Added by Nate Austin, last edited by David S Wood on Dec 07, 2006 (view change)
Labels: (None) EDIT

# Parking Lot - Adapter

## Session Objectives

By the end of this session students should ...

- 

## Session Overview

| Activity |
| --- |
| Lab - Impound Lot |
| Adapter Pattern |

## Session Notes

### Lab - Impound Lot

Tell the students that you would like your Attendant to be able to manage the impound lots of an evil towing company (and still manage the existing lots). Give them the appropriate jar file to include in their project.

Versions

- DEMO:Java 1.4
- DEMO:Java 5
- DEMO:Test source

Discussion
The source code for the ImpoundLot and related exceptions will not be provided. Source code for ImpoundLotTest will be, however. Talk about unit tests describing and documenting the functionality of the code under test.

Things to look for

- Conditional logic within the Attendant
- Want no modifications to Attendant at all

If they do not implement as Adapter of ImpoundLot, elicit ideas on how we can do this without the conditionals/without modifying Attendant. Send them back to reimplement.

### Adapter Pattern

**Elicit the formal description of the Adapter Pattern:**

Title Unable to render embedded object: File

- Adapter/Wrapper

(Parking Lot [Adapter.png]) not found.

Problem

- Code that you don't control performs a function that you would like to use. Its interface is different from your existing infrastructure, so adapter wraps this with your interface to allow it to function seamlessly in your environment.

Pattern/Solution

- Can be used to unify two interfaces to fit code you don't control within your existing code or to isolate version-specific code
- Exceptions should also be handled internally and replaced with exceptions known to the outside world
- Occasionally it may be necessary to duplicate state from the adaptee inside the adapter for things like observer

Examples

- In Swing: JTree allows all different types of nodes in its tree (text, images, objects, etc.), JTable does the same (drop downs, editable areas, images, text, back it by a database, etc.)

Benefits/Liabilities

- Amount of work required depends on how similar the two interfaces are.
- Separates library version-specific code
- Can hide important behavior from the client
- Tends to generalize code to allow further refactoring/elimination of duplication

Refactoring steps

- Extract interface from the existing class's interface you'd like to use
- Change all references to use this new interface
- Implement these methods to perform the proper functions using the class you're adapting
- Test using the new adapted class