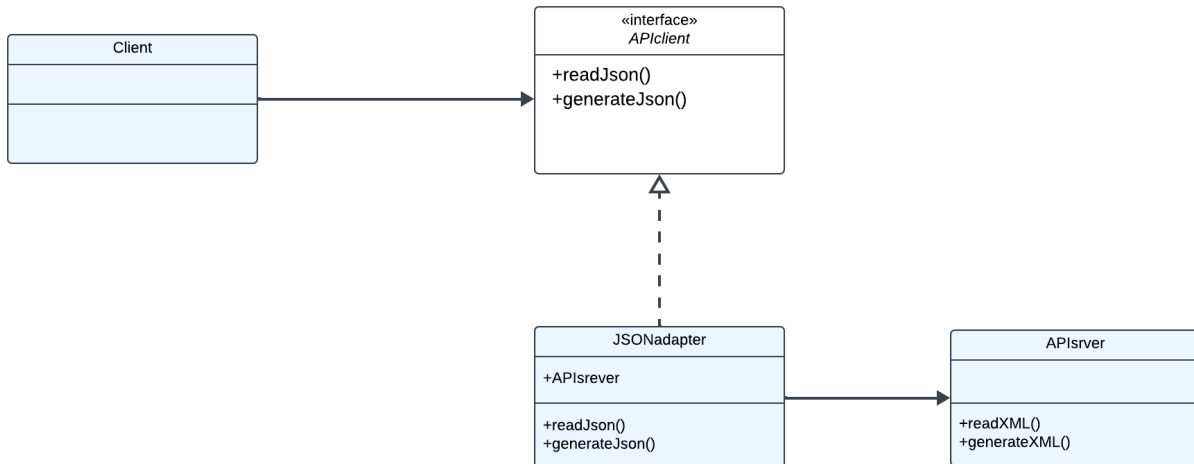


Design Patterns-3

Heba Saeed Hamdan202010678

-The 1'st direction "API client to API server "using JSONadapter because the client read and generate content in JSON format:



- APIclient "target":

```
public interface APIclient {  
    public void readJson();  
    public String generateJson();  
}
```

- APIserver "Adaptee":

```
public class APIserver {  
    public void readXML(){  
        System.out.println("data from server with XML format");  
    }  
    public String generateXML(){  
        return " Generating XML data from server";  
    }  
}
```

- **JSONadapter “Adapter”:**

```
public class JSONadapter implements APIclient {  
    APIserver xmlformat;
```

```
    public JSONadapter(APIserver xmlformat) {  
        this.xmlformat=xmlformat;  
    }  
}
```

```
@Override
```

```
public void readJson() {  
    xmlformat.readXML();  
}
```

```
@Override
```

```
public String generateJson() {  
    return xmlformat.generateXML();  
}  
}
```

- **Client class “test ”://Q3**

```
public class Main {  
    public static void main(String[] args) {  
        APIserver xmlformat=new APIserver();  
        APIclient content=new JSONadapter(xmlformat);  
        System.out.println(content.generateJson());  
    }  
}
```

```

12 public class Main {
13
14     public static void main(String[] args) {
15         APIServer xmlformat=new APIServer();
16         APIClient content=new JSONAdapter(xmlformat);
17         System.out.println(content.generateJson());
18     }
19
20 }
21

```

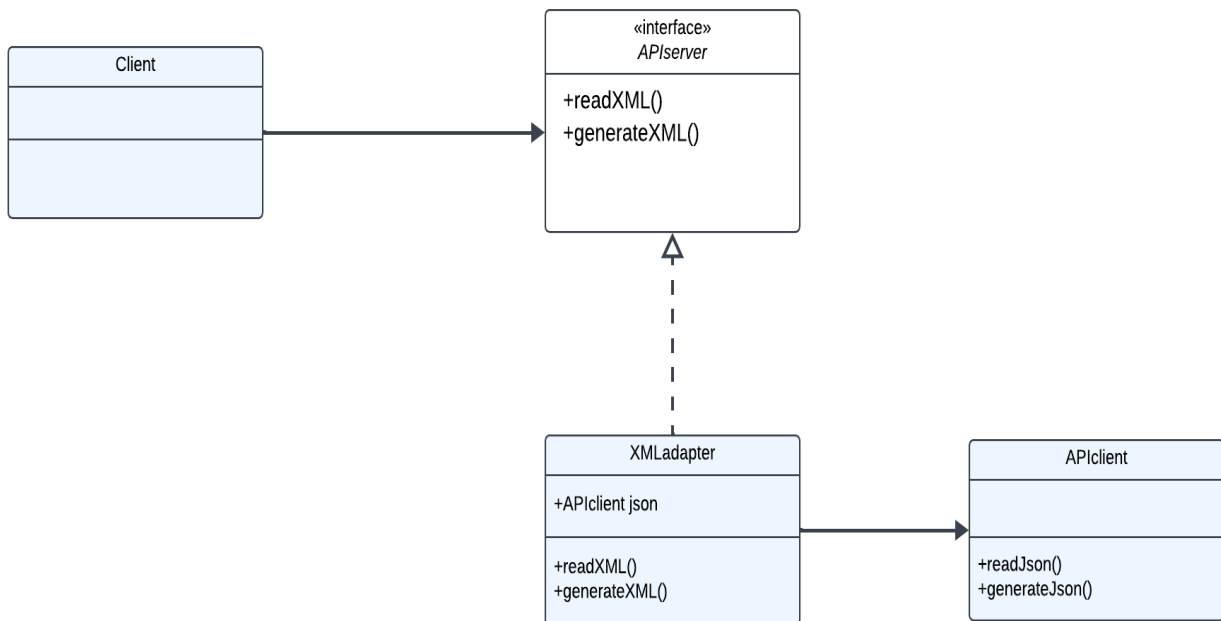
Output - assign3 (run) x

```

run:
Generating XML data from server
BUILD SUCCESSFUL (total time: 0 seconds)

```

-The second direction “API server to API client ”using XMLAdapter because the server read and generate content in XML format:



- **APIServer “target”:**

```

public interface APIServer {
    public void readXML();
    public String generateXML();
}

```

```
}
```

- **APIclient “Adaptee”:**

```
public class APIclient {  
    public void readJson(){  
        System.out.println("Data from client with JSON format");  
    }  
    public String generateJson(){  
        return "Generating JSON data from client";  
    }  
}
```

- **XMLadapter “Adapter”:**

```
public class XMLadapter implements APIserver{  
    APIclient jsonformat;  
  
    public XMLadapter(APIclient jsonformat) {  
        this.jsonformat = jsonformat;  
    }  
  
    @Override  
    public void readXML() {  
        jsonformat.readJson();  
    }  
}
```

```

    }

    @Override
    public String generateXML() {
        return jsonformat.generateJson();
    }

}

```

- Client "Test"://Q3

```

public class Main {

    public static void main(String[] args) {

        APIClient json=new APIClient();

        APIServer content=new XMLAdapter(json);

        System.out.println(content.generateXML());

    }

}

```

The screenshot shows an IDE with a Java file containing the following code:

```

8  /**
9   *
10  * @author Lenovo
11  */
12  public class Main {
13      public static void main(String[] args) {
14          APIClient json=new APIClient();
15          APIServer content=new XMLAdapter(json);
16          System.out.println(content.generateXML());
17      }
18  }
19

```

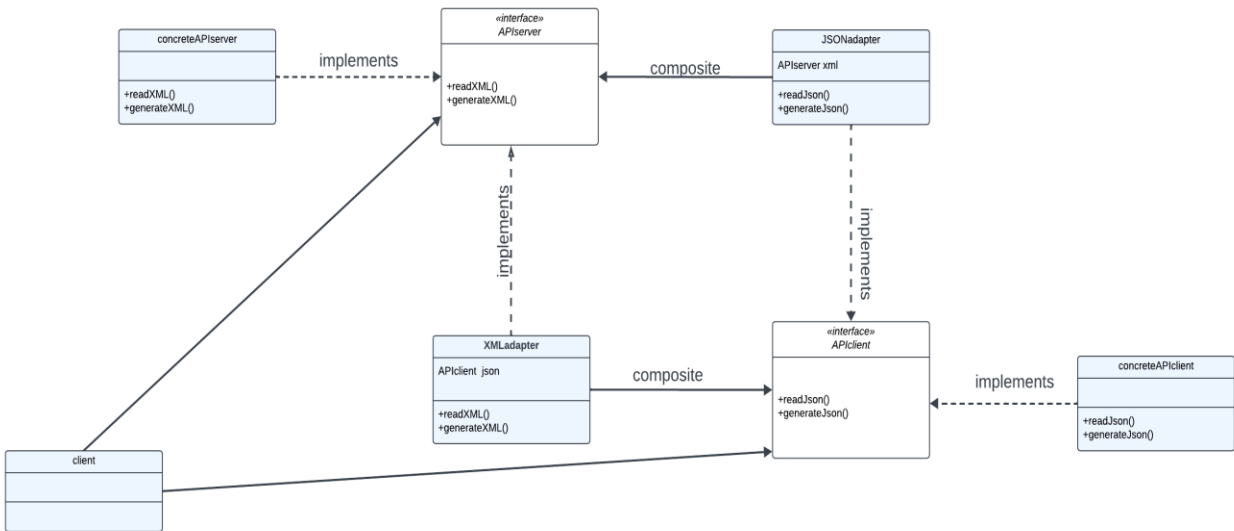
Below the code editor, the 'Output' window is visible, showing the following text:

```

run:
Generating JSON data from client
BUILD SUCCESSFUL (total time: 0 seconds)

```

Another solution that contain one main :



- **API client as “target” in case if get JSON format:**

```

public interface APIClient {
    public void readJson();
    public String generateJson();
}
  
```

- **concreteAPIClient jsust for implimintation of readJson and generateJson:**

```

public class APIClientconcrete implements APIClient{

    public void readJson(){
        System.out.println("Data from client with JSON format");
    }

    public String generateJson(){
        return "Generating JSON data from client";
    }
}
  
```

- **API server as “target” in case if get XML format:**

```
public interface APIserver {  
    public void readXML();  
    public String generateXML();  
}
```

- **concreteAPIserver jsust for implimintation of readXML and generateXML:**

```
public class APIserverconcrete implements APIserver {  
    public void readXML(){  
        System.out.println("Data from server with XML format");  
    }  
    public String generateXML(){  
        return " Generating XML data from server";  
    }  
}
```

- **adapter class:**

```
public class JSONadapter implements APIclient{  
    APIserver xml;  
  
    public JSONadapter(APIserver xml) {  
        this.xml = xml;  
    }  
  
    @Override  
    public void readJson(){  
        xml.readXML();  
    }  
  
    @Override  
    public String generateJson(){  
        return xml.generateXML();  
    }  
}
```

- **adapter class:**

```
public class XMLadapter implements APIserver {  
    APIclient json;  
  
    public XMLadapter(APIclient json) {  
        this.json = json;  
    }  
  
    @Override  
    public void readXML(){  
        json.readJson();  
    }  
}
```



```

    }

    @Override

    public String generateXML(){

    return json.generateJson();

    }

}

```

- Test drive:

```

public static void main(String[] args) {

    APIServer xml=new APIServerconcrete();

    APIClient client=new JSONadapter(xml);

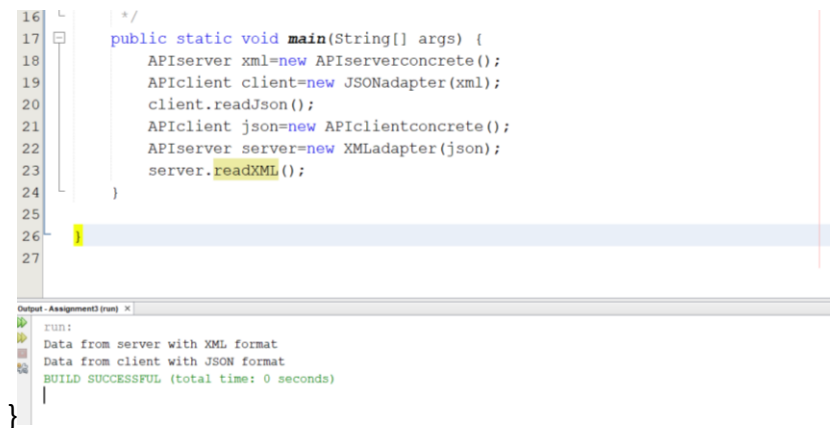
    client.readJson();

    APIClient json=new APIClientconcrete();

    APIServer server=new XMLadapter(json);

    server.readXML();}

```



The screenshot shows an IDE with a Java file. The code defines a `main` method that creates an `APIServer` and an `APIClient` using concrete implementations, then calls `readJson()` and `readXML()` on them. The output window at the bottom shows the results of running the program.

```

16  */
17  public static void main(String[] args) {
18      APIServer xml=new APIServerconcrete();
19      APIClient client=new JSONadapter(xml);
20      client.readJson();
21      APIClient json=new APIClientconcrete();
22      APIServer server=new XMLadapter(json);
23      server.readXML();
24  }
25
26
27

```

Output - Assignment3 (run) x

```

RUN:
Data from server with XML format
Data from client with JSON format
BUILD SUCCESSFUL (total time: 0 seconds)

```