Dependency Inversion and Adapters

Back To The Temperature Sensor

Early in the **`WeatherStation`** constructor

```
KelvinTempSensor sensor = new KelvinTempSensor() ;
```

On the surface this looks exactly like **`Barometer`**:

- We create a concrete sensor object in the weather station.
- This limits weather station reusability with different sensors.
- So:
  - Define an interface, say **`IKelvinTempSensor`**.
  - Implement the interface for all real & simulated sensor classes.
  - Create the desired concrete sensor in main or other driver method.
  - Inject this object into the **`WeatherStation`** constructor.

But there is more here than meets the eye!

Problems With The Temperature Sensor

- The interface represents an "odd" notion of what temperature looks like:
  - Scaled integer from 0 to 65535?
  - Measures up to 655.35 °K?
  - That's a weird upper bound - why is it there?
- The designers thought the problem was selecting an integrating the best sensor.

Problems With The Temperature Sensor

- The interface represents an "odd" notion of what temperature looks like:
  - Scaled integer from 0 to 65535?
  - Measures up to 655.35 °K?
  - That's a weird upper bound - why is it there?
- The designers thought the problem was selecting an integrating the best sensor.
- The designers were **WRONG**!
- The **real problem** is how to hide the details of specific sensor used from the weather station.
- All the weather station needs is a general value representing the temperature in some reasonable form.

Design Caveats (Uncle Bob Martin)

- Woe is the designer who prematurely decides on a database, and then finds that flat files would have been sufficient.
- Woe is the designer who prematurely decides upon a web-server, only to find that all the team really needed was a simple socket interface.
- Woe is the team whose designers prematurely impose a framework upon them, only to find that the framework provides powers they don't need and adds constraints they can't live with.
- Blessed is the team whose designers have provided the means by which all these decisions can be deferred until there is enough information to make them. **[CDP!]**
- Blessed is the team whose designers have so isolated them from slow and resource hungry IO devices and frameworks that they can create fast and lightweight test environments.
- Blessed is the team whose designers care about what really matters, and defer those things that don't.

Dependency Inversion Principle

- Low-level components should depend on high-level components, not the other way around.

- OR -

- High-level components should not depend on low-level components. Both should depend on abstractions.
- Abstractions should not depend on details (of low level entities). Details should depend on abstractions.

- OR -

- High-level components control the interface to low-level components.

Dependency Inversion & Temperature Sensors

The **`WeatherStation`** decides on the interface it wants:

```
public interface ITempSensor {
    double getCelsius() ;
}
```

Specific sensors must conform (somehow) to this interface

```
class SoondarSensor implements ITempSensor {
    . . . Soondar specific code . . .
}
class EBestSensor implements ITempSensor {
    . . . EBest specific code . . .
}
```

What About Existing KelvinTempSensor?

Approach #1: Change the code

- Make the **`KelvinTempSensor`** class implement **`ITempSensor`**.
- Change the body of the code to convert from the scaled integer in °K to a double precision number in °C.
- Replace the **reading()** method with **`getCelsius()`**.

What About Existing KelvinTempSensor?

Approach #1: Change the code

- Make the **`KelvinTempSensor`** class implement **`ITempSensor`**.
- Change the body of the code to convert from the scaled integer in °K to a double precision number in °C.
- Replace the **reading()** method with **`getCelsius()`**.

Potential problems:

- Small changes here, but in general there would be many changes.
- We might not have the source code, only a precompiled .class or .jar file.

What About Existing KelvinTempSensor?

Approach #1: Change the code

- Make the **`KelvinTempSensor`** class implement **`ITempSensor`**.
- Change the body of the code to convert from the scaled integer in °K to a double precision number in °C.
- Replace the **reading()** method with **`getCelsius()`**.

Potential problems:

- Small changes here, but in general there would be many changes.
- We might not have the source code, only a precompiled .class or .jar file.

Approach #2: Create an **Adapter**.

Adapters

What are adapters for?

- Have an existing entity (2-prong outlet, temperature sensor class).
- Which does what is required (deliver A/C electricity, provides the temperature).
- But in a way we can't use (no grounding, temperature in scaled Kelvin).
- So we create an adapter (3-prong adapter, temperature adapter class).

In software design, this is the goal of the **Adapter Pattern**.

Reasons for Software Adapters

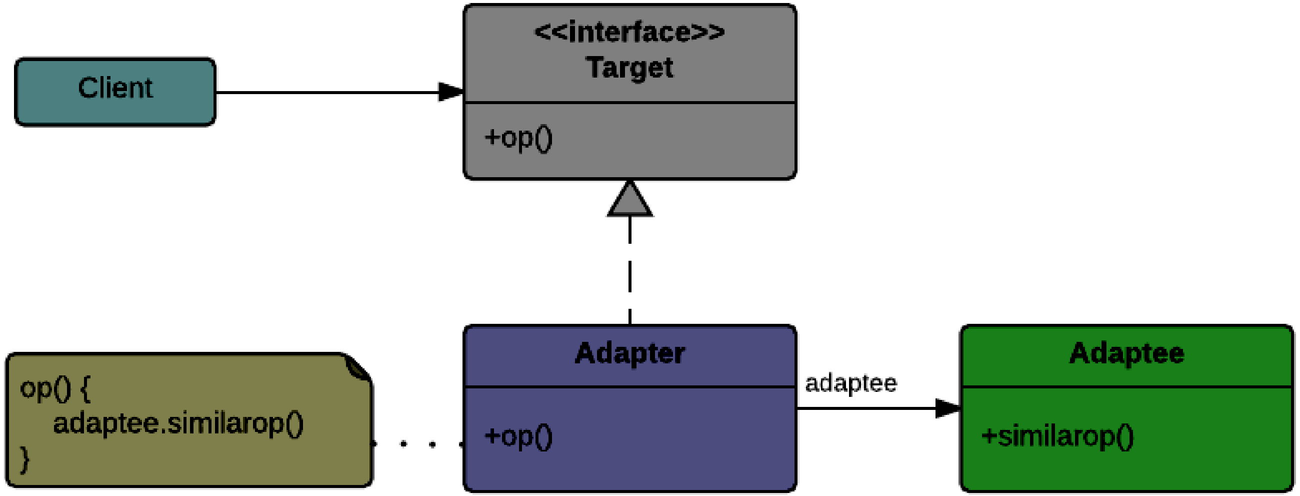
Class of the object we at hand has:

- Different method names.
- Different return types or values.
- Different argument types or counts.
- Different partitioning of class responsibilities.

Usually a combination of the above.

Software Adapter UML

Class Diagram



Client

<<interface>>
Target

+op()

Adapter

+op()

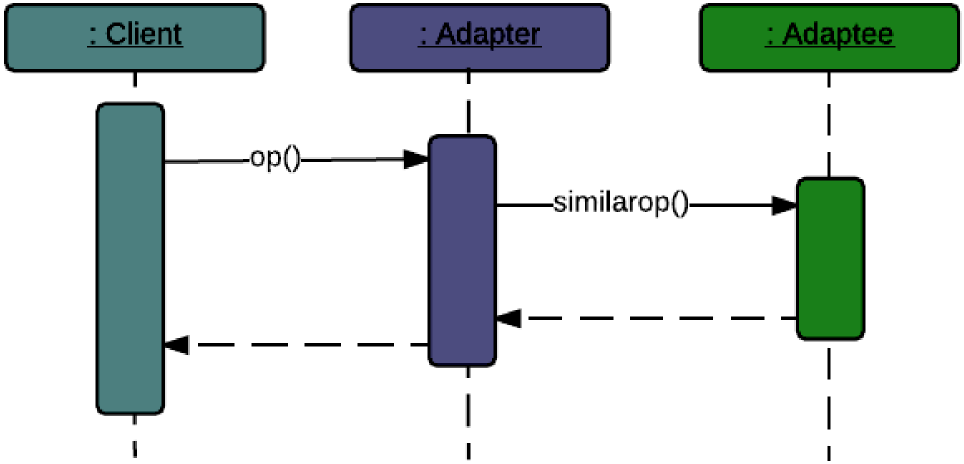
adaptee

Adaptee

+similarop()

```
op() {
    adaptee.similarop()
}
```

Sequence Diagram



: Client

: Adapter

: Adaptee

op()

similarop()

Temperature Sensor Adapter

```
public interface ITempSensor {
    public double getCelsius() ;
}
. . .
public class KTempAdapter implements ITempSensor {
    private KelvinTempSensor kts = new KelvinTempSensor() ;
    private K2C_CONVERT = -27315 ;
    public double getCelsius() {
        return (kts.reading() + K2C_CONVERT) / 100.0 ;
    }
}
```

To use this in our application:

1. Create a **`KTempAdapter`** object in the UI main method.
2. Inject this into the **`WeatherStation`** as a constructor argument.
3. The **`WeatherStation`** argument is, of course, of type **`ITempSensor`**.
4. Change **`WeatherStation`** code dependent on **`KelvinTempSensor`** to use what is returned by the **`ITempSensor`** objects.