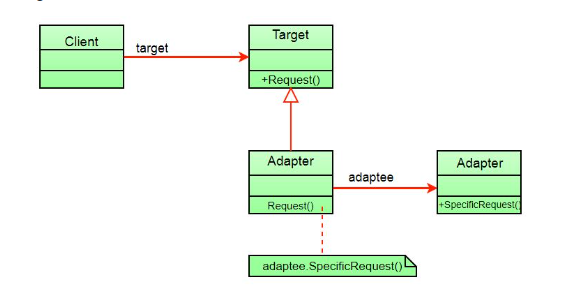
**Structural Design Patterns**

**Adapter Pattern**

* allows objects with incompatible interfaces to collaborate.
* It acts as a bridge between two unrelated, and sometimes even completely incompatible interfaces, similar to how a scanner acts as a bridge between a paper and a computer.
* When you travel from the US to Europe for the first time, you may get a surprise when trying to charge your laptop. The power plug and sockets standards are different in different countries. That’s why your US plug won’t fit a German socket. The problem can be solved by using a power plug adapter that has the American-style socket and the European-style plug.
* The Adapter Pattern is also known as **Wrapper.**
* It is used:
* When an object needs to utilize an existing class with an incompatible interface.
* When you want to create a reusable class that cooperates with classes which don't have compatible interfaces.
* When you want to create a reusable class that cooperates with classes which don't have compatible interfaces.
* adapters are used when we have a class (Client) expecting some type of object and we have an object (Adaptee) offering the same features but exposing a different interface.



**Implementation:**

The Builder interface is our most general interface, and it provides a method that accepts a building type and its location:

|  |
| --- |
| public interface Builder  {  public void build(String type, String location);  } |

The AdvancedBuilder interface provides two methods, one to build a house, and one to build a skyscrapper:

public interface AdvancedBuilder {

public void buildHouse(String location);

public void buildSkyscrapper(String location);

}

These two interfaces are unrelated. Yes, they share the theme, but they're unrelated as far as code is concerned.

public class HouseBuilder implements AdvancedBuilder {

@Override

public void buildHouse(String location) {

System.out.println("Building a house located in the " + location + "area!");

}

@Override

public void buildSkyscrapper(String location) {

//don't implement

}

}

another concrete class is created:

public class SkyscrapperBuilder implements AdvancedBuilder {

@Override

public void buildSkyscrapper(String location) {

System.out.println("Building a skyscrapper in the " + location + "area!");

}

@Override

public void buildHouse(String location) {

//don't implement

}

}

Here comes the adapter part - to connect these two interfaces, a BuilderAdapter implementing Builder is made:

public class BuilderAdapter implements Builder {

AdvancedBuilder advancedBuilder;

public BuilderAdapter(String type) {

if(type.equalsIgnoreCase("House")) {

advancedBuilder = new HouseBuilder();

} else if(type.equalsIgnoreCase("Skyscrapper")) {

advancedBuilder = new SkyscrapperBuilder();

}

}

@Override

public void build(String type, String location) {

if(type.equalsIgnoreCase("House")) {

advancedBuilder.buildHouse(location);

} else if(type.equalsIgnoreCase("Skyscrapper")) {

advancedBuilder.buildSkyscrapper(location);

}

}

}

finally implement the solution and use the Builder interface's method with the BuilderAdapter to build the supported building types.

public class BuilderImplementation implements Builder {

BuilderAdapter builderAdapter;

@Override

public void build(String type, String location) {

if(type.equalsIgnoreCase("House") || type.equalsIgnoreCase("Skyscrapper")) {

builderAdapter = new BuilderAdapter(type);

builderAdapter.build(type, location);

} else {

System.out.println("Invalid building type.");

}

}

}

And to observe the result:

public class Main {

public static void main(String[] args) {

BuilderImplementation builderImpl = new BuilderImplementation();

builderImpl.build("house", "Downtown");

builderImpl.build("Skyscrapper", "City Center");

builderImpl.build("Skyscrapper", "Outskirts");

builderImpl.build("Hotel", "City Center");

}

}

Running the piece of code above will yield:

Building a house located in the Downtown area!

Building a skyscrapper in the City Center area!

Building a skyscrapper in the Outskirts area!

Invalid building type.