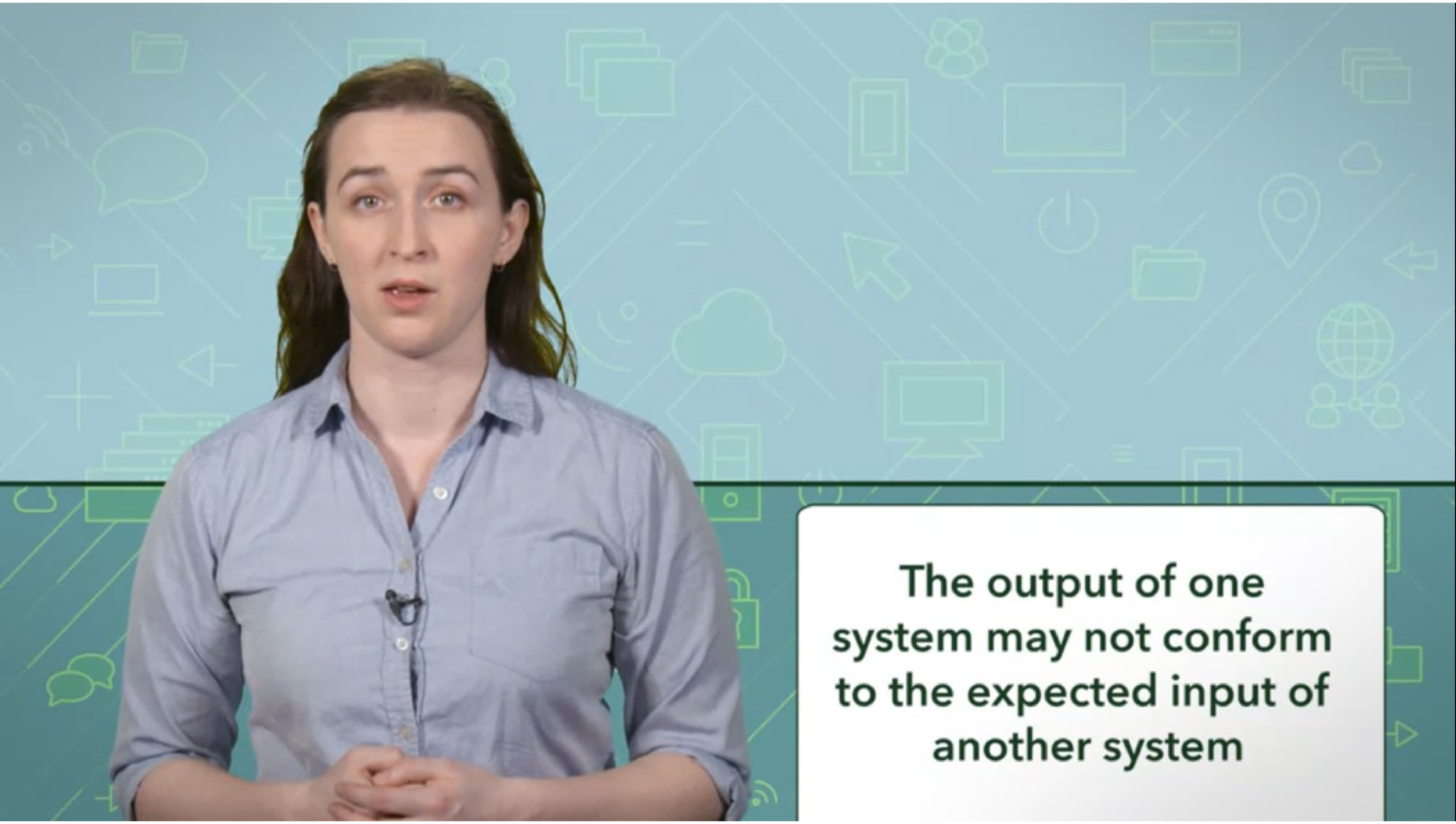
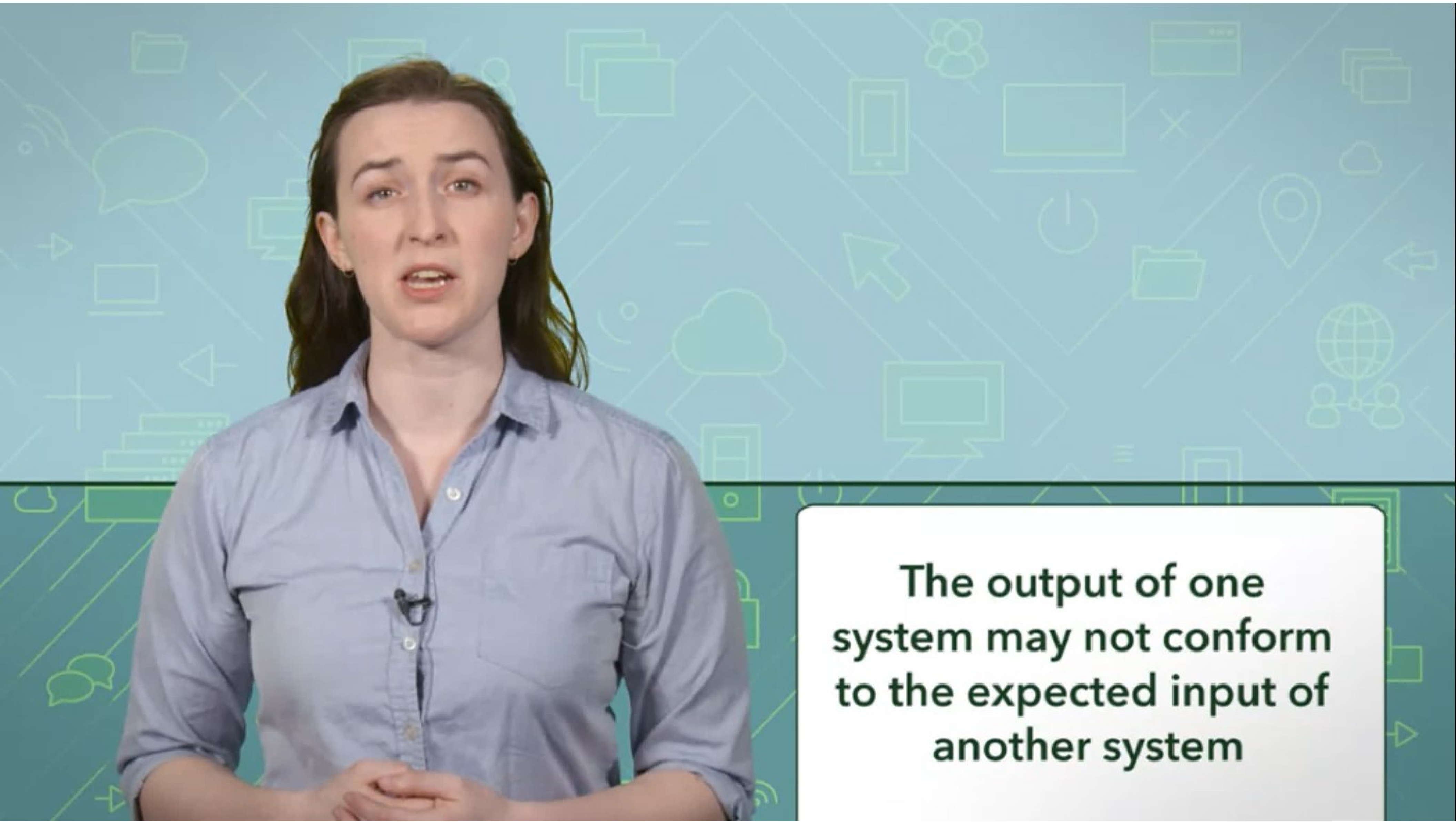
Adapter Pattern

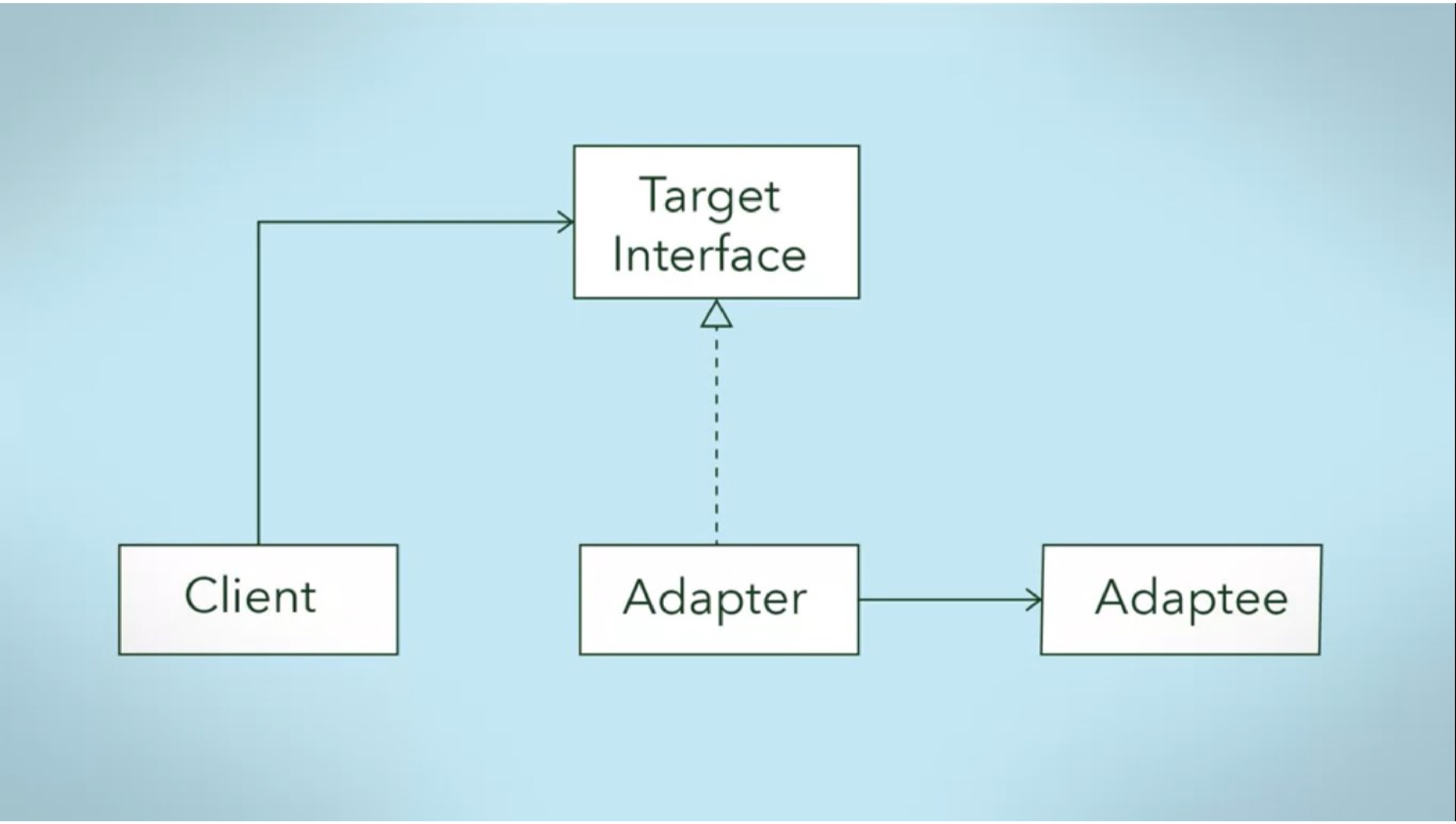


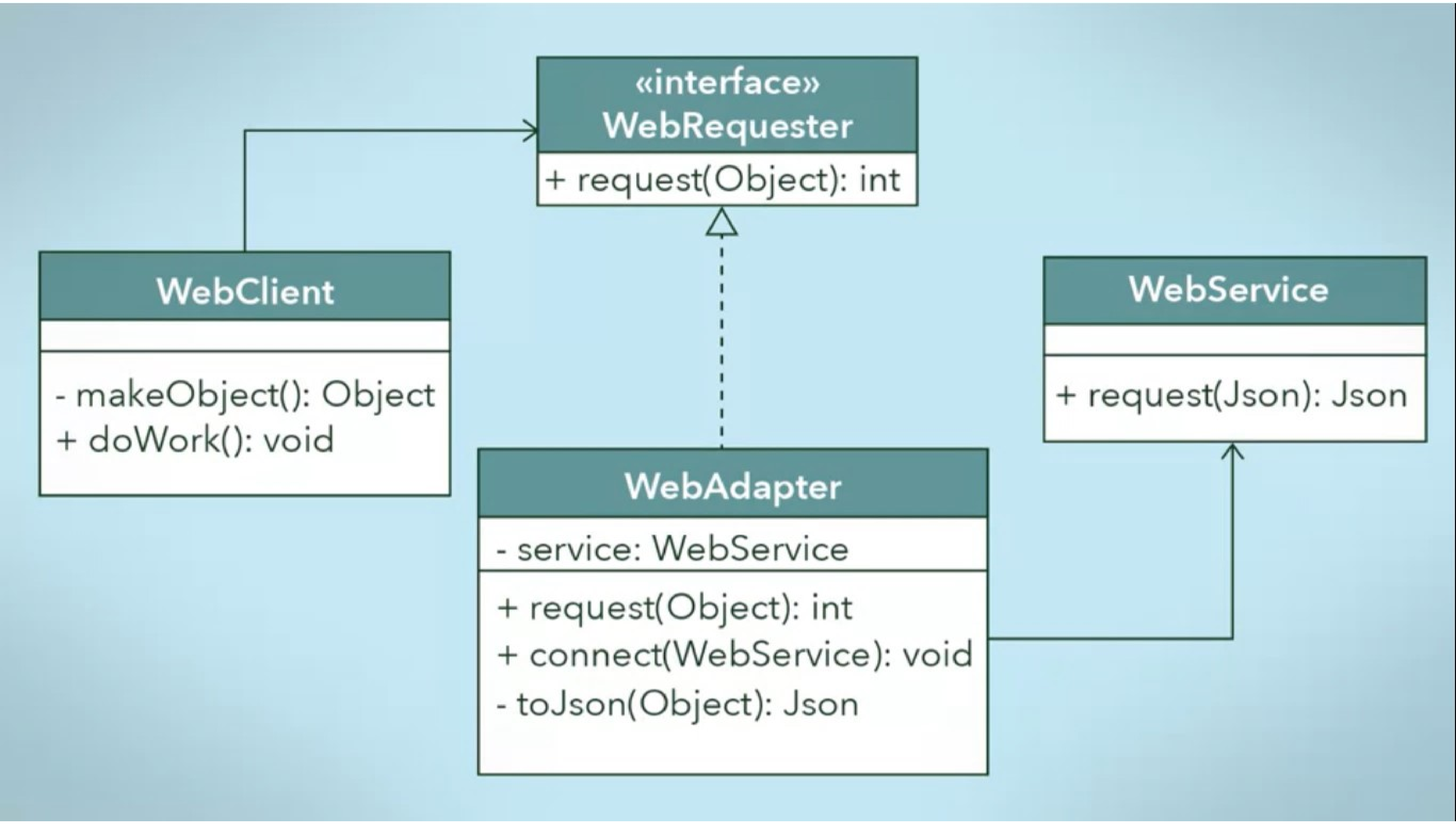


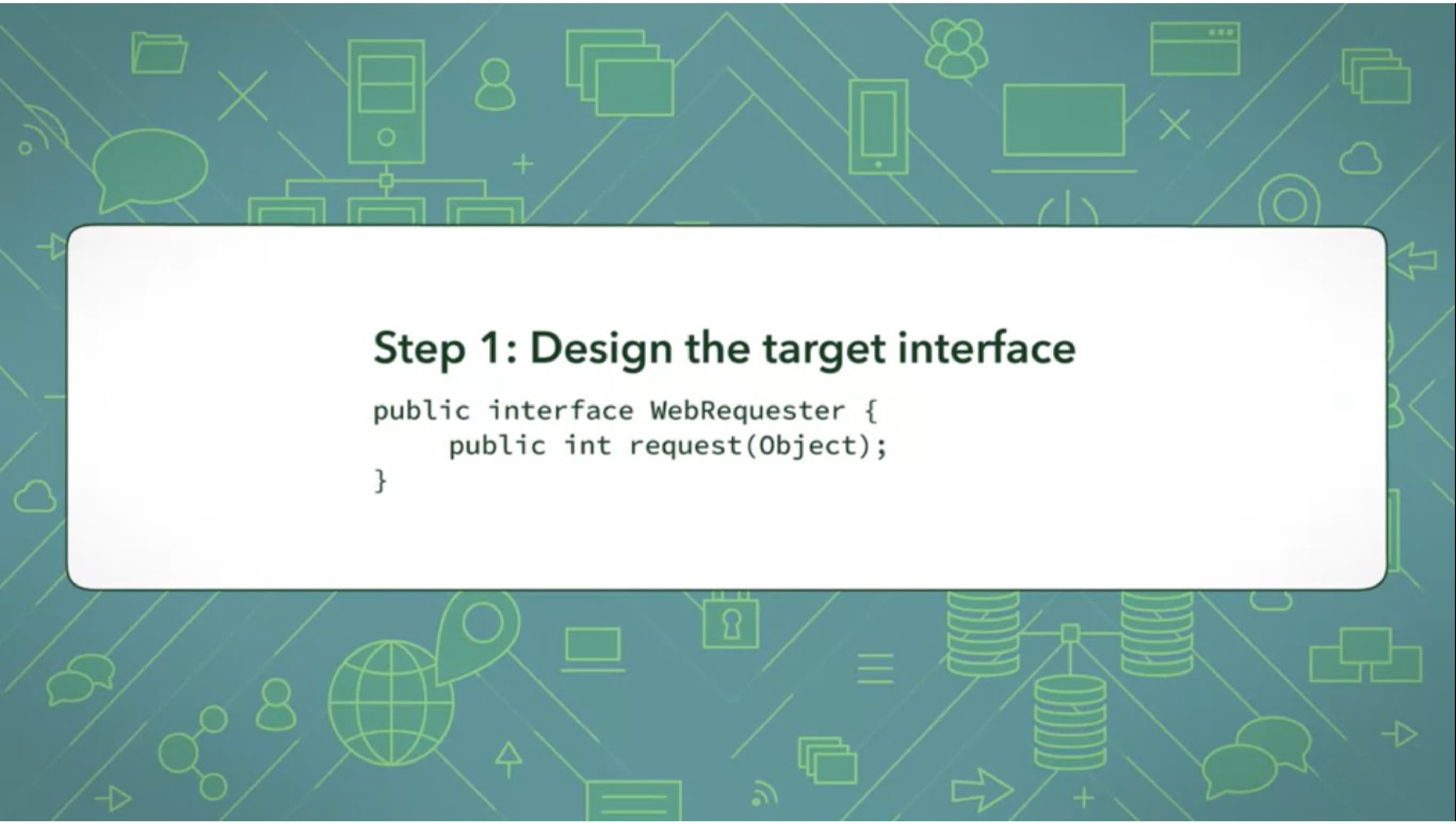
The output of one system may not conform to the expected input of another system



The output of one system may not conform to the expected input of another system


Target Interface
Client
Adapter
Adaptee


«interface»
WebRequester
+ request(Object): int
WebClient
- makeObject(): Object
+ doWork(): void
WebService
+ request(Json): Json
WebAdapter
- service: WebService
+ request(Object): int
+ connect(WebService): void
- toJson(Object): Json

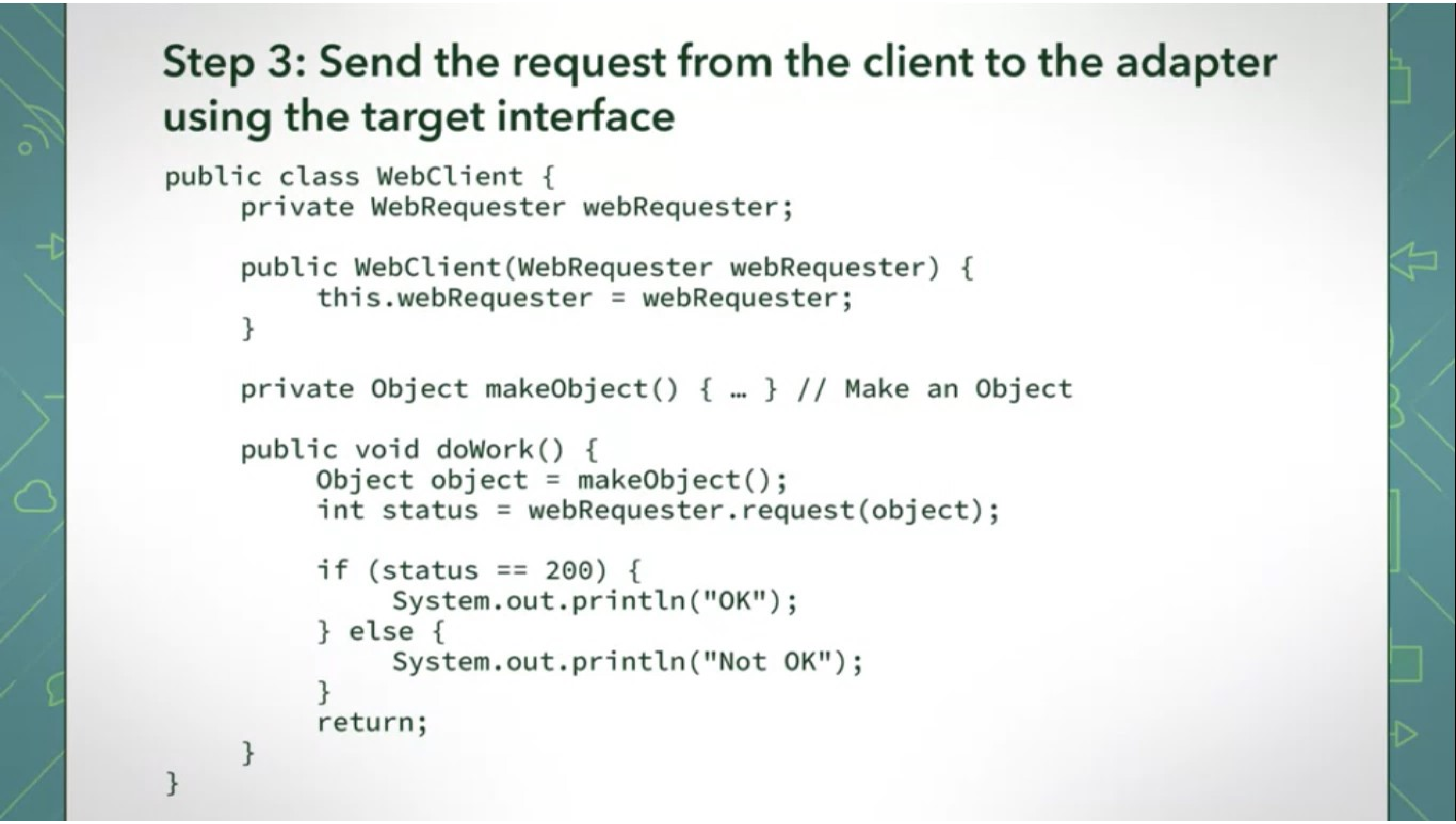


Step 1: Design the target interface

```
public interface WebRequester {
    public int request(Object);
}
```

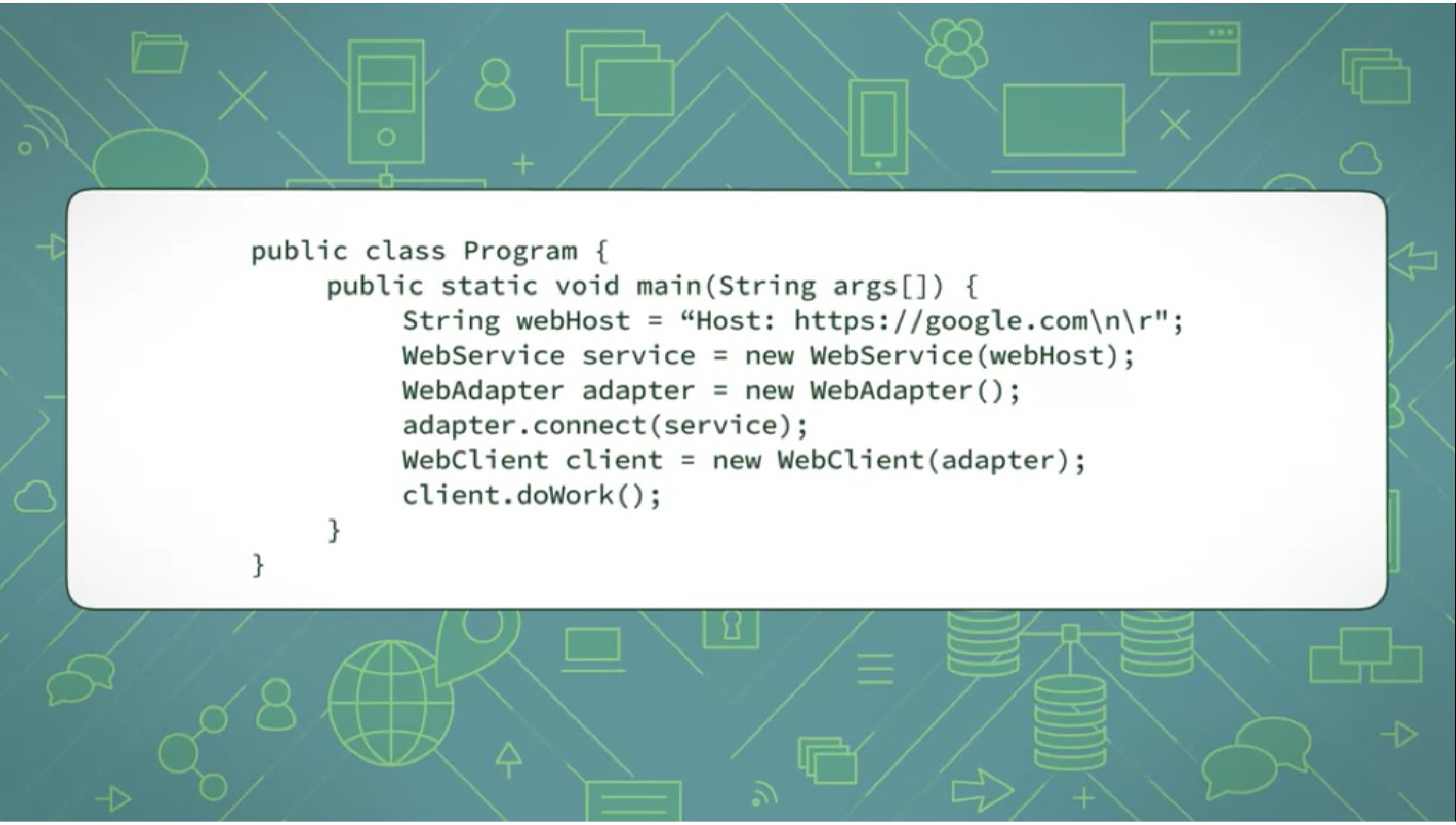
Step 2: Implement the target interface with the adapter class

```java
public class WebAdapter implements WebRequester {
    private WebService service;

    public void connect(WebService currentService) {
        this.service = currentService;
    }

    public int request(Object request) {
        Json result = this.toJson(request);
        Json response = service.request(result);
        if (response != null)
            return 200; // OK status code
        return 500; // Server error status code
    }

    private Json toJson(Object input) { … }
}
```

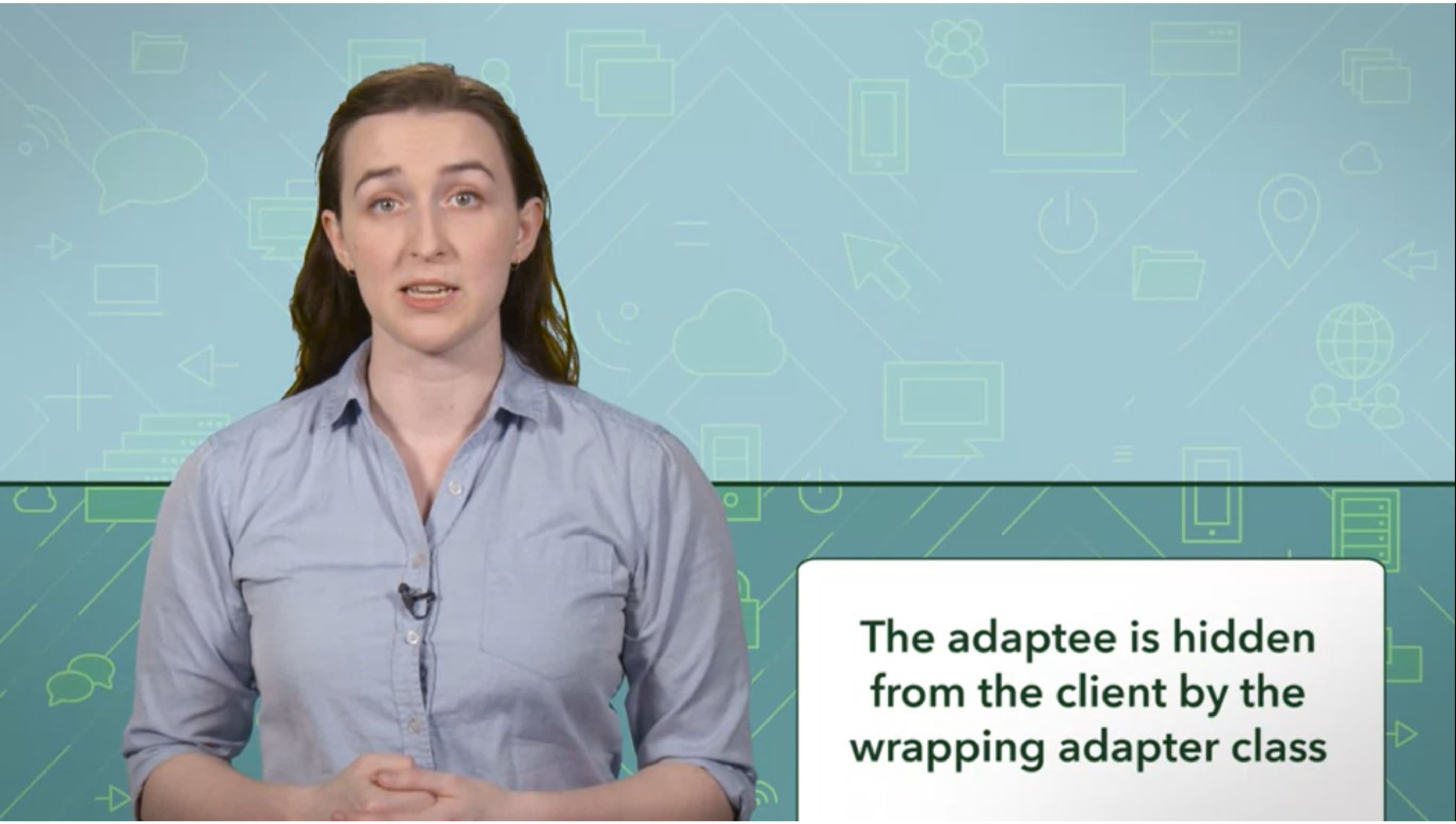


Step 3: Send the request from the client to the adapter using the target interface

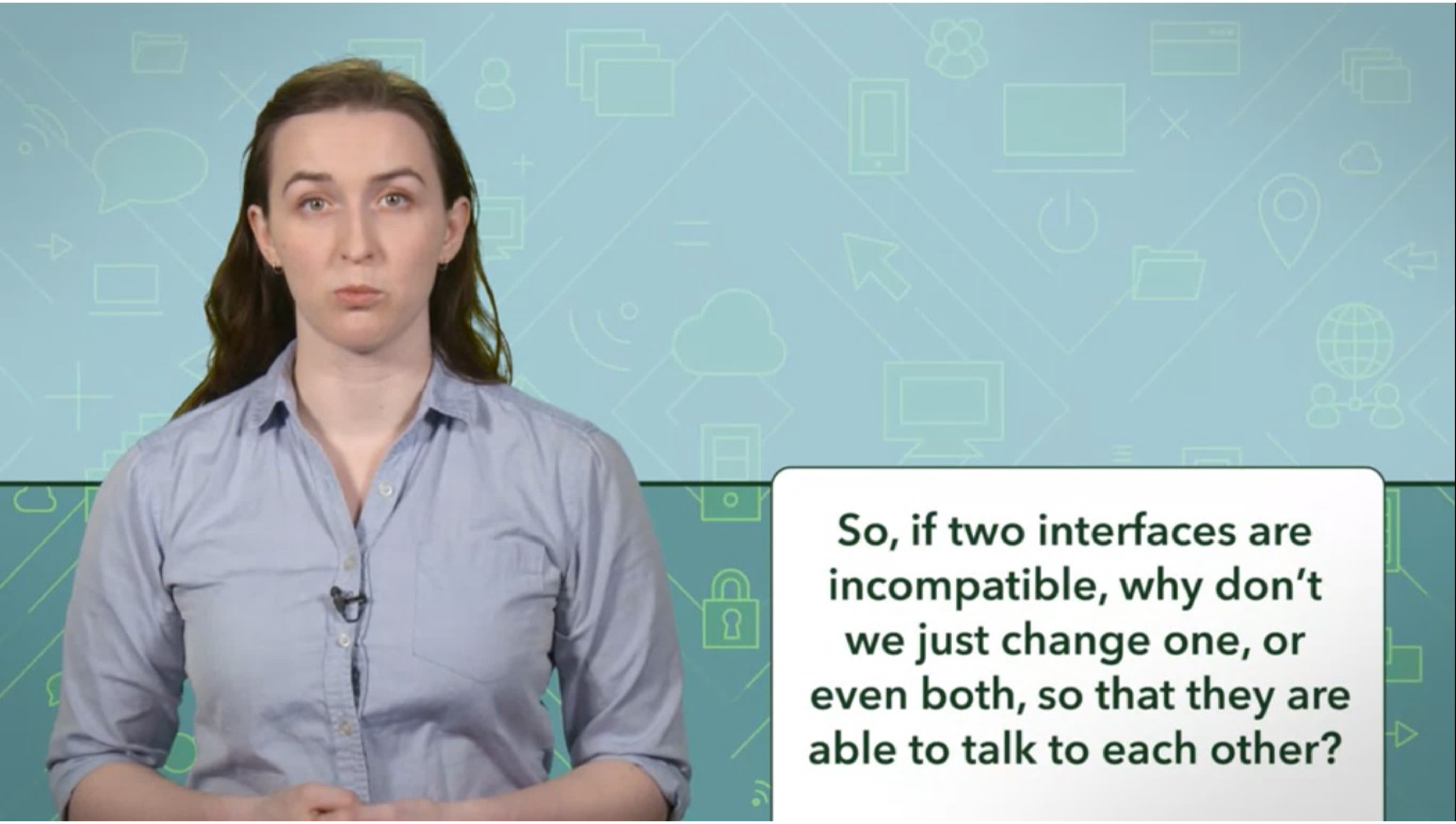
```java
public class WebClient {
     private WebRequester webRequester;

     public WebClient(WebRequester webRequester) {
          this.webRequester = webRequester;
     }

     private Object makeObject() { … } // Make an Object

     public void doWork() {
          Object object = makeObject();
          int status = webRequester.request(object);

          if (status == 200) {
               System.out.println("OK");
          } else {
               System.out.println("Not OK");
          }
          return;
     }
}
```



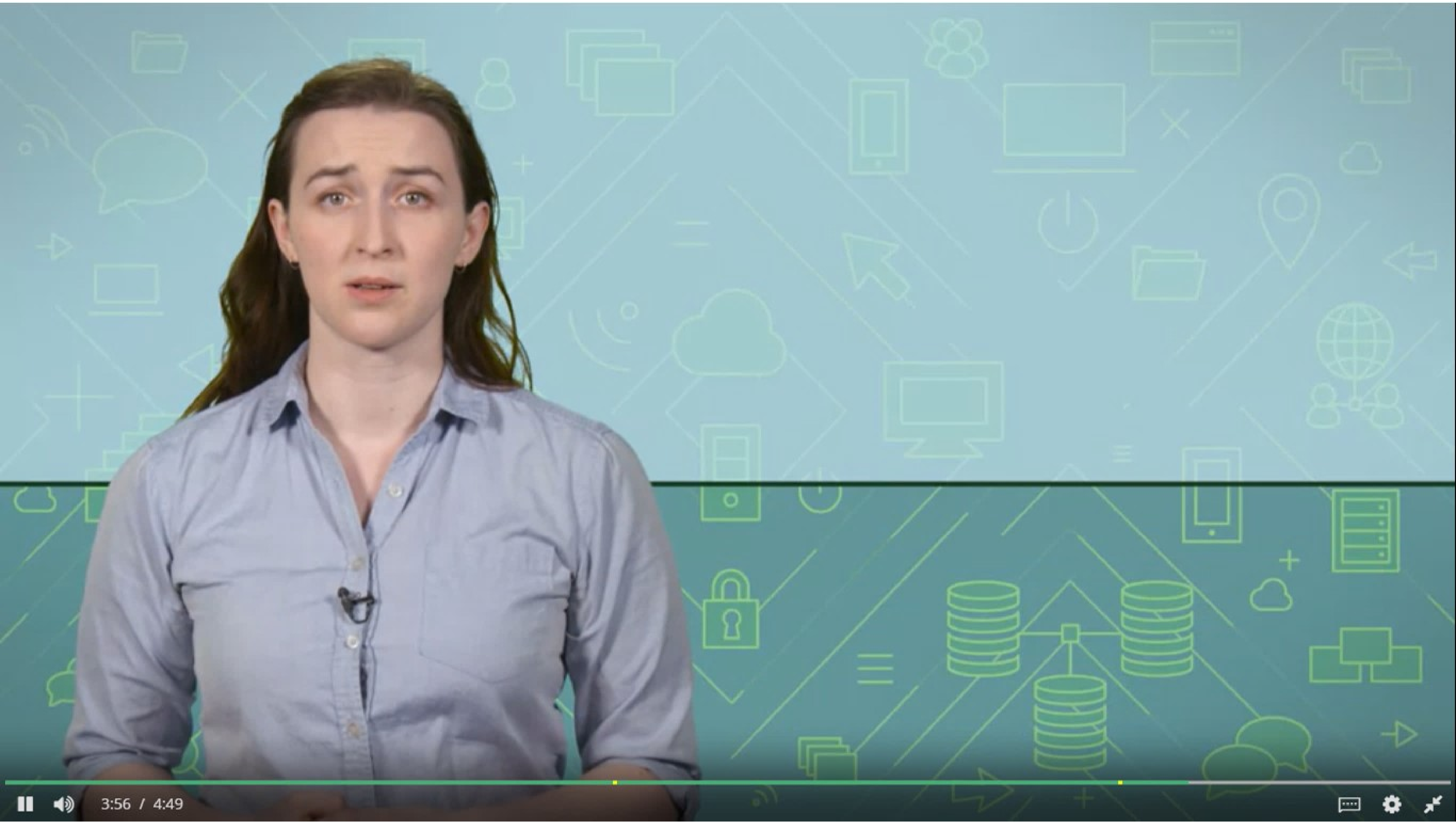
```
public class Program {
    public static void main(String args[]) {
        String webHost = “Host: https://google.com\n\r";
        WebService service = new WebService(webHost);
        WebAdapter adapter = new WebAdapter();
        adapter.connect(service);
        WebClient client = new WebClient(adapter);
        client.doWork();
    }
}
```

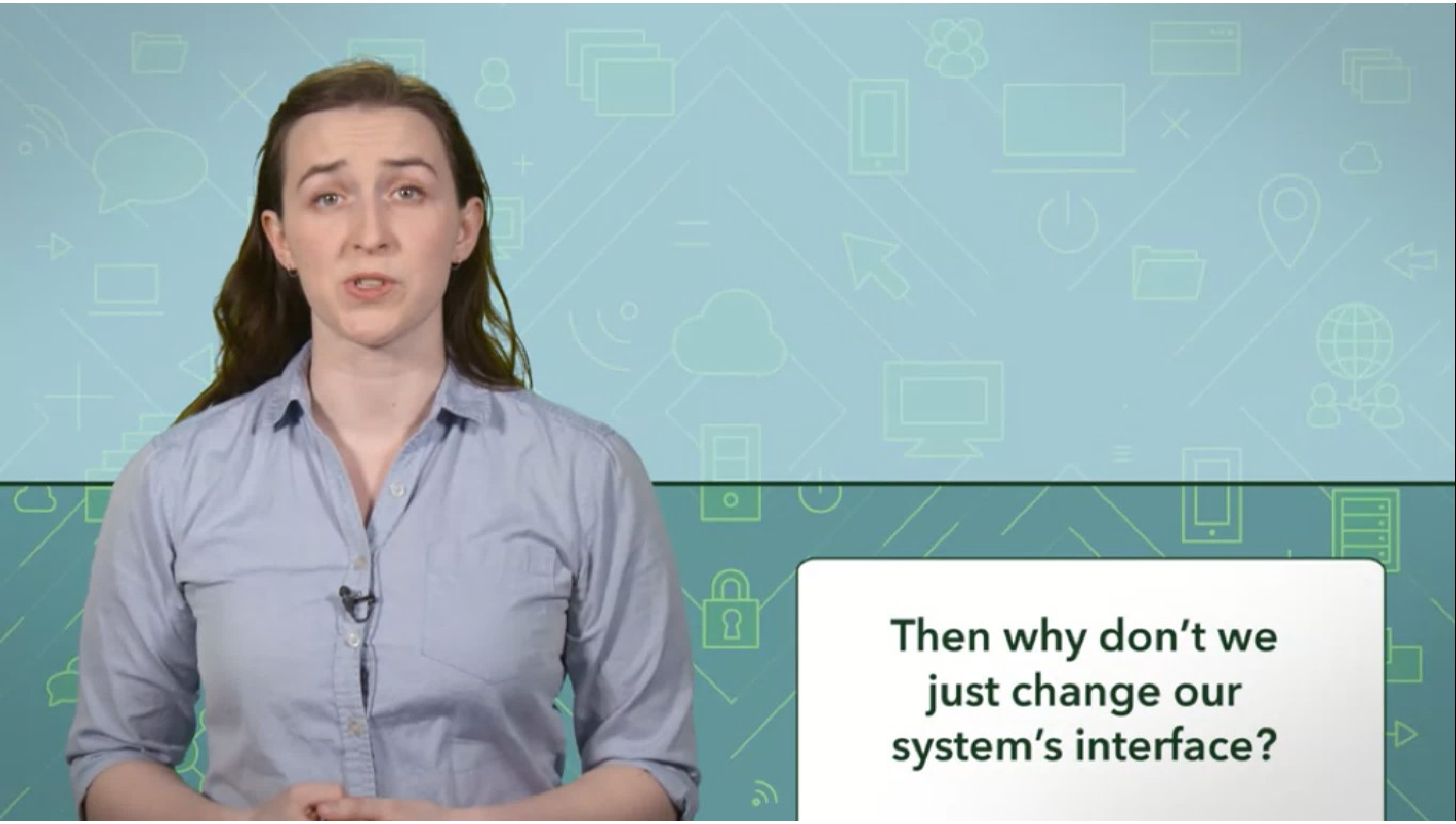


The adaptee is hidden from the client by the wrapping adapter class

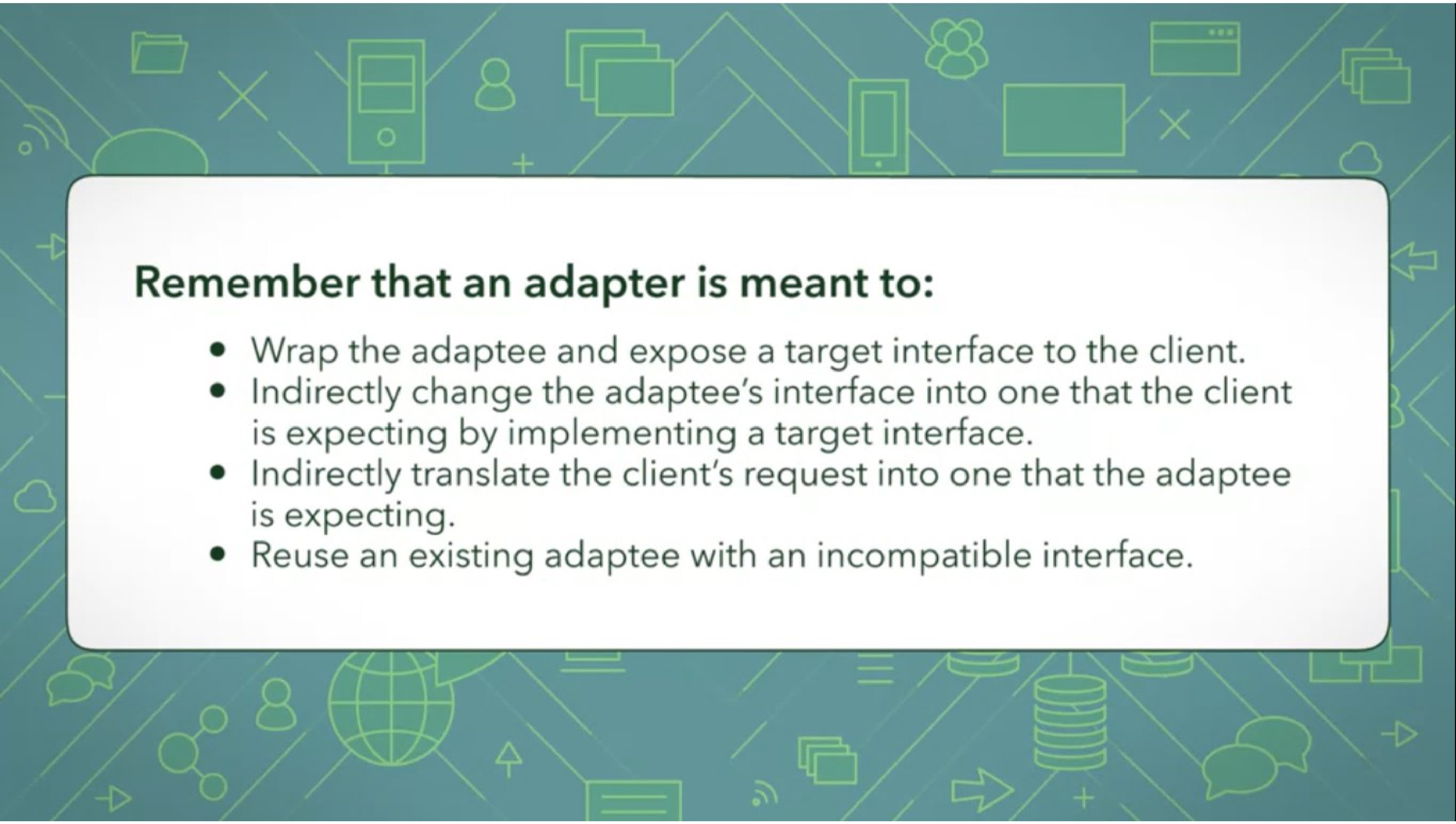


So, if two interfaces are incompatible, why don't we just change one, or even both, so that they are able to talk to each other?


3:56 / 4:49

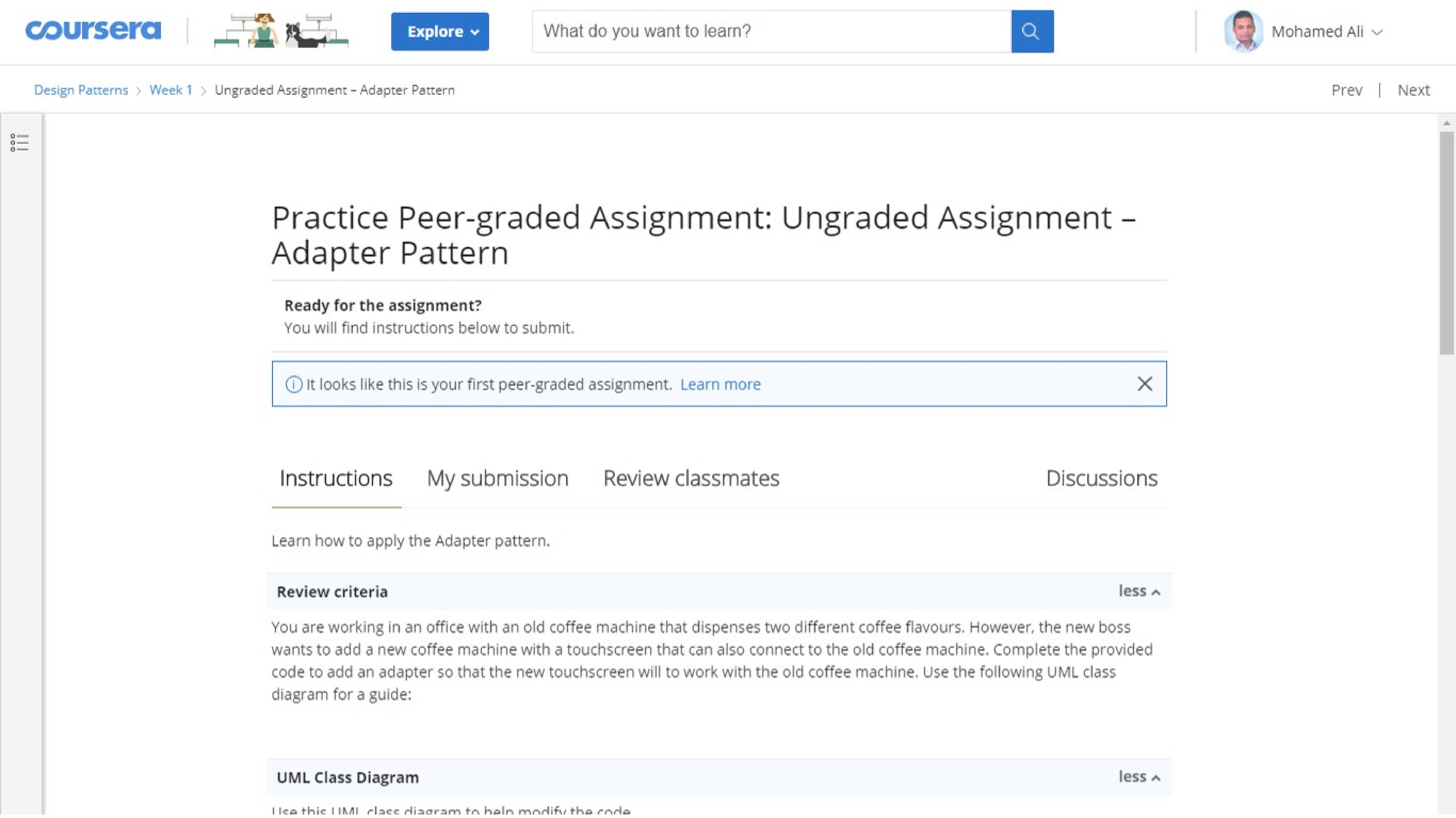


Then why don't we just change our system's interface?



Remember that an adapter is meant to:

- Wrap the adaptee and expose a target interface to the client.
- Indirectly change the adaptee's interface into one that the client is expecting by implementing a target interface.
- Indirectly translate the client's request into one that the adaptee is expecting.
- Reuse an existing adaptee with an incompatible interface.


coursera
Explore
What do you want to learn?
Mohamed Ali
Design Patterns > Week 1 > Ungraded Assignment – Adapter Pattern
Prev | Next

Practice Peer-graded Assignment: Ungraded Assignment – Adapter Pattern

Ready for the assignment?
You will find instructions below to submit.

It looks like this is your first peer-graded assignment. Learn more

Instructions
My submission
Review classmates
Discussions

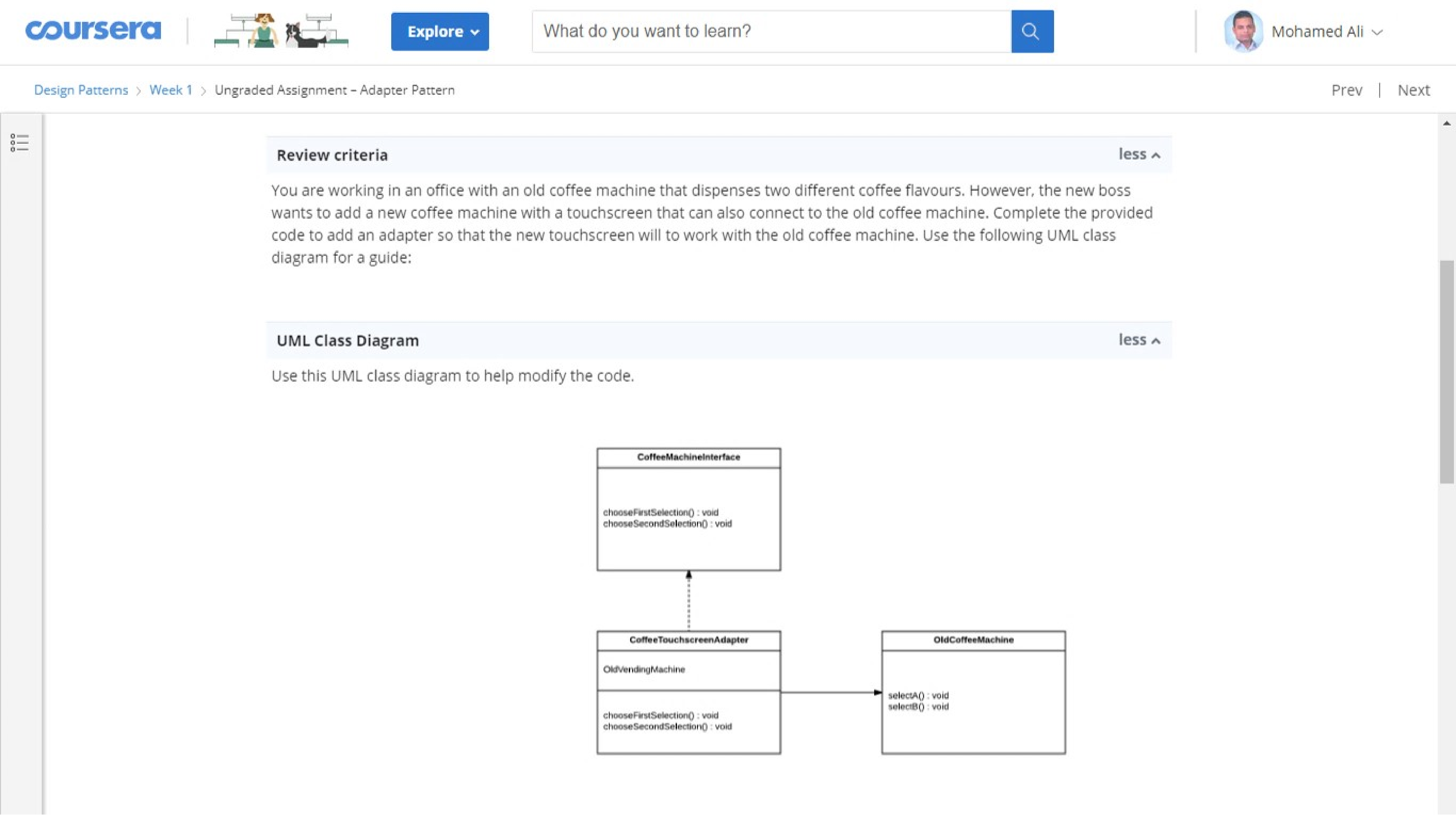
Learn how to apply the Adapter pattern.

Review criteria less

You are working in an office with an old coffee machine that dispenses two different coffee flavours. However, the new boss wants to add a new coffee machine with a touchscreen that can also connect to the old coffee machine. Complete the provided code to add an adapter so that the new touchscreen will to work with the old coffee machine. Use the following UML class diagram for a guide:

UML Class Diagram less

Use this UML class diagram to help modify the code

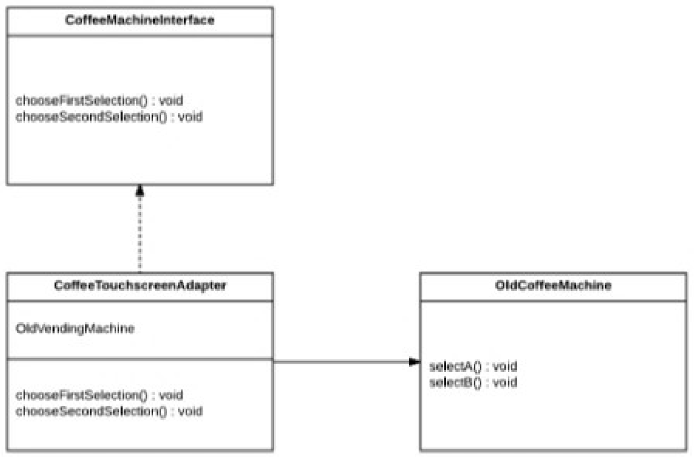


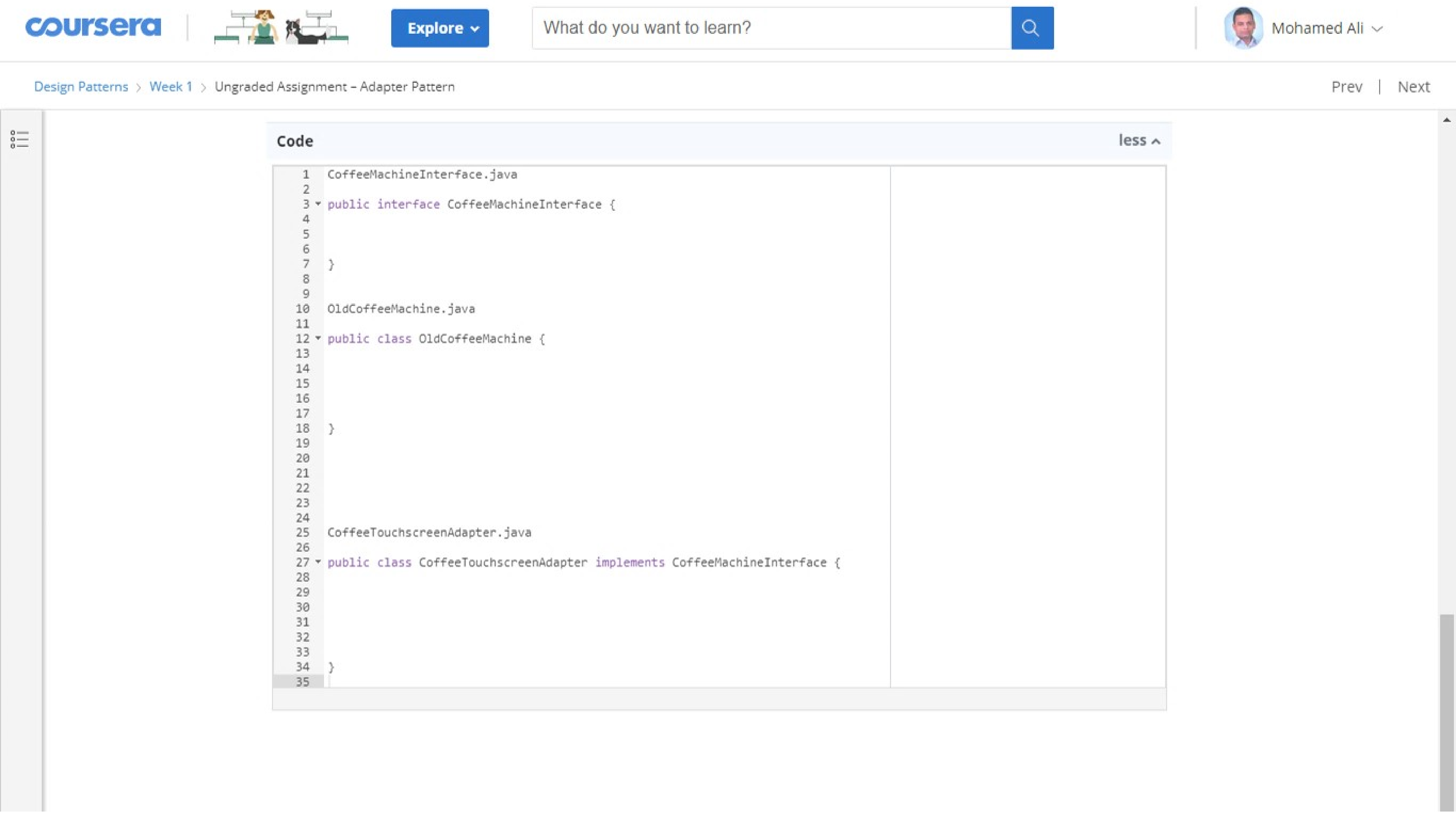
Review criteria

You are working in an office with an old coffee machine that dispenses two different coffee flavours. However, the new boss wants to add a new coffee machine with a touchscreen that can also connect to the old coffee machine. Complete the provided code to add an adapter so that the new touchscreen will to work with the old coffee machine. Use the following UML class diagram for a guide:

UML Class Diagram

Use this UML class diagram to help modify the code.

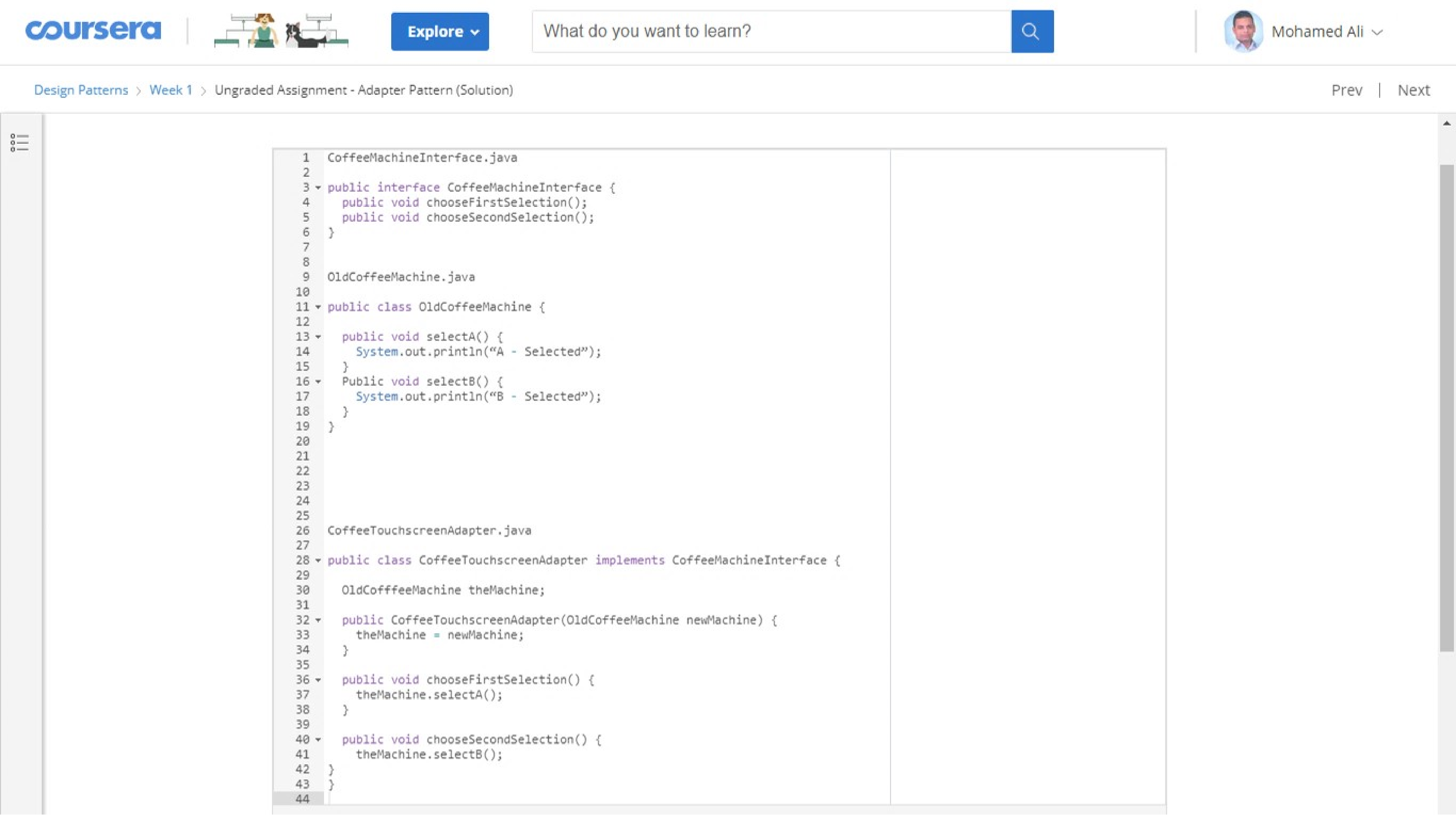

CoffeeMachineInterface
chooseFirstSelection() : void
chooseSecondSelection() : void
CoffeeTouchscreenAdapter
OldVendingMachine
chooseFirstSelection() : void
chooseSecondSelection() : void
OldCoffeeMachine
selectA() : void
selectB() : void



Design Patterns > Week 1 > Ungraded Assignment – Adapter Pattern

Code

```
CoffeeMachineInterface.java

public interface CoffeeMachineInterface {



}


OldCoffeeMachine.java

public class OldCoffeeMachine {





}






CoffeeTouchscreenAdapter.java

public class CoffeeTouchscreenAdapter implements CoffeeMachineInterface {






}

```


coursera
Explore
What do you want to learn?
Mohamed Ali
Design Patterns > Week 1 > Ungraded Assignment - Adapter Pattern (Solution)
Prev | Next

```
CoffeeMachineInterface.java

public interface CoffeeMachineInterface {
  public void chooseFirstSelection();
  public void chooseSecondSelection();
}


OldCoffeeMachine.java

public class OldCoffeeMachine {

  public void selectA() {
    System.out.println("A - Selected");
  }
  Public void selectB() {
    System.out.println("B - Selected");
  }
}






CoffeeTouchscreenAdapter.java

public class CoffeeTouchscreenAdapter implements CoffeeMachineInterface {

  OldCofffeeMachine theMachine;

  public CoffeeTouchscreenAdapter(OldCoffeeMachine newMachine) {
    theMachine = newMachine;
  }

  public void chooseFirstSelection() {
    theMachine.selectA();
  }

  public void chooseSecondSelection() {
    theMachine.selectB();
}
}
```