CS 525 - ASD Advanced Software Development

MS.CS Program

Department of Computer Science Rene de Jong, MsC.



CS 525 - ASD Advanced Software Development

© 2019 Maharishi University of Management

All course materials are copyright protected by international copyright laws and remain the property of the Maharishi University of Management. The materials are accessible only for the personal use of students enrolled in this course and only for the duration of the course. Any copying and distributing are not allowed and subject to legal action.



Lesson 8 Adapter pattern

L1: ASD Introduction

L2: Strategy, Template method

L3: Observer pattern

L4: Composite pattern, iterator pattern

L5: Command pattern

L6: State pattern

L7: Chain Of Responsibility pattern

Midterm

L8: Proxy, Adapter, Mediator

L9: Factory, Builder, Decorator, Singleton

L10: Framework design

L11: Framework implementation

L12: Framework example: Spring framework

L13: Framework example: Spring framework

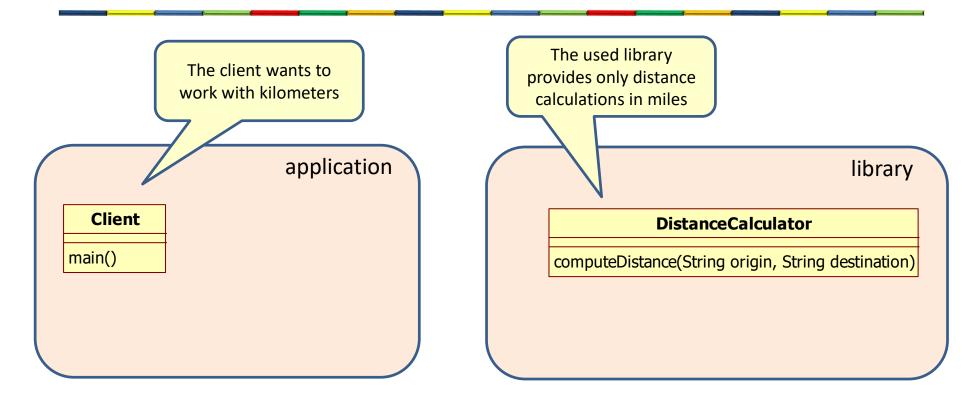
Final

Adapter pattern

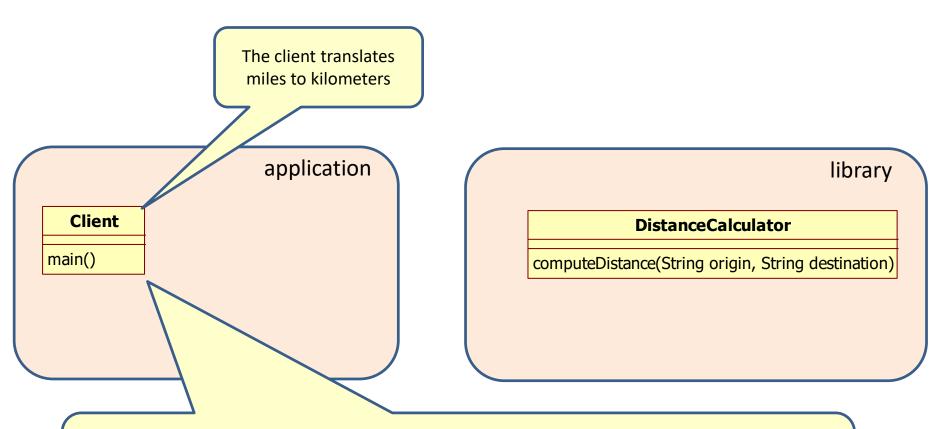
- Translates the existing interface of a class into an interface that the client requires.
 - (Reusable) wrapper



Design problem



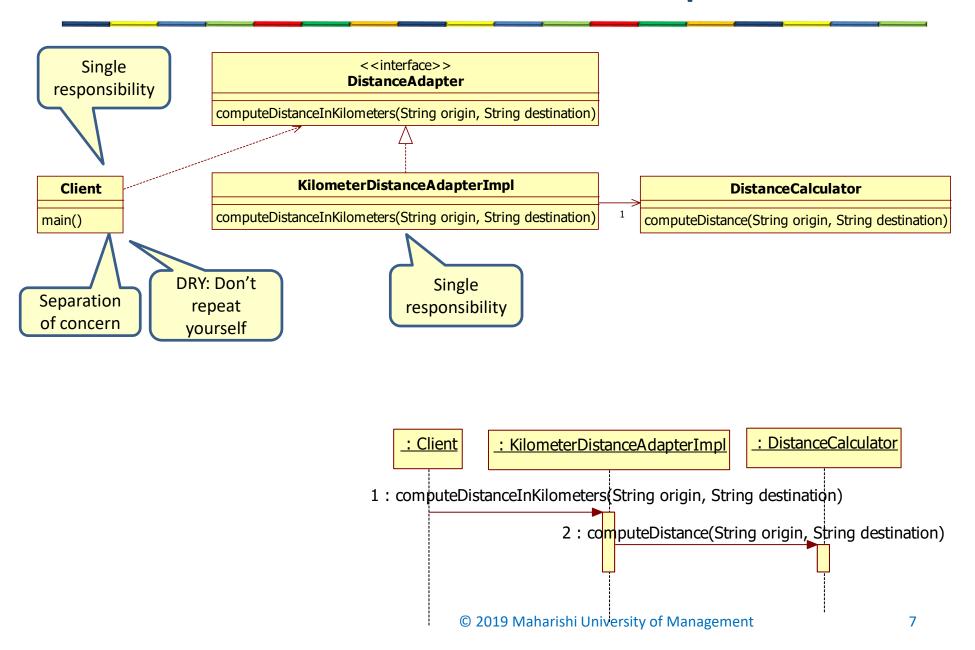
Solution



Disadvantages:

- Translation from miles to kilometers is not reusable.
 Every client that uses this library has to implement the translation (DRY: don't repeat yourself)
- 2. The client does multiple things (single responsibility)

Better solution: Adapter



Adapter + Adaptee

```
public class DistanceCalculator {
  public double computeDistance(String origin, String destination) {
    return (new Random()).nextInt(100);
  }
}
```

```
public interface DistanceAdapter {
  double computeDistanceInKilometers(String origin, String destination);
  void setDistanceCalculator(DistanceCalculator distanceCalculator);
}
```

```
public class KilometerDistanceAdapterImpl implements DistanceAdapter {
   private DistanceCalculator distanceCalculator;

   public double computeDistanceInKilometers(String origin, String destination) {
      double distanceInMiles = distanceCalculator.computeDistance(origin, destination);
      return distanceInMiles * 1.609344;
   }

   public void setDistanceCalculator(DistanceCalculator distanceCalculator) {
      this.distanceCalculator = distanceCalculator;
   }
}
```

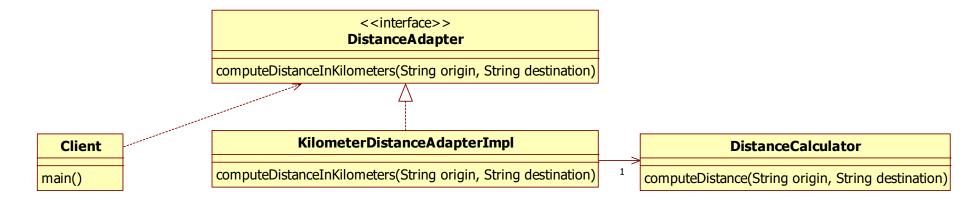
Client

```
public class Client {

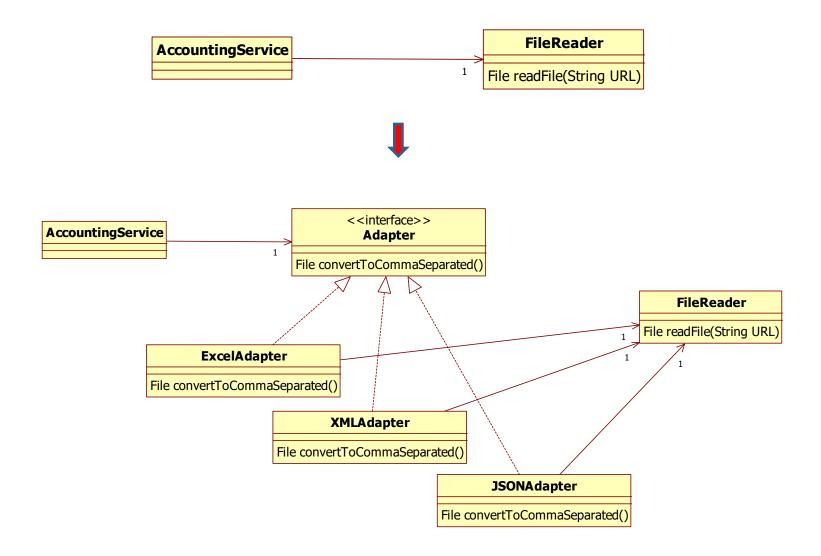
public static void main(String[] args) {
    DistanceCalculator distanceCalculator = new DistanceCalculator();
    double distanceInMiles = distanceCalculator.computeDistance("city1", "city2");
    System.out.println("The distance between city1 and city2 ="+distanceInMiles+" miles");

DistanceAdapter distanceAdapter = new KilometerDistanceAdapterImpl();
    distanceAdapter.setDistanceCalculator(distanceCalculator);

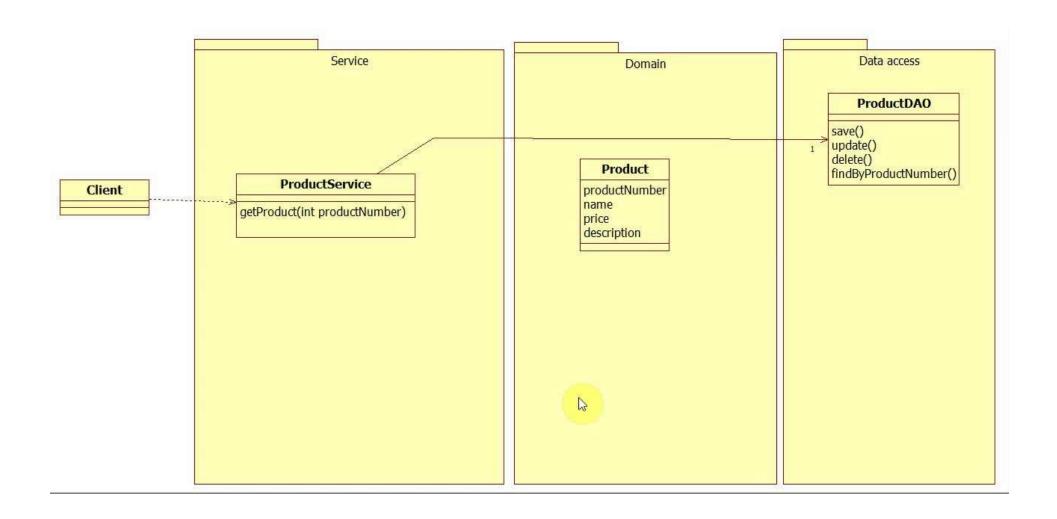
double distanceInKilometers = distanceAdapter.computeDistanceInKilometers("city3", "city4");
    System.out.println("The distance between city3 and city4 ="+distanceInKilometers+" kilometers");
}
```



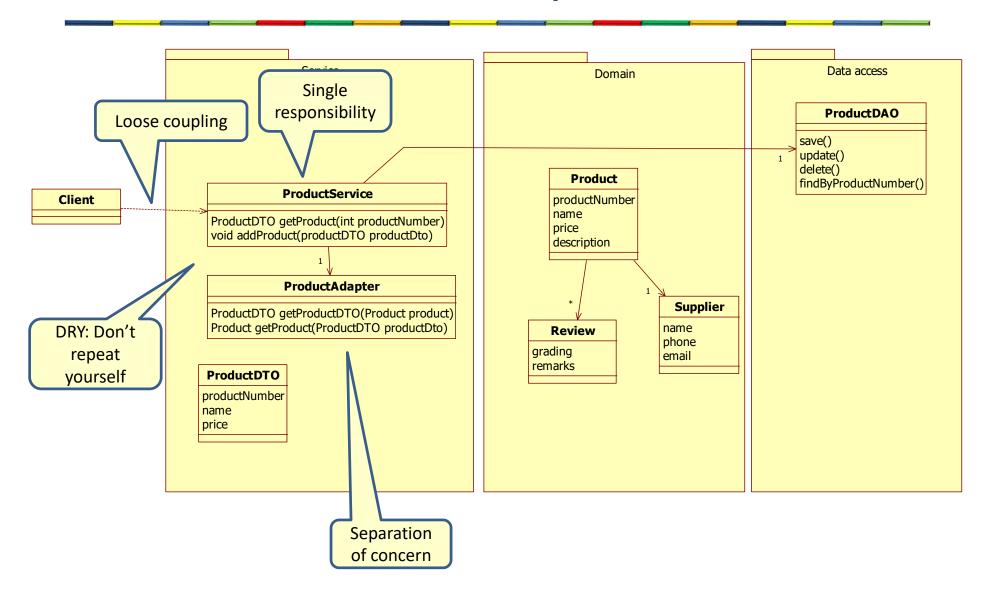
Where are adapters used



Where are adapters used



Where are adapters used



Adapter and Proxy: wrapper

Proxy

Same interface as dynamic proxy

Client

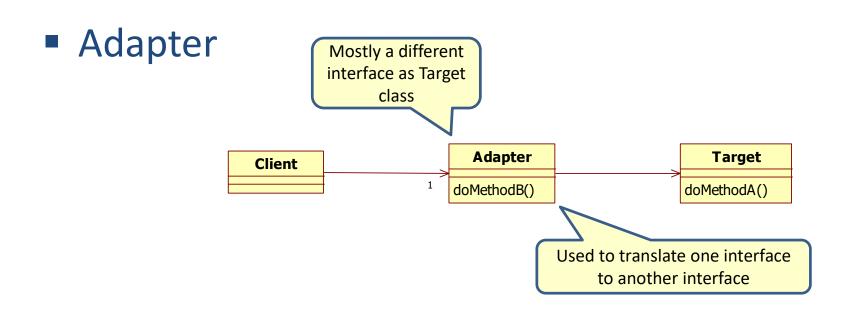
Proxy

Target

doMethodA()

Target

doMethodA()



Used to add logic

Main point

- The Adapter translates an existing interface to a required interface.
- Life is found in layers, from the most abstract transcendental layer (Unified Field) to the most concrete layer.