



AARHUS UNIVERSITET

# Software Engineering and Architecture

Pattern Catalog: Adapter



- LunaTown requirement
  - The rate should correlate the phases of the moon !
    - Double rates at full moon 😊
  - You must use the implementation bought from a consultancy company (closed source! **final class**)
- Challenge:
  - The interface does not match ours 😞

Fragment: chapter/adapter/src/paystation/thirdparty/LunaRateCalculator.java

```
public int calculateRateForAmount( double dollaramount ) {
```

```
    public interface RateStrategy {  
        /** return the number of minutes parking time the  
         * provided amount of payment is valid for.  
         * @param amount payment in some currency.  
         * @return number of minutes parking time  
         */  
        public int calculateTime( int amount );  
    }
```



- ③-①-②: We have done almost all the steps
  - ③: Encapsulate what varies (rate calculations)
  - ①: We have the RateStrategy interface
- But we cannot ② because the provided rate calculator does *not* implement RateStrategy!
- But we can ② *compose behavior* even further to solve the problem



# Solution

- I will *put an intermediate object between the two*, one that does the *translation* from one interface to the other:

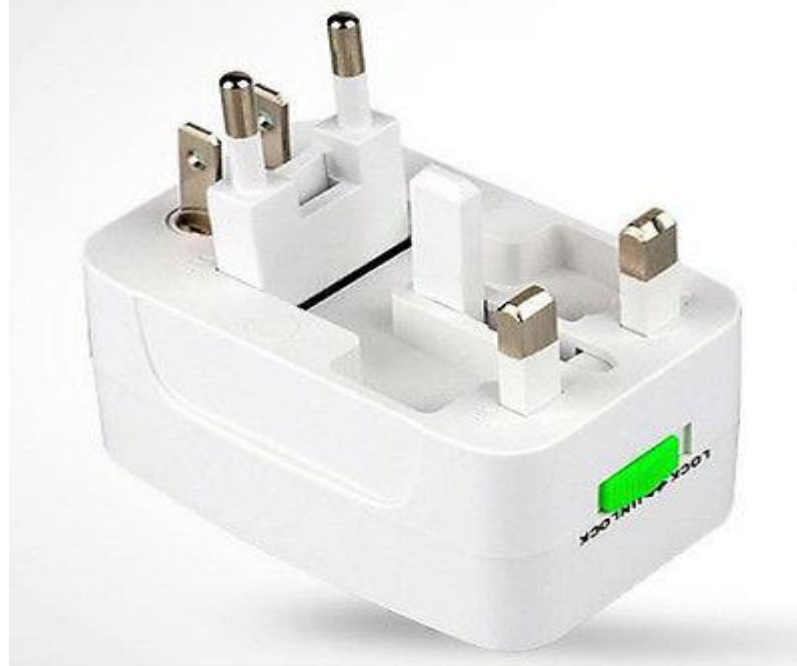
```
/** An adapter for adapting the Lunatown rate calculator
 */
public class LunaAdapter implements RateStrategy {
    private LunaRateCalculator calculator;
    public LunaAdapter() {
        calculator = new LunaRateCalculator();
    }

    public int calculateTime( int amount ) {
        double dollar = amount / 100.0;
        return calculator.calculateRateForAmount( dollar );
    }
}
```



AARHUS UNIVERSITET

# Analogy

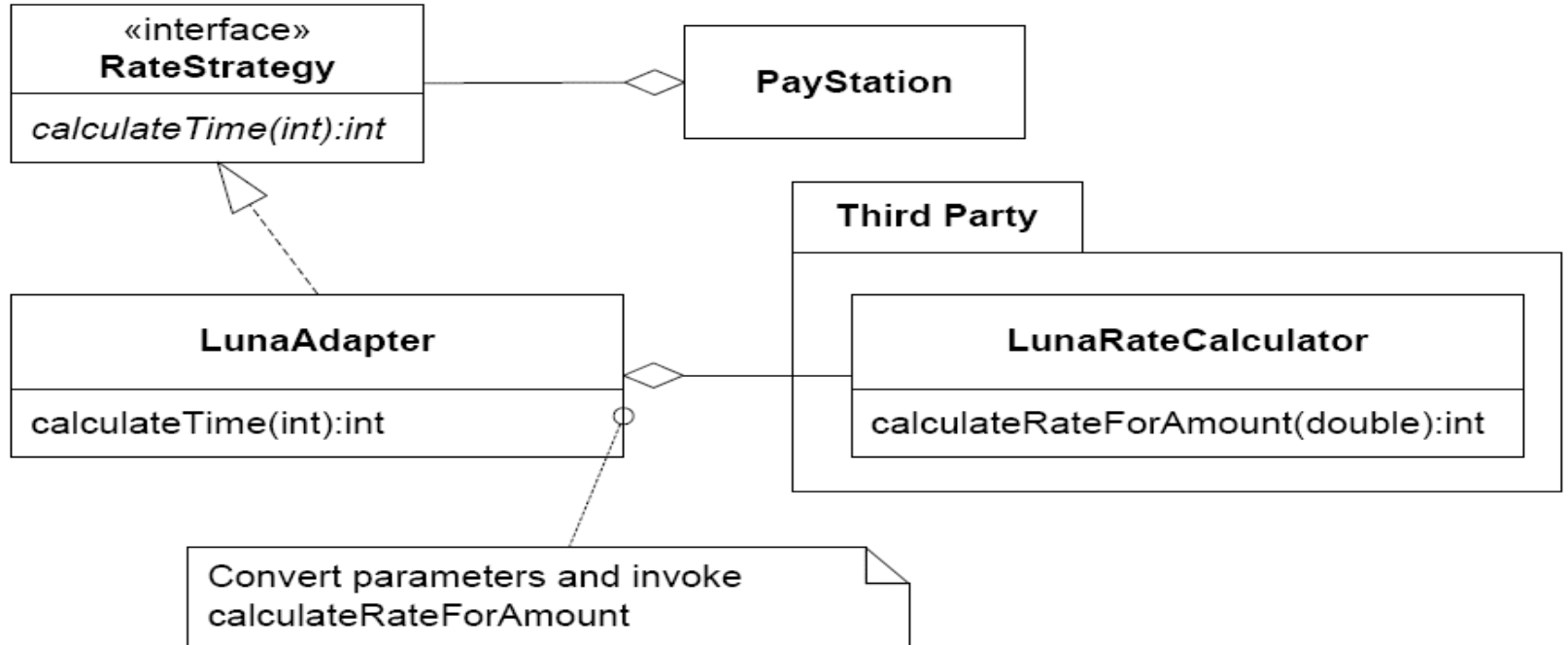


# The Beijing version 😊





# Structure of our solution





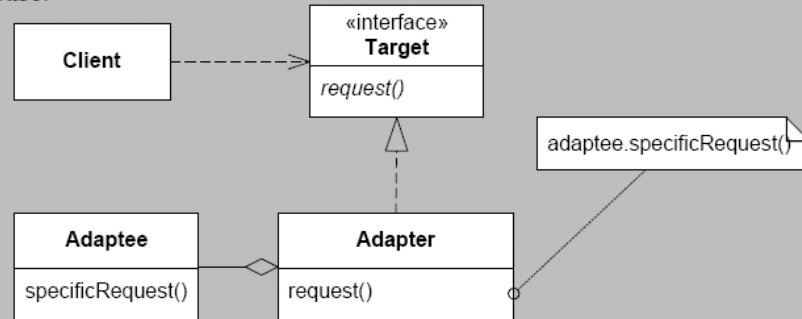
## [21.1] Design Pattern: Adapter

**Intent** Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

**Problem** You have a class with desirable functionality but its interface and/or protocol does not match that of the client needing it.

**Solution** You put an intermediate object, the adapter, between the client and the class with the desired functionality. The adapter conforms to the interface used by the client and delegate actual computation to the adaptee class, potentially performing parameter, protocol, and return value translations in the process.

**Structure:**



**Roles** Target encapsulates behavior used by the Client. The Adapter implements the Target role and delegate actual processing to the Adaptee performing parameter and protocol translations in the process.

**Cost - Benefit** Adapter lets objects collaborate that otherwise are incompatible. A single adapter can work with many adaptees—that is, all the adaptee's subclasses.





# Consequences

- **Benefits**
  - Makes a client work with an otherwise incompatible object
  - One adapter can adapt many type of adaptee's namely all subclasses
- **Liabilities**
  - Adaptation spectrum: from simple method name conversions to radically different interfaces
    - Adapters for gui toolkits