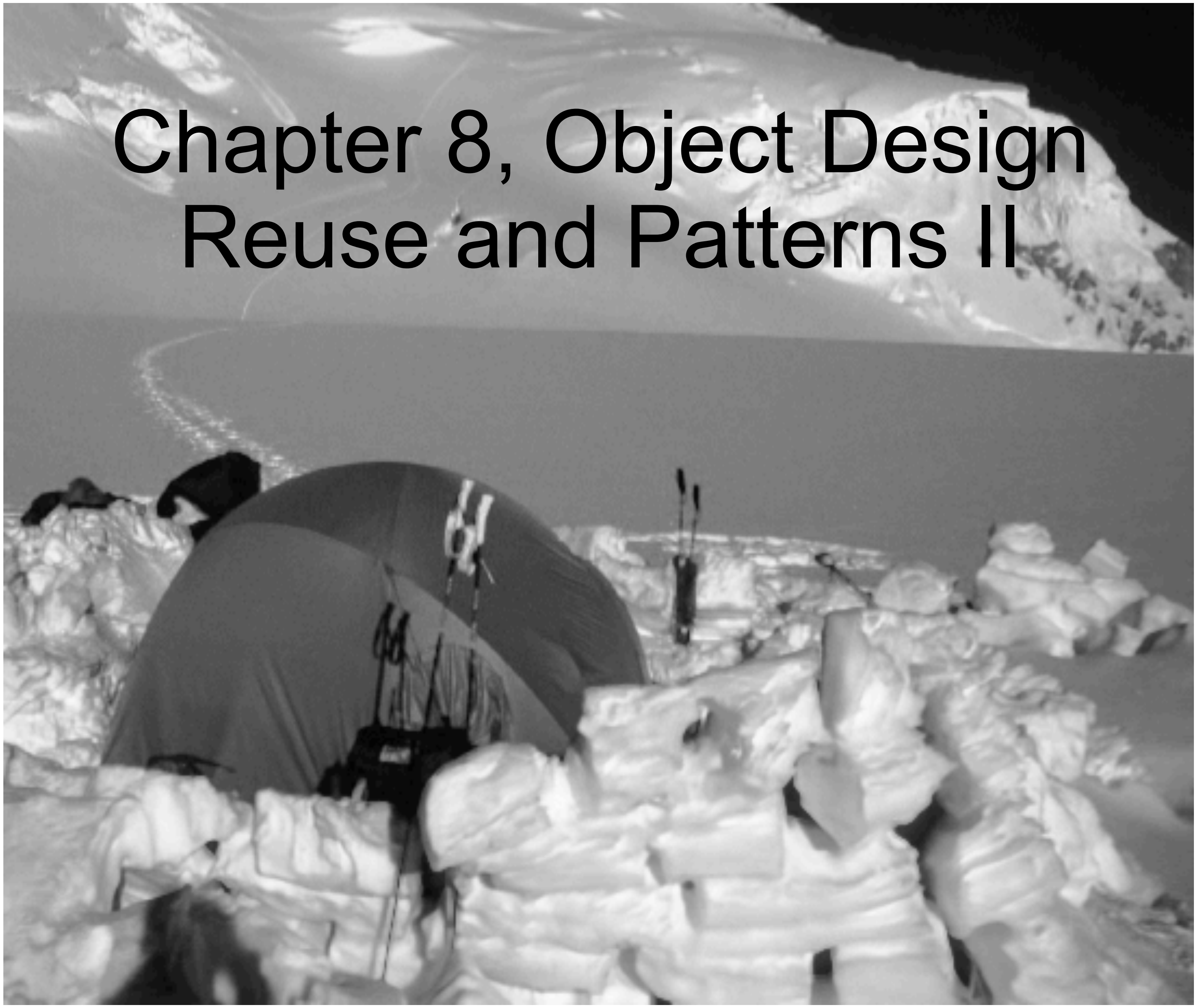
Object-Oriented Software Engineering
Using UML, Patterns, and Java



Chapter 8, Object Design Reuse and Patterns II

Use design patterns to keep system models simple

- ♦ During Object Modeling we do many transformations and changes to the object model.
- ♦ It is important to make sure the object design model stays simple!

Outline of the Lecture

- ♦ Design Patterns
  - Usefulness of design patterns
  - Design Pattern Categories
- ♦ Patterns covered
  1. Composite: Model dynamic aggregates
  2. Facade: Interfacing to subsystems
  3. Adapter: Interfacing to existing systems (legacy systems)
  4. Bridge: Interfacing to existing and future systems
  5. Abstract Factory
  6. Proxy
  7. Command
  8. Strategy

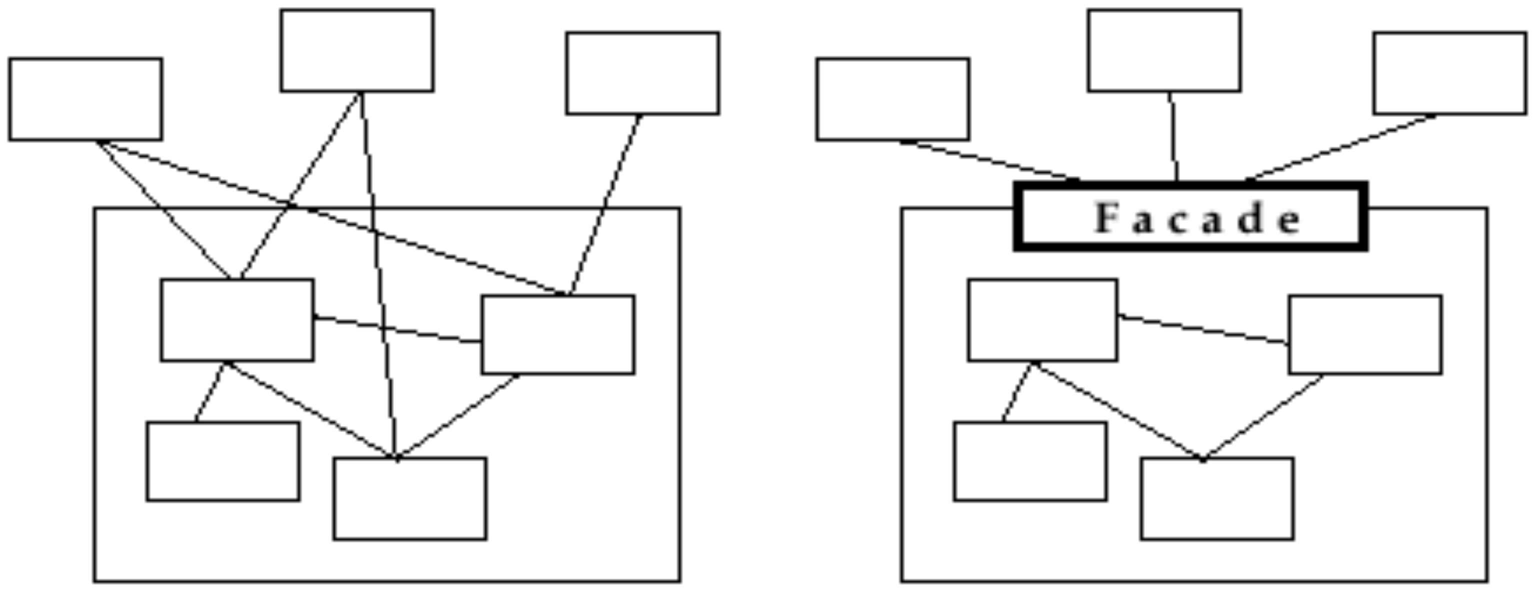
A design pattern has four elements:

1. Name
2. Problem description
3. Solution:

   stated as a set of collaboration classes and interfaces
4. Consequences

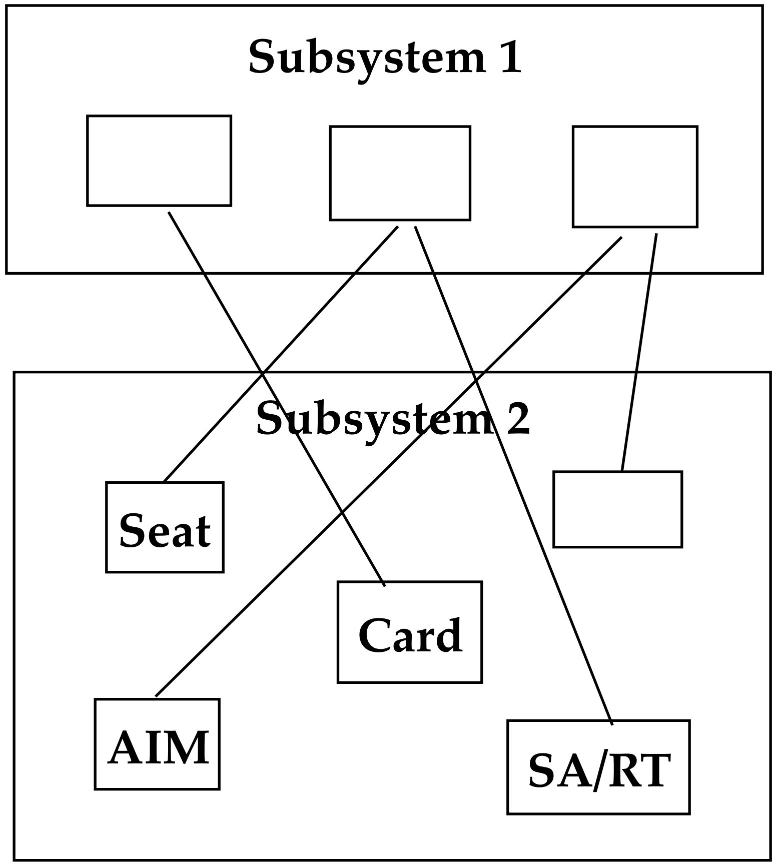
Facade Pattern

- Provides a unified interface to a set of objects in a subsystem.
- A facade defines a higher-level interface that makes the subsystem easier to use (i.e. it abstracts out the gory details)
- Facades allow us to provide a closed architecture



Design Example

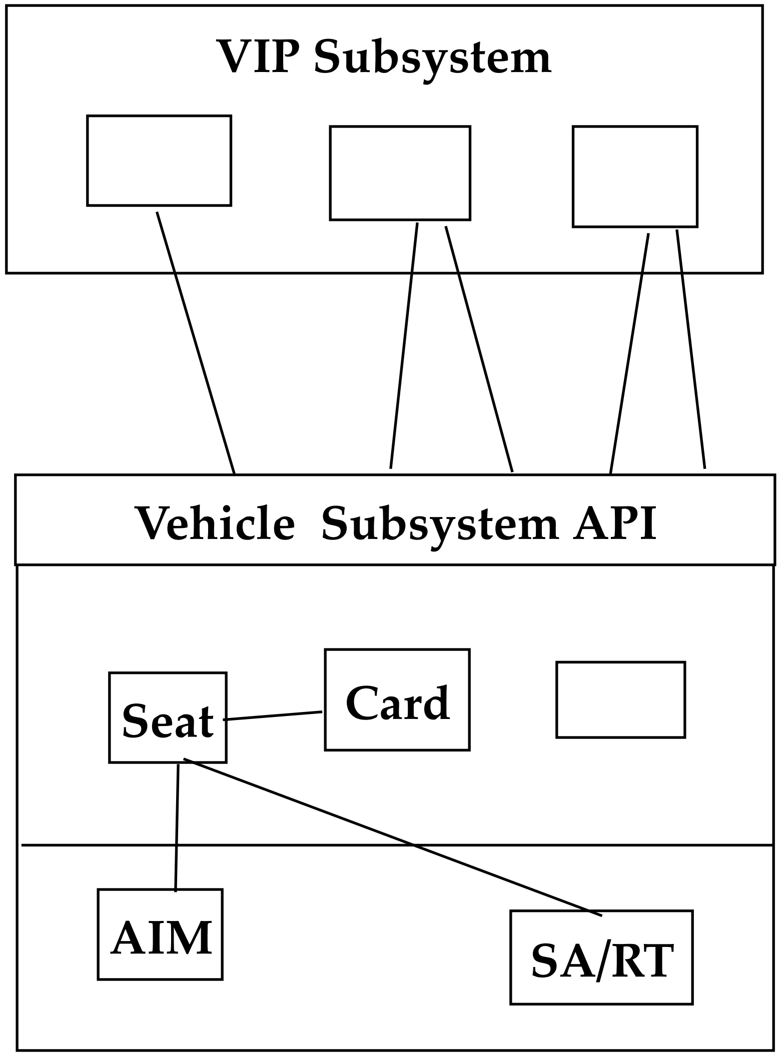
- Subsystem 1 can look into the Subsystem 2 (vehicle subsystem) and call on any component or class operation at will.
- This is “Ravioli Design”
- Why is this good?
  - Efficiency
- Why is this bad?
  - Can’t expect the caller to understand how the subsystem works or the complex relationships within the subsystem.
  - the subsystem will be misused, leading to non-portable code



Realizing an Opaque Architecture with a Facade

- The subsystem decides exactly how it is accessed.
- No need to worry about misuse by callers
- If a facade is used

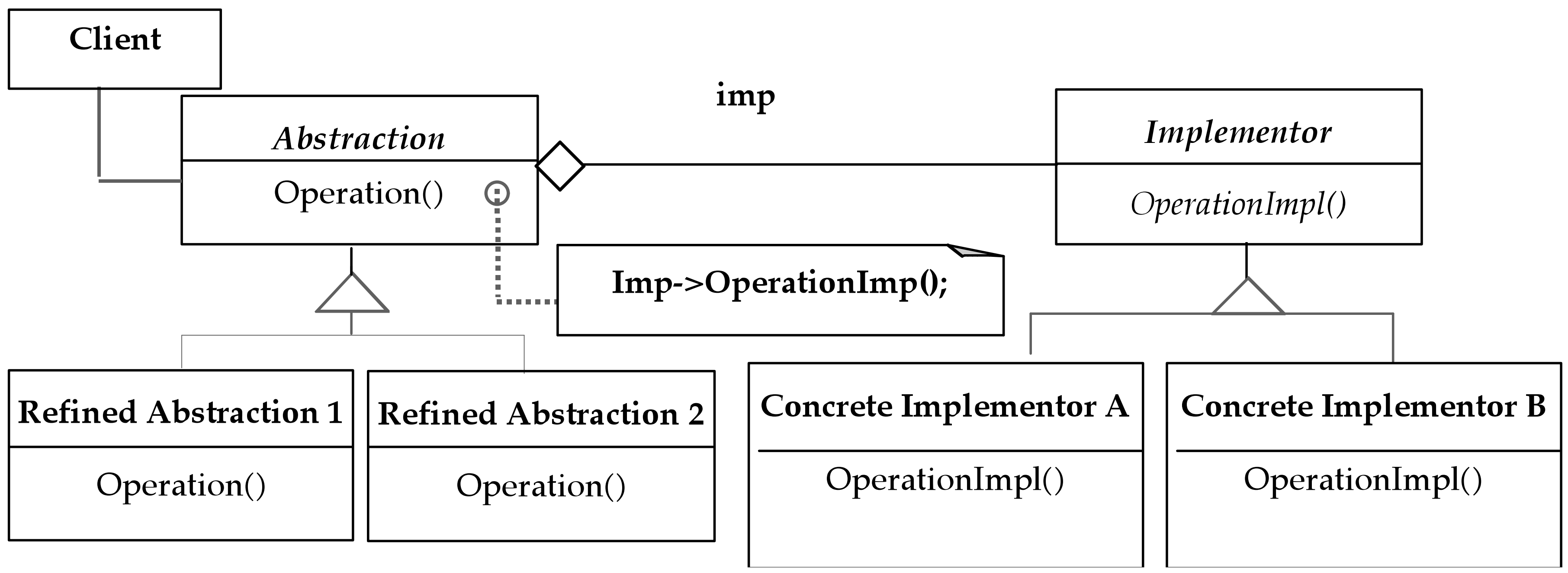
  the subsystem can be used in an early integration test
  - We need to write only a driver



Bridge Pattern

- ♦ Use a bridge to “decouple an abstraction from its implementation so that the two can vary independently”. (From [Gamma et al 1995])
- ♦ Also know as a Handle/Body pattern.
- ♦ Allows different implementations of an interface to be decided upon dynamically.

Bridge Pattern



Design Patterns encourage reusable Designs

- A facade pattern should be used by all subsystems in a software system. It defines all the services of the subsystem.
  - The facade will delegate requests to the appropriate components within the subsystem. Most of the time the facade does not need to be changed. When the component is changed,

  Reduce coupling
- Adapters should be used to interface to existing components.
  - For example, a smart card software system should provide an adapter for a particular smart card reader and other hardware that it controls and queries.

  Reuse
- Bridges should be used to interface to a set of objects
  - where the full set is not completely known at analysis or design time.
  - when the subsystem must be extended later after the system has been deployed and client programs are in the field(dynamic extension).

  Extensibility

Summary

- Design patterns are partial solutions to common problems
  - such as separating an interface from a number of alternate implementations
  - wrapping around a set of legacy classes
  - protecting a caller from changes associated with specific platforms.

- A design pattern is composed of a small number of classes
  - use delegation and inheritance
  - provide a robust and modifiable solution.

- These classes can be adapted and refined for the specific system under construction.
  - Customization of the system
  - Reuse of existing solutions

- Composite Pattern:
  - Models trees with dynamic width and dynamic depth
  - Recursive part-whole hierarchy
- Facade Pattern:
  - Interface to a subsystem
  - Reduce coupling (encapsulating subsystems)
- Adapter Pattern:
  - Interface to reality
  - Reuse (wrapping around legacy code)
- Bridge Pattern:
  - Interface to reality and prepare for future
  - Extensible (allow alternate implementation)