

# Software Engineering I: Software Technology

WS 2008/09

## *Object Design*

Bernd Bruegge  
*Applied Software Engineering*  
*Technische Universitaet Muenchen*

# Details for the Mid-Term

- Date 18 December 2008
- Location: MW 0001
- Duration 18:00 to 19:30

Please arrive in time so we can start at 18:00

- Closed book exam
- Format: Paper-based, handwritten notes
- Questions about definitions and/or modeling activities from material Lecture 1 to the lecture on Dec 16.
- Questions in English
- Answers in English or German

# Outline of Today

- Definition: Object Design vs Detailed Design
- System Design vs Object Design
- Object Design Activities
- Reuse examples
  - Whitebox and Blackbox Reuse
- Object design leads also to new classes
- Implementation vs Specification Inheritance
- Inheritance vs Delegation
- Class Libraries and Frameworks
- Exercises: Documenting the Object Design
  - JavaDoc, Doxygen

# Object Design

- Purpose of object design:
  - Prepare for the implementation of the analysis model based on system design decisions
  - Transform analysis and system design models
- Investigate alternative ways to implement the analysis model
  - Use design goals: minimize execution time, memory and other measures of cost.
- Object Design serves as the basis of implementation

# Terminology: Naming of Design Activities

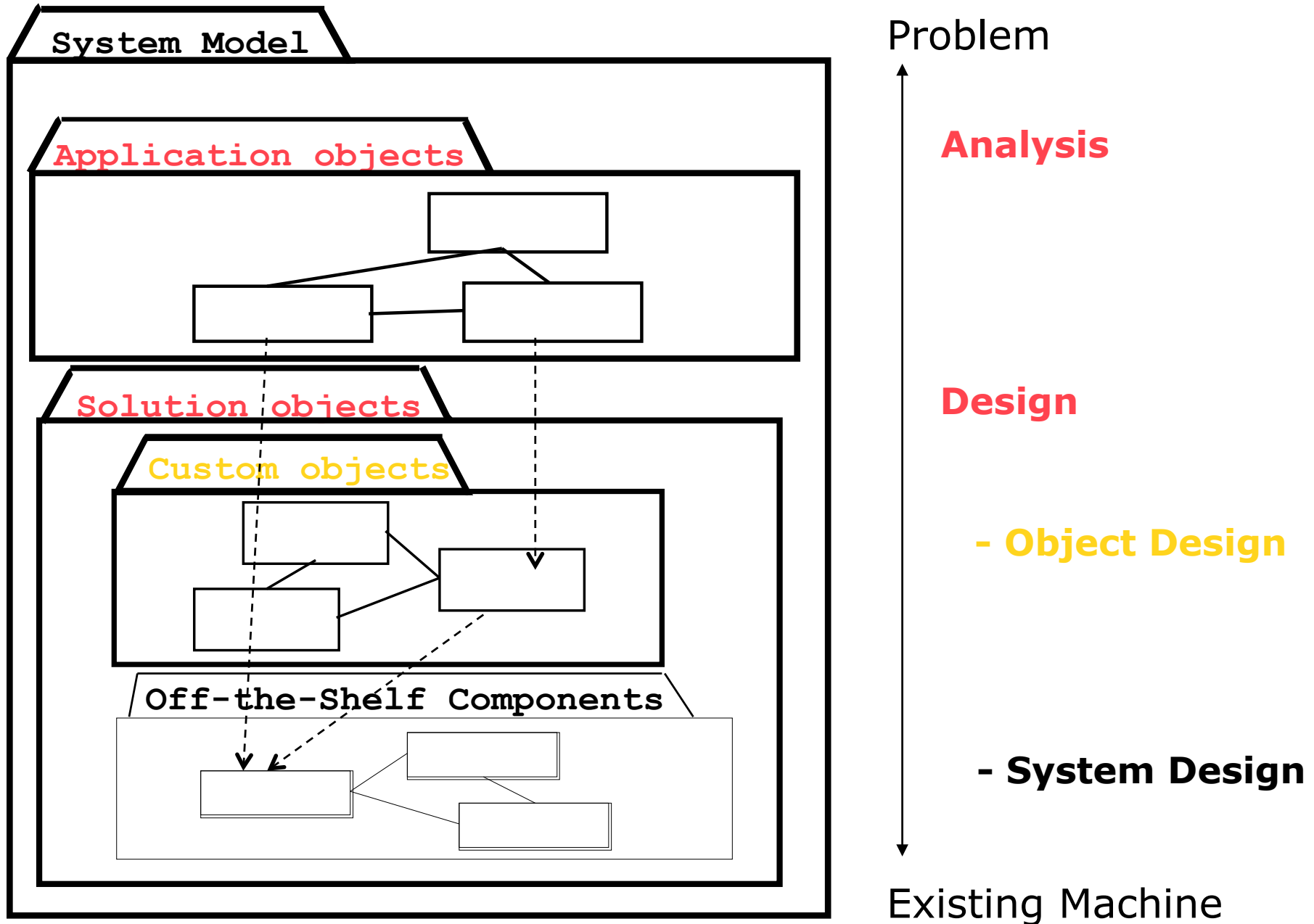
## Methodology: Object-oriented software engineering (OOSE)

- *System Design*
  - Decomposition into subsystems, etc
- *Object Design*
  - Data structures and algorithms chosen
- *Implementation*
  - Implementation language is chosen

## Methodology: Structured analysis/structured design (SA/SD)

- *Preliminary Design*
  - Decomposition into subsystems, etc
  - Data structures are chosen
- *Detailed Design*
  - Algorithms are chosen
  - Data structures are refined
  - Implementation language is chosen.

# System Development as a Set of Activities



# Design means “Closing the Gap”



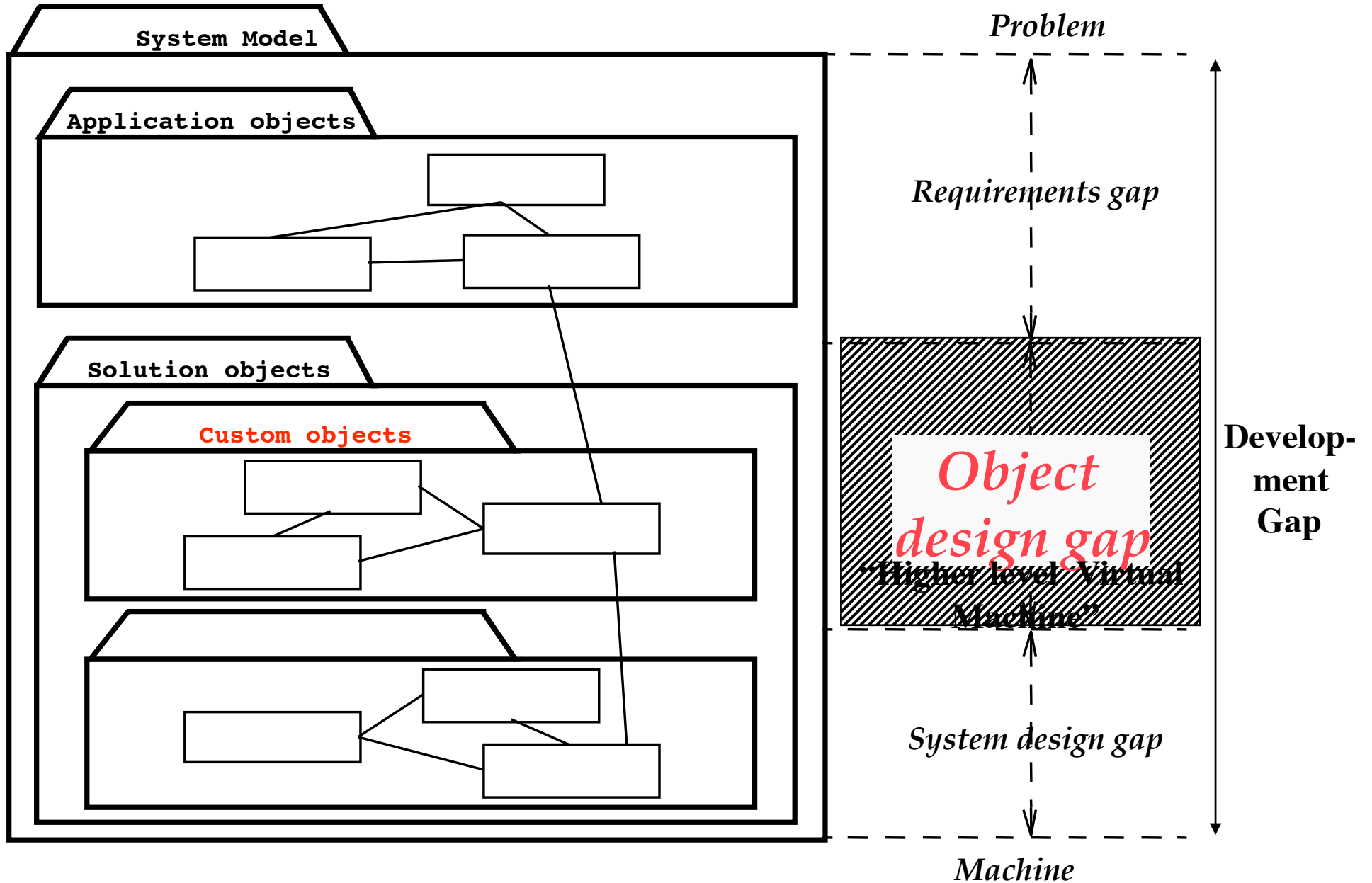
“Subsystem 1”: Rock material from the Southern Sierra Nevada mountains (moving north)

**Example of a Gap:  
San Andreas Fault**

“Subsystem 3” closing the Gap:  
San Andreas Lake

“Subsystem 2”: San Francisco Bay Area

# Design means "Closing the Gap"





# Object Design consists of 4 Activities

## 1. Reuse: Identification of existing solutions

- Use of inheritance
- Off-the-shelf components and additional solution objects
- Design patterns

## 2. Interface specification

- Describes precisely each class interface

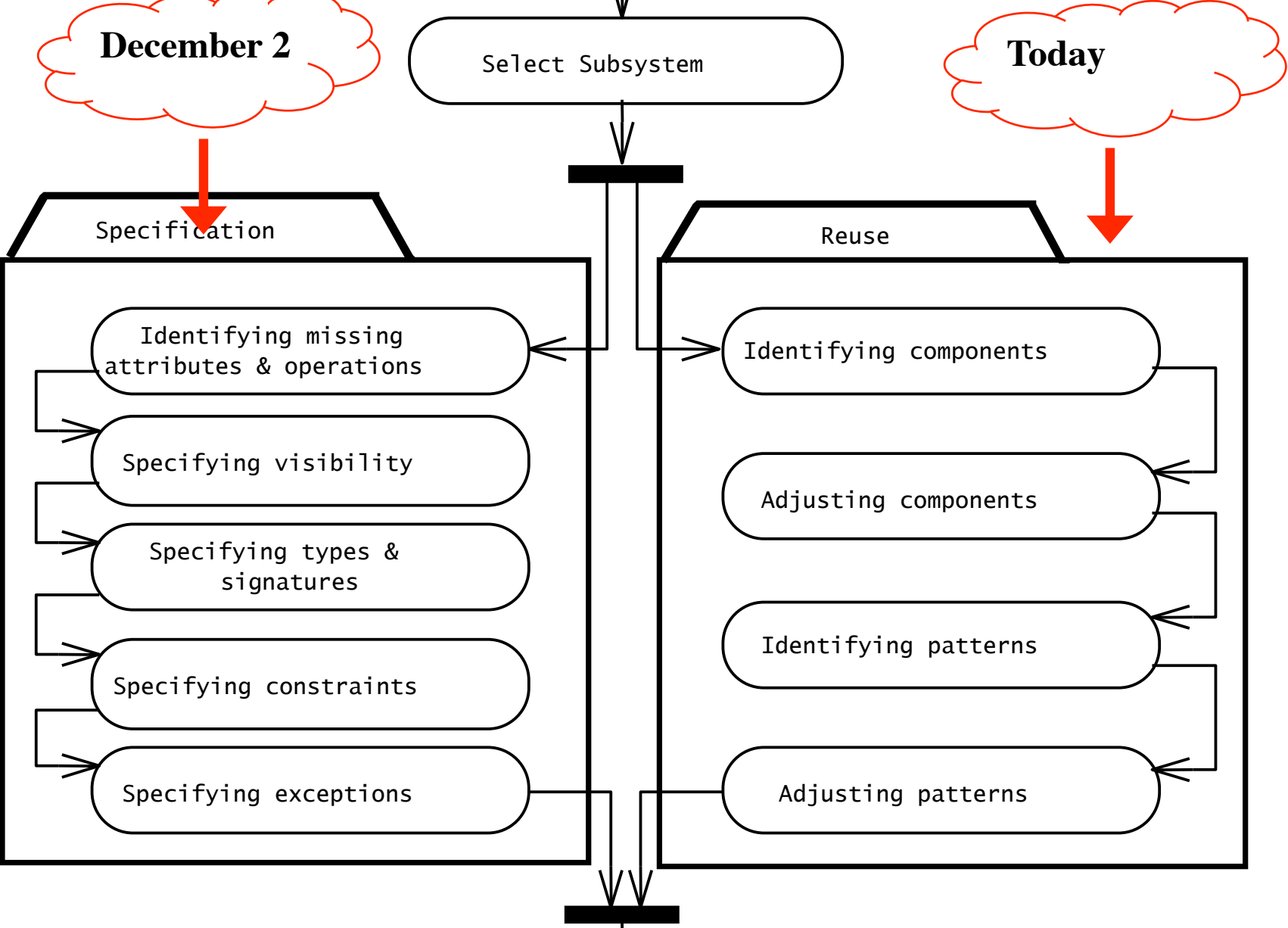
## 3. Object model restructuring

- Transforms the object design model to improve its understandability and extensibility

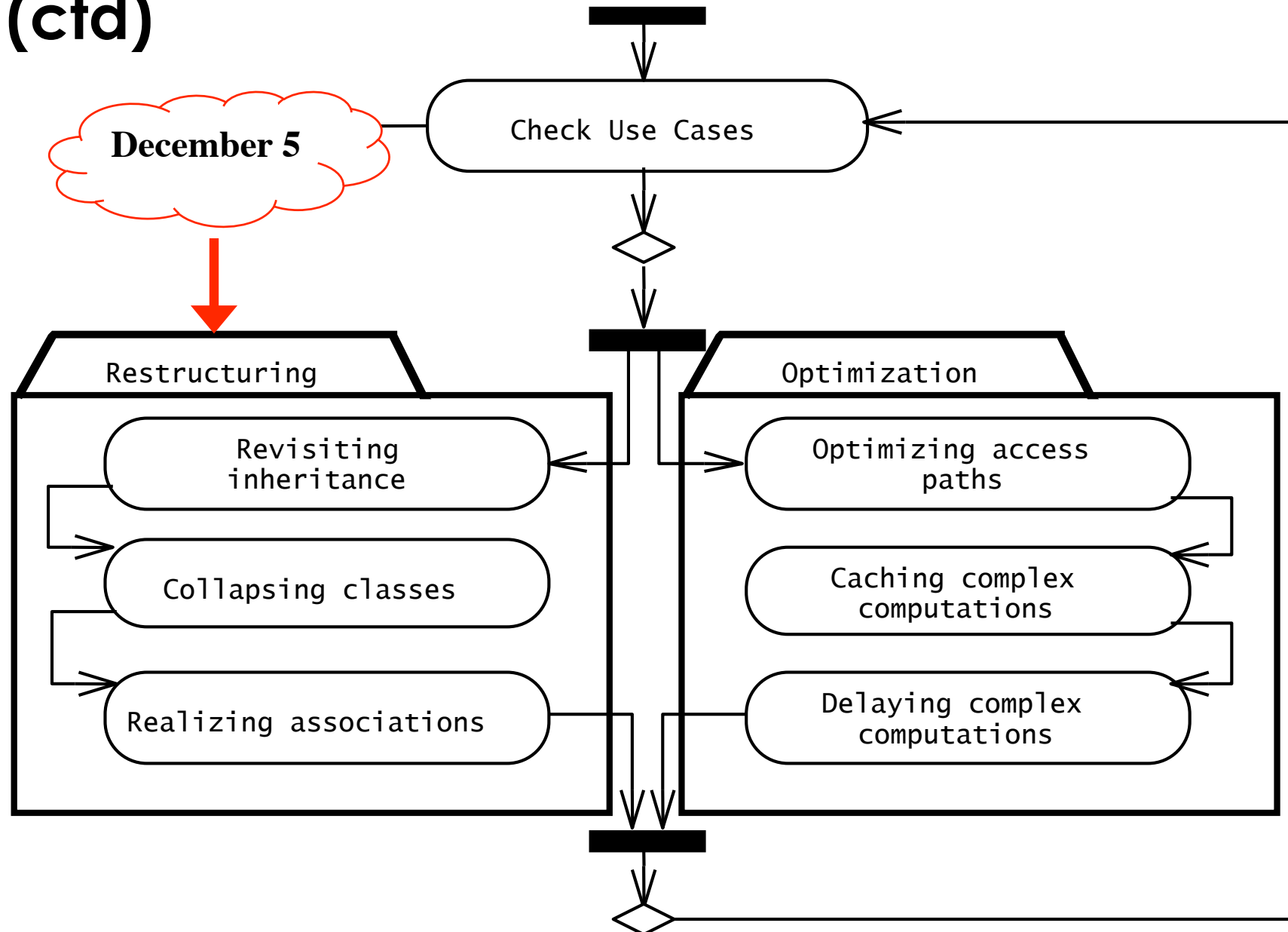
## 4. Object model optimization

- Transforms the object design model to address performance criteria such as response time or memory utilization.

# Object Design Activities



# Detailed View of Object Design Activities (ctd)



# One Way to do Object Design

1. Identify the missing components in the design gap
2. Make a build or buy decision to obtain the missing component

=> **Component-Based Software Engineering:**

The design gap is filled with available components ("0 % coding").

- **Special Case: COTS-Development**
  - COTS: Commercial-off-the-Shelf
  - The design gap is completely filled with commercial-off-the-shelf-components.

=> **Design with standard components.**

# Design with Standard Components is like solving a Traditional Jigsaw Puzzle



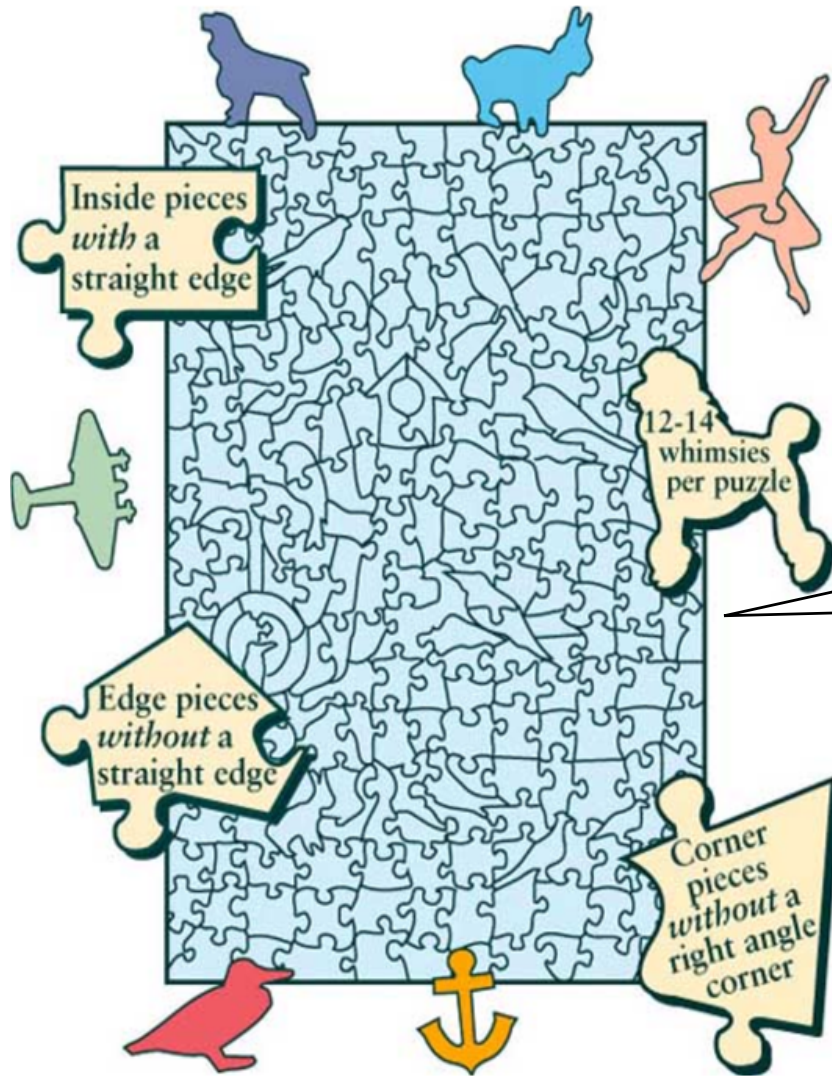
Remaining  
puzzle piece  
("component")



## Design Activities:

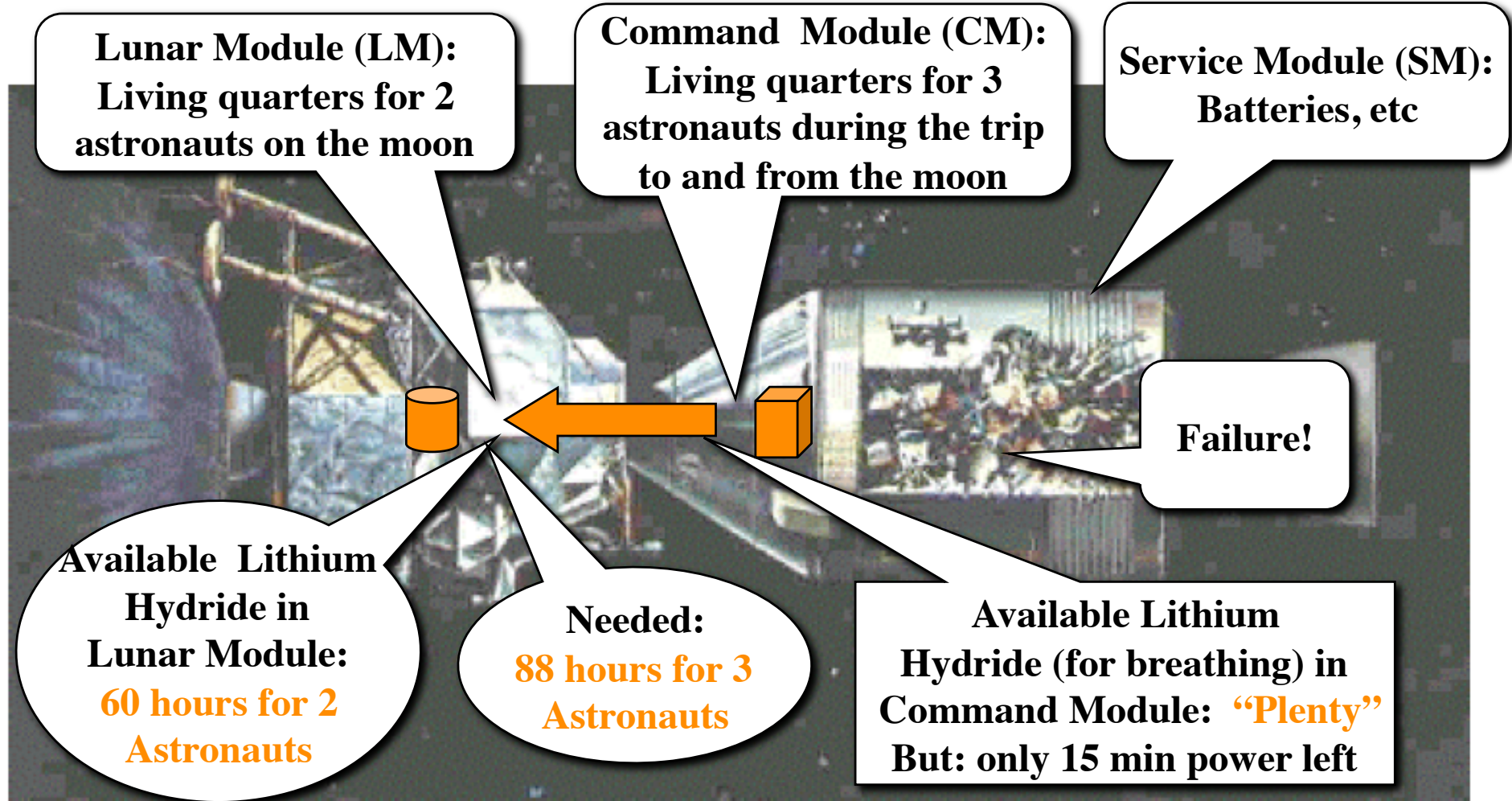
1. Identify the missing components
2. Make a build or buy decision to get the missing component.

# What do we do if we have non-Standard Components?



## Advanced Jigsaw Puzzles

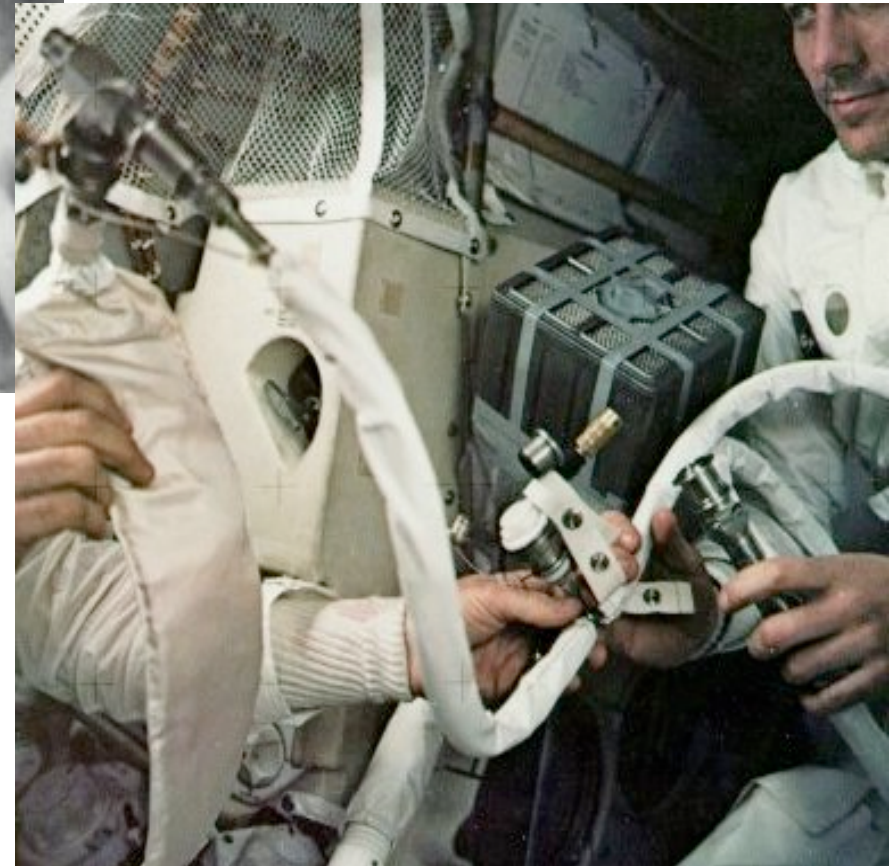
# Apollo 13: "Houston, we've had a Problem!"



The LM was **designed** for 60 hours for 2 astronauts staying 2 days on the moon  
**Redesign challenge:** Can the LM be used for 12 man-days (2 1/2 days until reentry into Earth)?

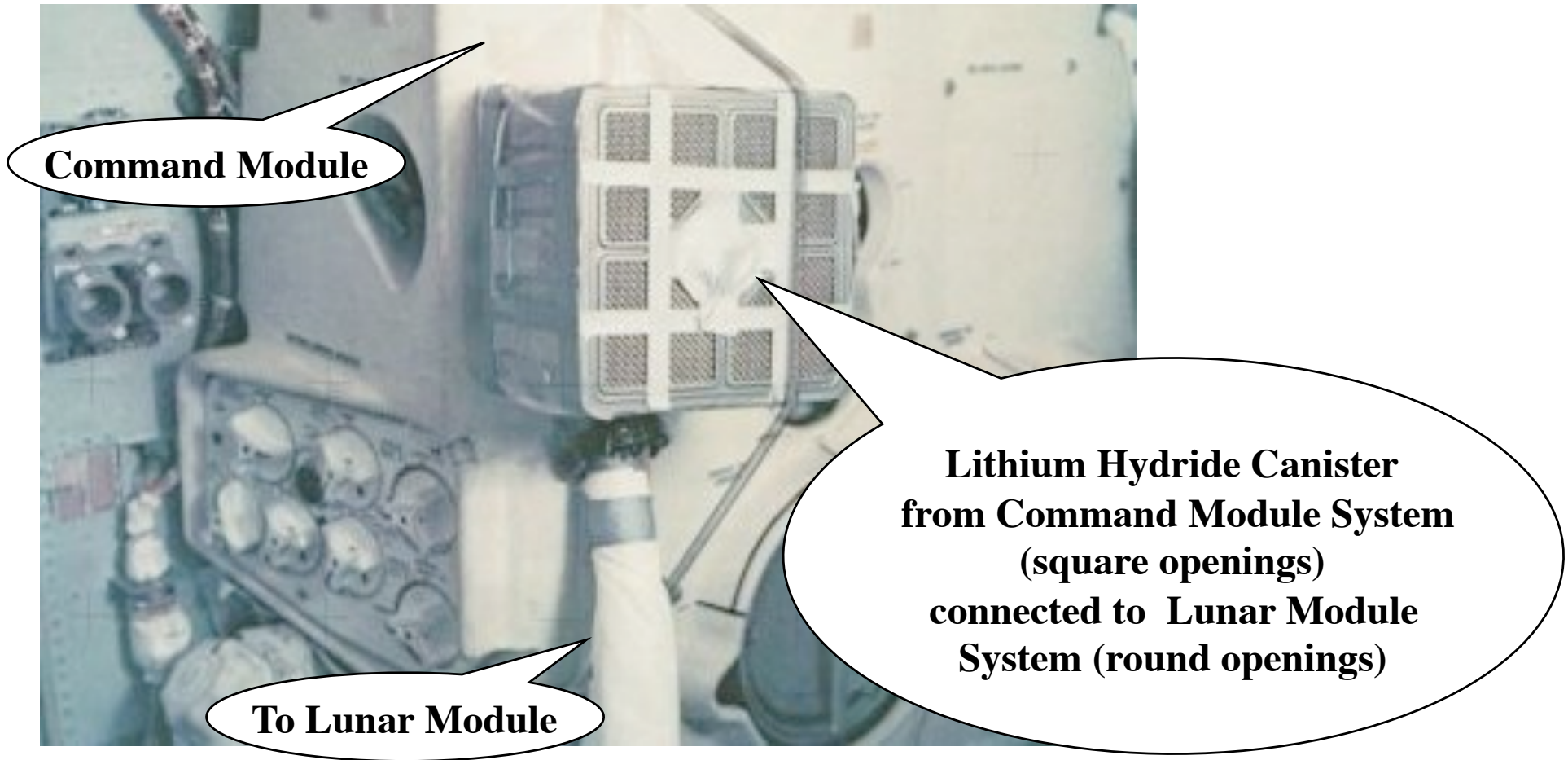
**Proposal:** Reuse Lithium Hydride Canisters from CM in LM  
**Problem:** Incompatible openings in Lithium Hydride Canisters

# Apollo 13: “Fitting a square peg in a round hole”





# A Typical Object Design Challenge: Connecting Incompatible Components

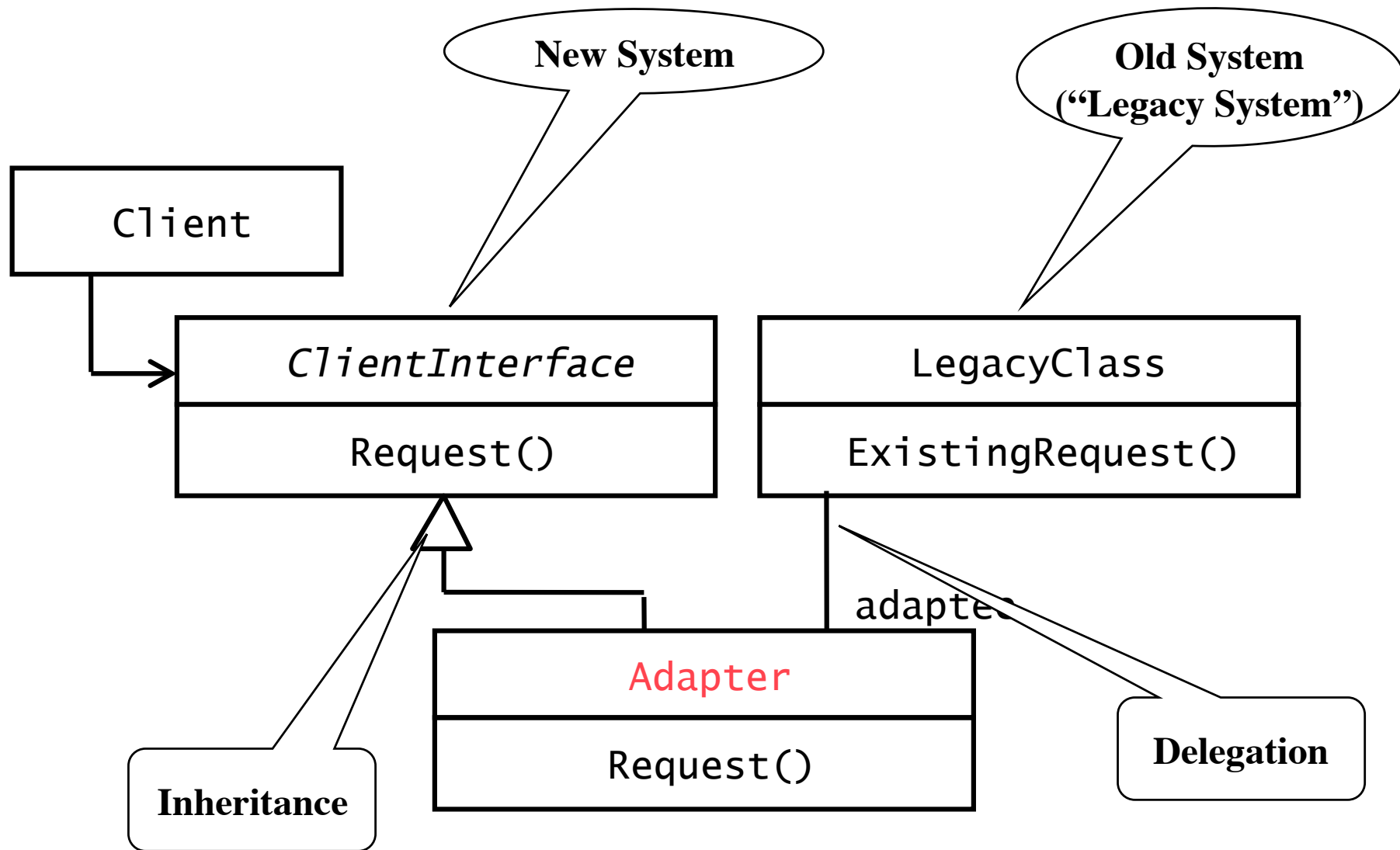


Source: <http://www.hq.nasa.gov/office/pao/History/SP-350/ch-13-4.html>

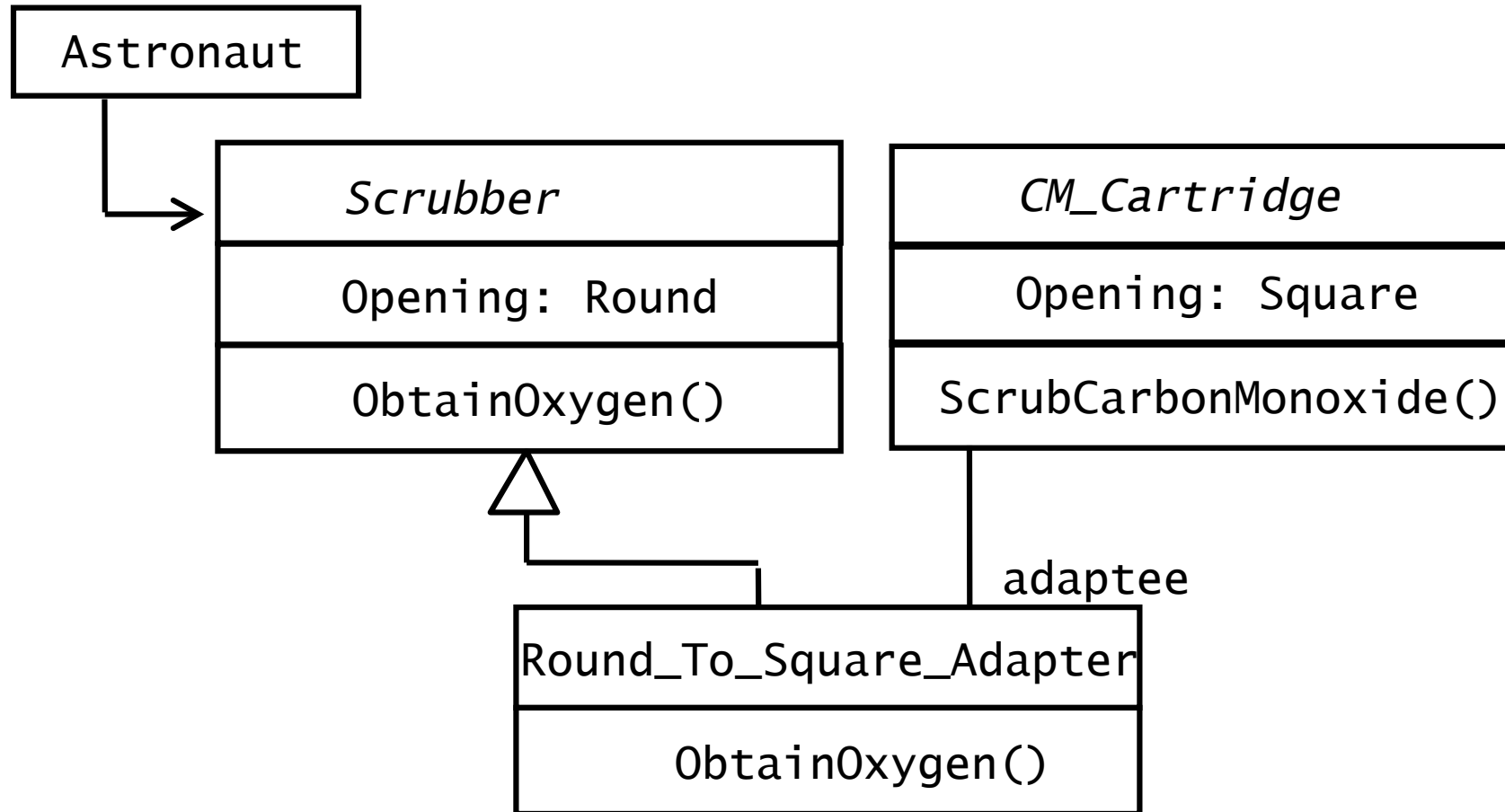
# Adapter Pattern

- **Adapter Pattern:** Converts the interface of a component into another interface expected by the calling component
- Used to provide a new interface to existing legacy components (Interface engineering, reengineering)
- Also known as a wrapper
- Two adapter patterns:
  - Class adapter:
    - Uses multiple inheritance to adapt one interface to another
  - Object adapter:
    - Uses single inheritance and delegation.

# Adapter Pattern



# Adapter for Scrubber in Lunar Module

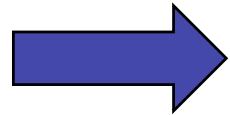


- Using a carbon monoxide scrubber (round opening) in the lunar module with square cartridges from the command module (square opening)

# Modeling of the Real World

- Modeling of the real world leads to a system that reflects today's realities but not necessarily tomorrow's.
- There is a need for *reusable* and flexible designs
- Design knowledge such as the adapter pattern complements application domain knowledge and solution domain knowledge.

# Typical of Object Design Activities



## Identifying possibilities of reuse

- Identification of existing components
- Full definition of associations
- Full definition of classes
  - System Design => Service, Object Design => API
- Specifying contracts for each component
  - OCL (Object Constraint Language)
- Choosing algorithms and data structures
- Detection of solution-domain classes
- Optimization
- Increase of inheritance
- Decision on control
- Packaging

# Reuse of Code

- I have a list, but my customer would like to have a stack
  - The list offers the operations Insert(), Find(), Delete()
  - The stack needs the operations Push(), Pop() and Top()
  - Can I reuse the existing list?
- I am an employee in a company that builds cars with expensive car stereo systems. Can I reuse the existing car software in a home stereo system?

# Reuse of interfaces

- I am an off-shore programmer in Hawaii. I have a contract to implement an electronic parts catalog for DaimlerChrysler
  - How can I and my contractor be sure that I implement it correctly?
- I would like to develop a window system for Linux that behaves the same way as in Windows
  - How can I make sure that I follow the conventions for Windows XP windows and not those of MacOS X?
- I have to develop a new service for cars, that automatically call a help center when the car is used the wrong way.
  - Can I reuse the help desk software that I developed for a company in the telecommunication industry?



# Reuse of existing classes

- I have an implementation for a list of elements vom Typ int
- How can I reuse this list without major effort to build a list of customers, or a spare parts catalog or a flight reservation schedule?
- Can I reuse a class "Addressbook", which I have developed in another project, as a subsystem in my commercially obtained proprietary e-mail program?
  - Can I reuse this class also in the billing software of my dealer management system?

# Reuse

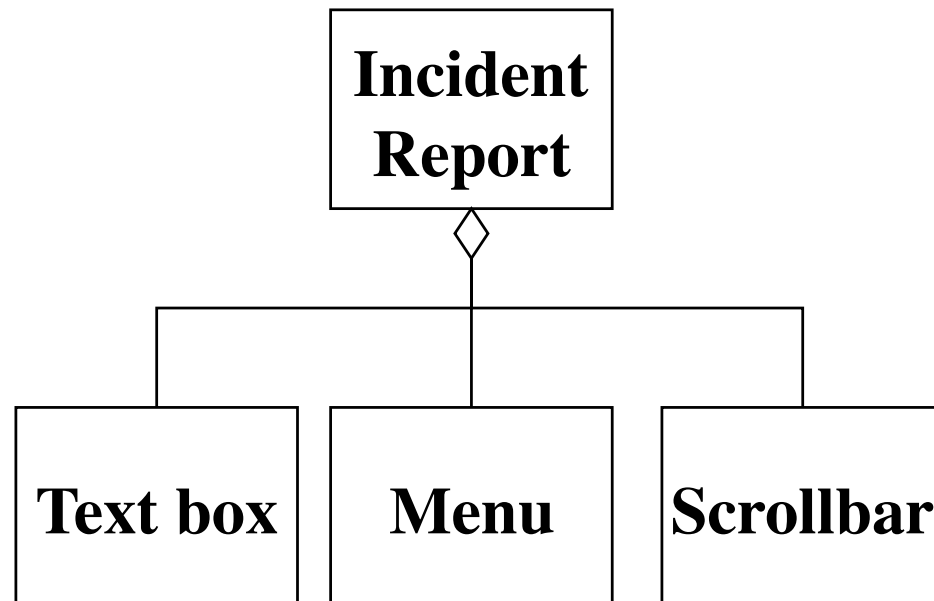
- Problem: Close the object design gap to develop new functionality
- Design goal:
  - Reuse knowledge from previous experience
  - Reuse functionality already available
- **Composition** (also called Black Box Reuse)
  - New functionality is obtained by aggregation
  - The new object with more functionality is an aggregation of existing objects
- **Inheritance** (also called White-box Reuse)
  - New functionality is obtained by inheritance
- In both cases: Identification of new classes

# Identification of new Classes during Object Design

Requirements Analysis  
(Language of Application Domain)



Object Design  
(Language of Solution Domain)



# Other Reasons for new Classes

- The implementation of algorithms may necessitate objects to hold values
- New low-level operations may be needed during the decomposition of high-level operations
- Example: `EraseArea()` in a drawing program
  - Conceptually very simple
  - Implementation is complicated:
    - Area represented by pixels
    - We need a `Repair()` operation to clean up objects partially covered by the erased area
    - We need a `Redraw()` operation to draw objects uncovered by the erasure
    - We need a `Draw()` operation to erase pixels in background color not covered by other objects.

# White Box and Black Box Reuse

- **White box reuse**
  - Access to the development products (models, system design, object design, source code) must be available
- **Black box reuse**
  - Access to models and designs is not available, or models do not exist
    - Worst case: Only executables (binary code) are available
    - Better case: A specification of the system interface is available.

# Types of Whitebox Reuse

## 1. Implementation inheritance

- Reuse of Implementations

## 2. Specification Inheritance

- Reuse of Interfaces

- Programming concepts to achieve reuse

- Inheritance

- Delegation
    - Abstract classes and Method Overriding
    - Interfaces

# Why Inheritance?

## 1. Organization (during analysis):

- Inheritance helps us with the construction of taxonomies to deal with the application domain
  - when talking the customer and application domain experts we usually find already existing taxonomies

## 2. Reuse (during object design):

- Inheritance helps us to reuse models and code to deal with the solution domain
  - when talking to developers

# The use of Inheritance

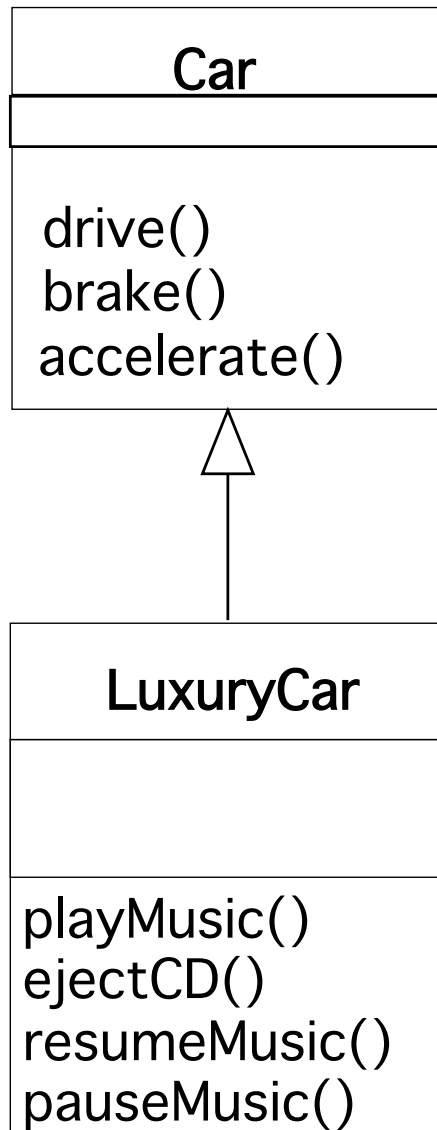
- Inheritance is used to achieve two different goals
  - Description of Taxonomies
  - Interface Specification
- **Description of taxonomies**
  - Used during *requirements analysis*
  - Activity: identify application domain objects that are hierarchically related
  - Goal: make the analysis model more understandable
- **Interface specification**
  - Used during *object design*
  - Activity: identify the signatures of all identified objects
  - Goal: increase reusability, enhance modifiability and extensibility



# Inheritance can be used during Modeling as well as during Implementation

- Starting Point is always the requirements analysis phase:
  - We start with use cases
  - We identify existing objects (“class identification”)
  - We investigate the relationship between these objects; “Identification of associations”:
    - general associations
    - aggregations
    - inheritance associations.

# Example of Inheritance in a Taxonomy



## Superclass:

```
public class Car {
    public void drive() {...}
    public void brake() {...}
    public void accelerate() {...}
}
```

## Subclass:

```
public class LuxuryCar extends Car
{
    public void playMusic() {...}
    public void ejectCD() {...}
    public void resumeMusic() {...}
    public void pauseMusic() {...}
}
```

# Inheritance comes in many Flavors 11 28 2008

The term Inheritance is used in four different ways:

- Specialization
- Generalization
- Specification Inheritance
- Implementation Inheritance.

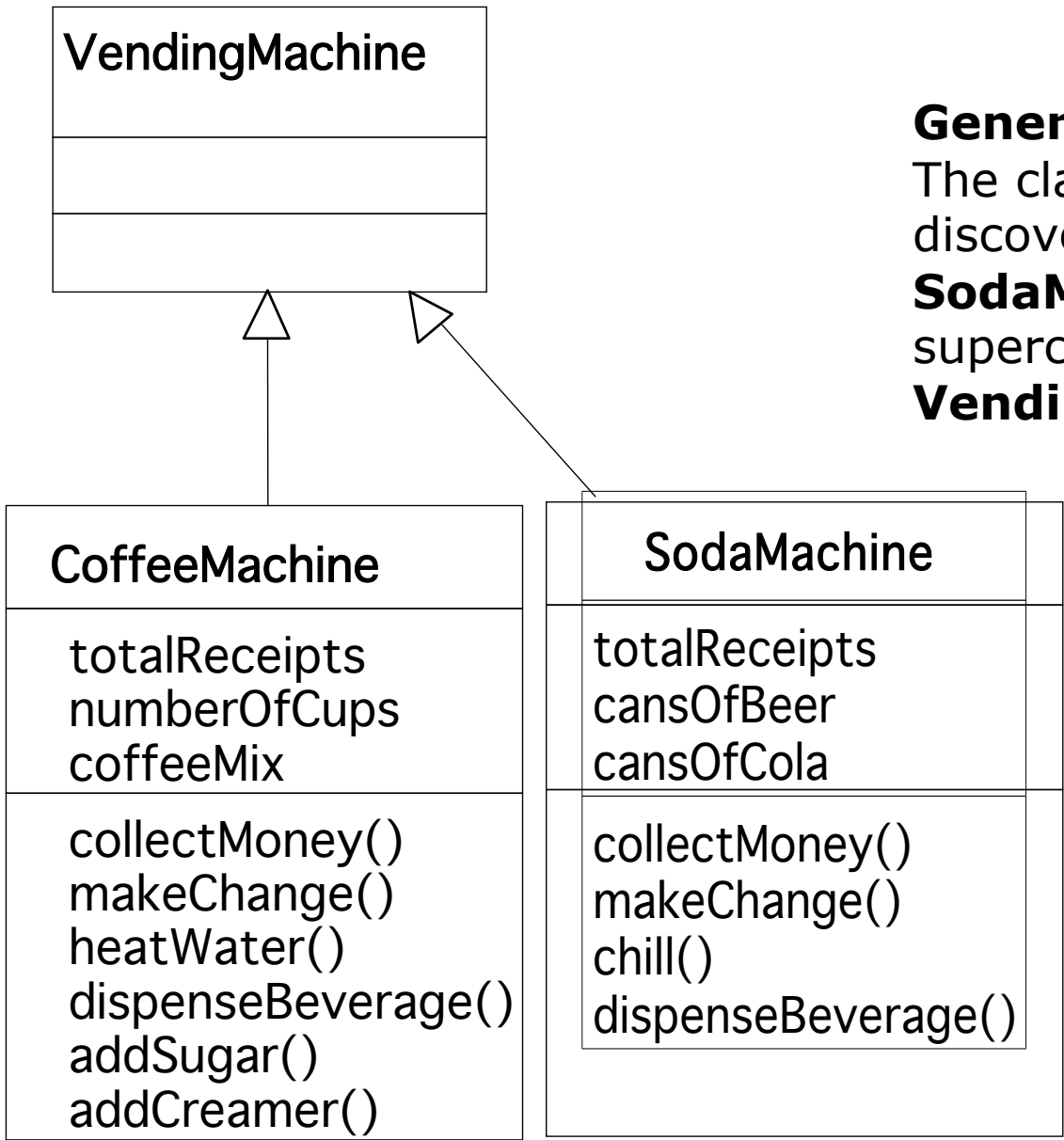
# Discovering Inheritance

- To “discover” inheritance associations, we can proceed in two ways, which we call specialization and generalization
- **Generalization**: the discovery of an inheritance relationship between two classes, where the sub class is discovered first.
- **Specialization**: the discovery of an inheritance relationship between two classes, where the super class is discovered first.

# Generalization

- First we find the subclass, then the super class
- This type of discovery occurs often in science

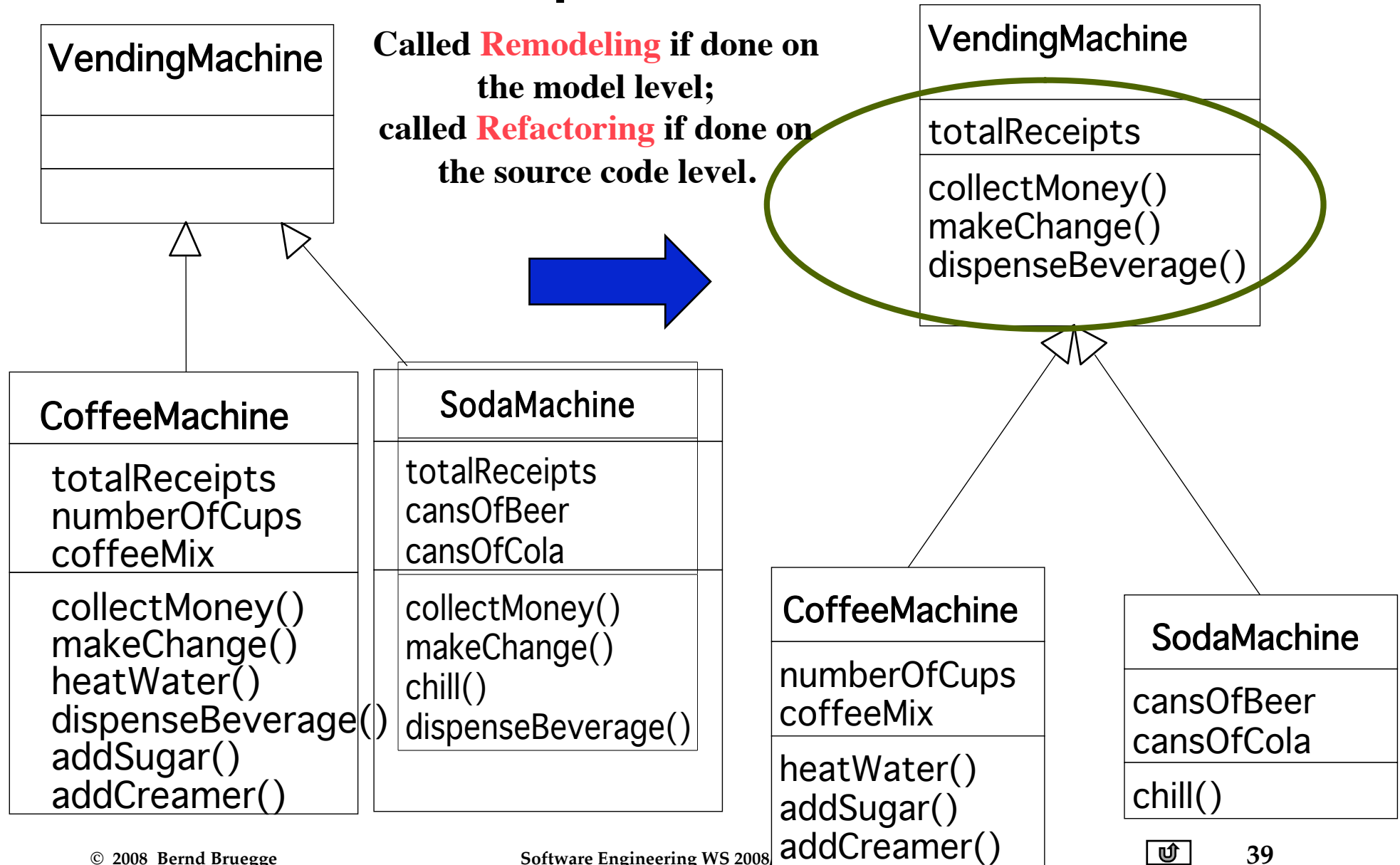
# Generalization Example: Modeling a Coffee Machine



## Generalization:

The class **CoffeeMachine** is discovered first, then the class **SodaMachine**, then the superclass **VendingMachine**

# Restructuring of Attributes and Operations is often a Consequence of Generalization



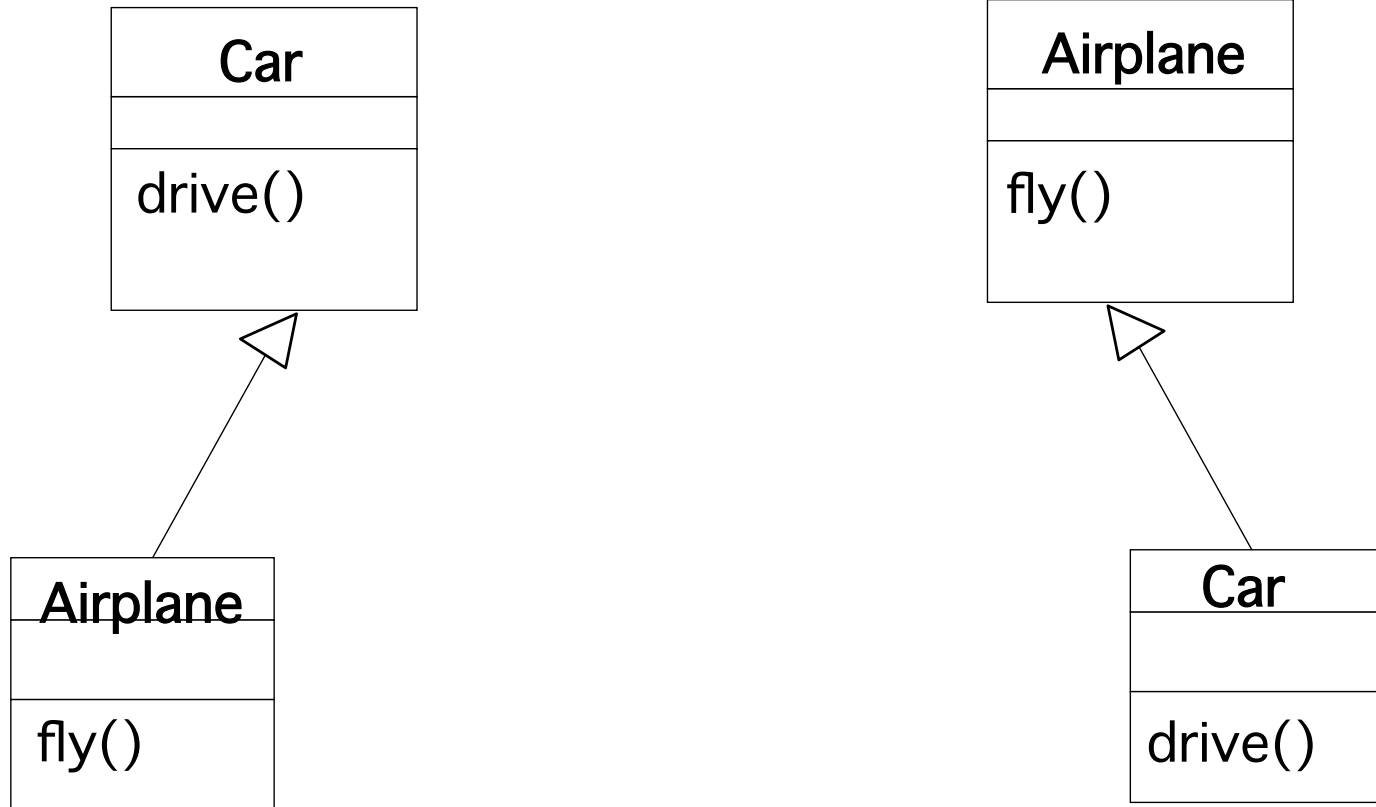
# Specialization

- Specialization occurs, when we find a subclass that is very similar to an existing class.
  - Example: A theory postulates certain particles and events which we have to find.
- Specialization can also occur unintentionally:





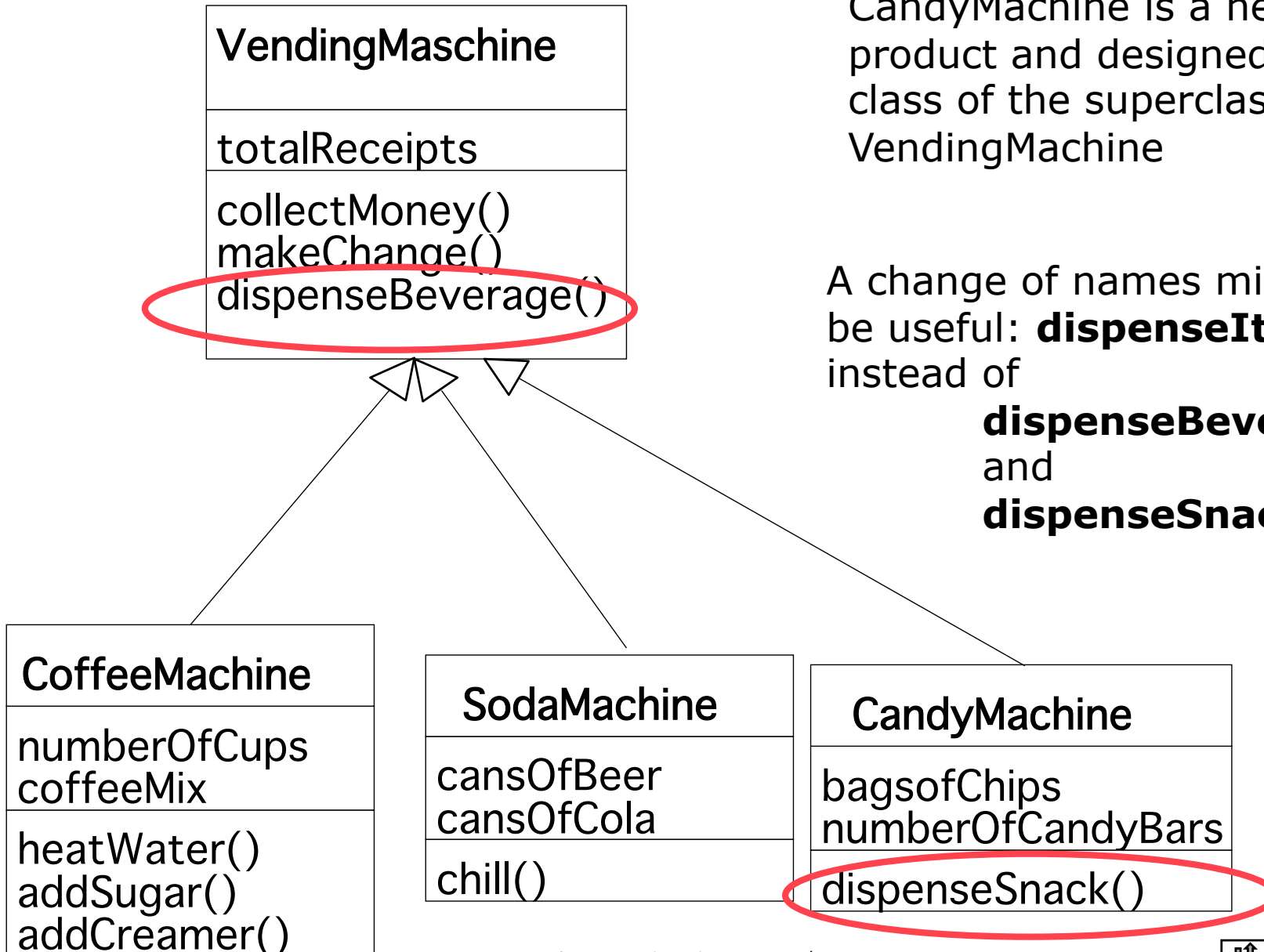
# Which Taxonomy models the scenario in the previous Slide?



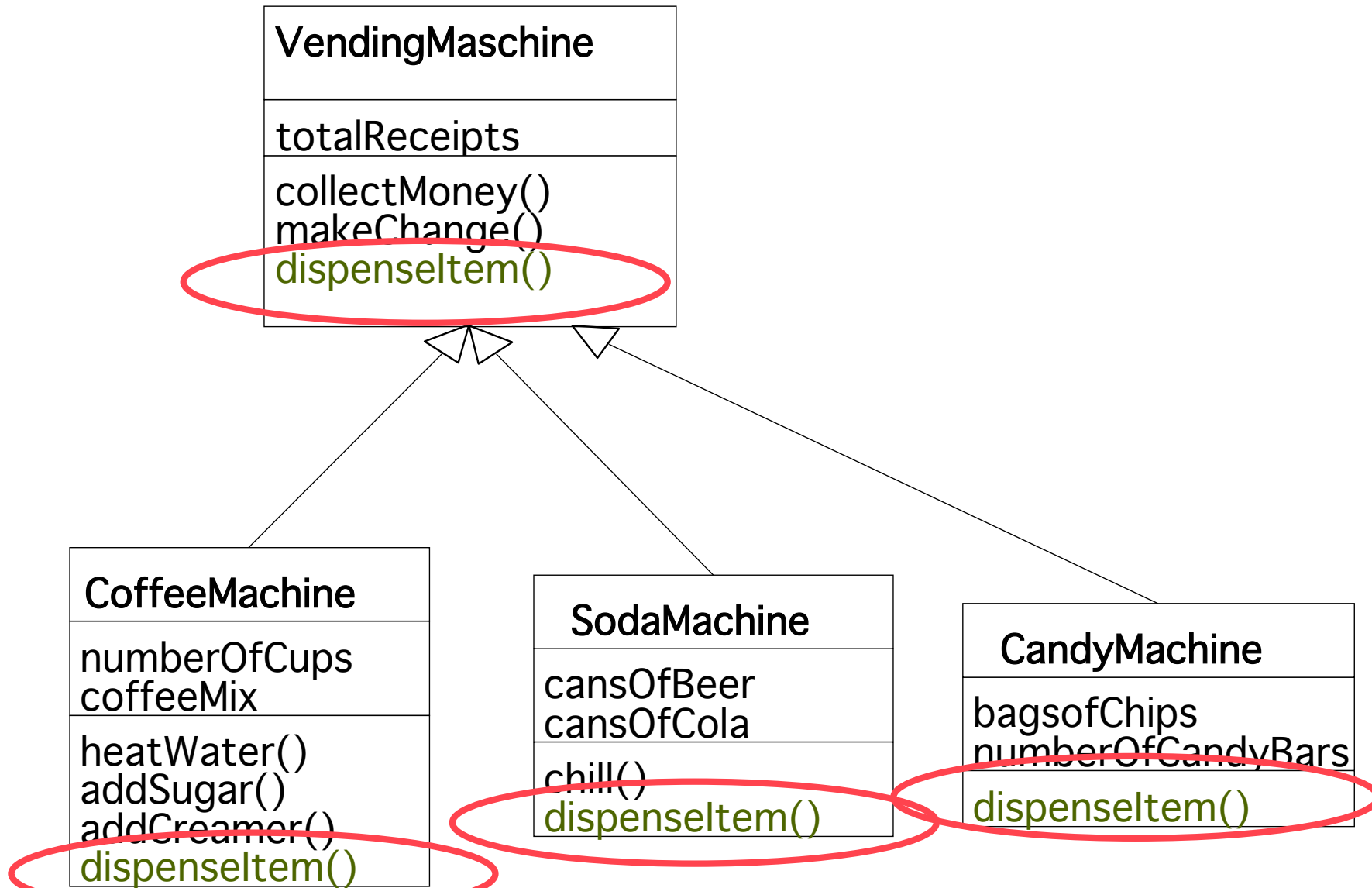
# Another Example of a Specialization

CandyMachine is a new product and designed as a subclass of the superclass VendingMachine

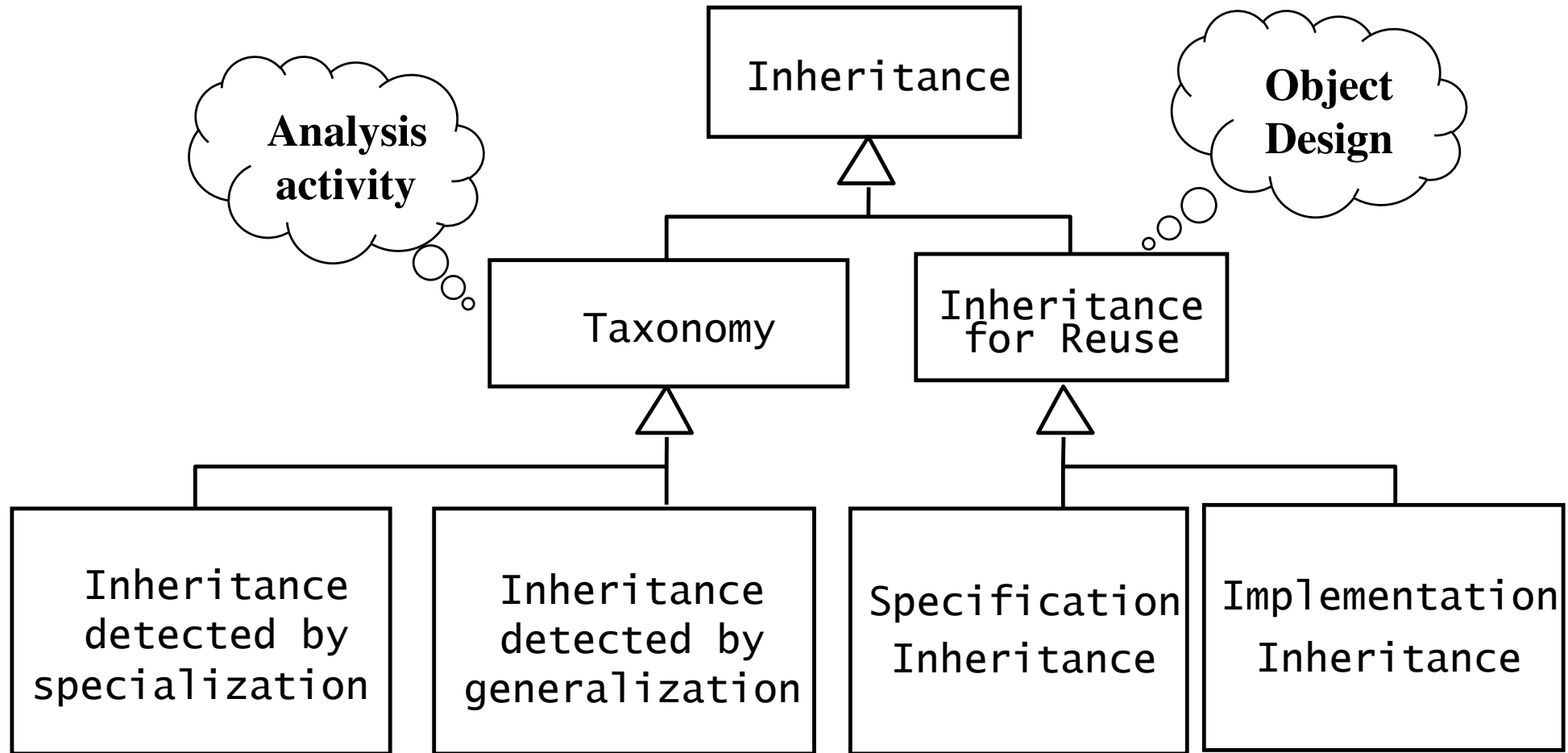
A change of names might now be useful: **dispenseItem()** instead of **dispenseBeverage()** and **dispenseSnack()**



# Example of a Specialization (2)



# Meta-Model for Inheritance



# Implementation Inheritance and Specification Inheritance

- **Implementation inheritance**
  - Also called class inheritance
  - Goal:
    - Extend an applications' functionality by reusing functionality from the super class
    - Inherit from an existing class with some or all operations already implemented
- **Specification Inheritance**
  - Also called subtyping
  - Goal:
    - Inherit from a specification
    - The specification is an abstract class with all operations specified, but not yet implemented.

# Implementation Inheritance v. Specification Inheritance

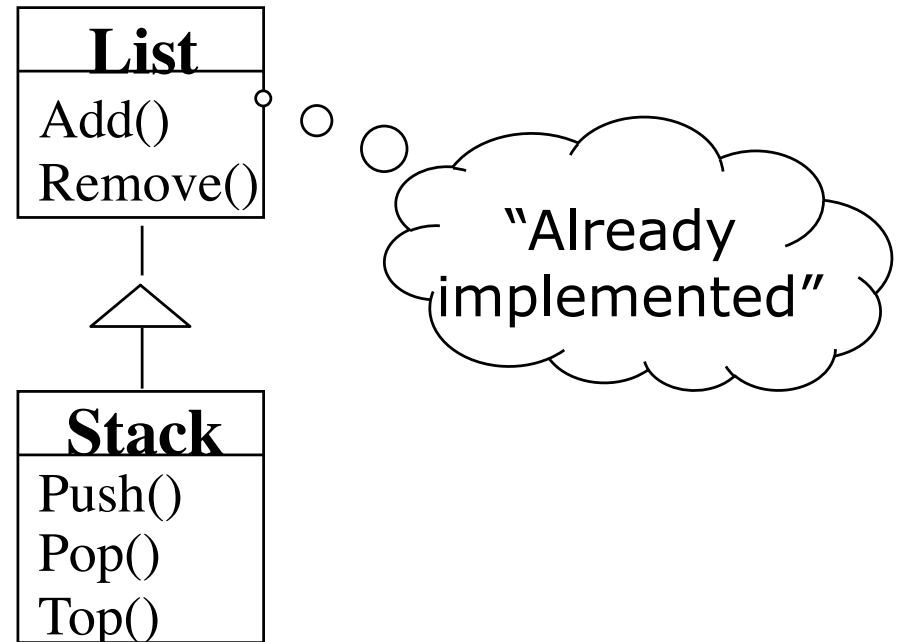
- **Implementation Inheritance:** The combination of inheritance and implementation
  - The Interface of the superclass is completely inherited
  - Implementations of methods in the superclass ("Reference implementations") are inherited by any subclass
- **Specification Inheritance:** The combination of inheritance and specification
  - The Interface of the superclass is completely inherited
  - Implementations of the superclass (if there are any) are not inherited.

# Example for Implementation Inheritance

- A very similar class is already implemented that does almost the same as the desired class implementation

Example:

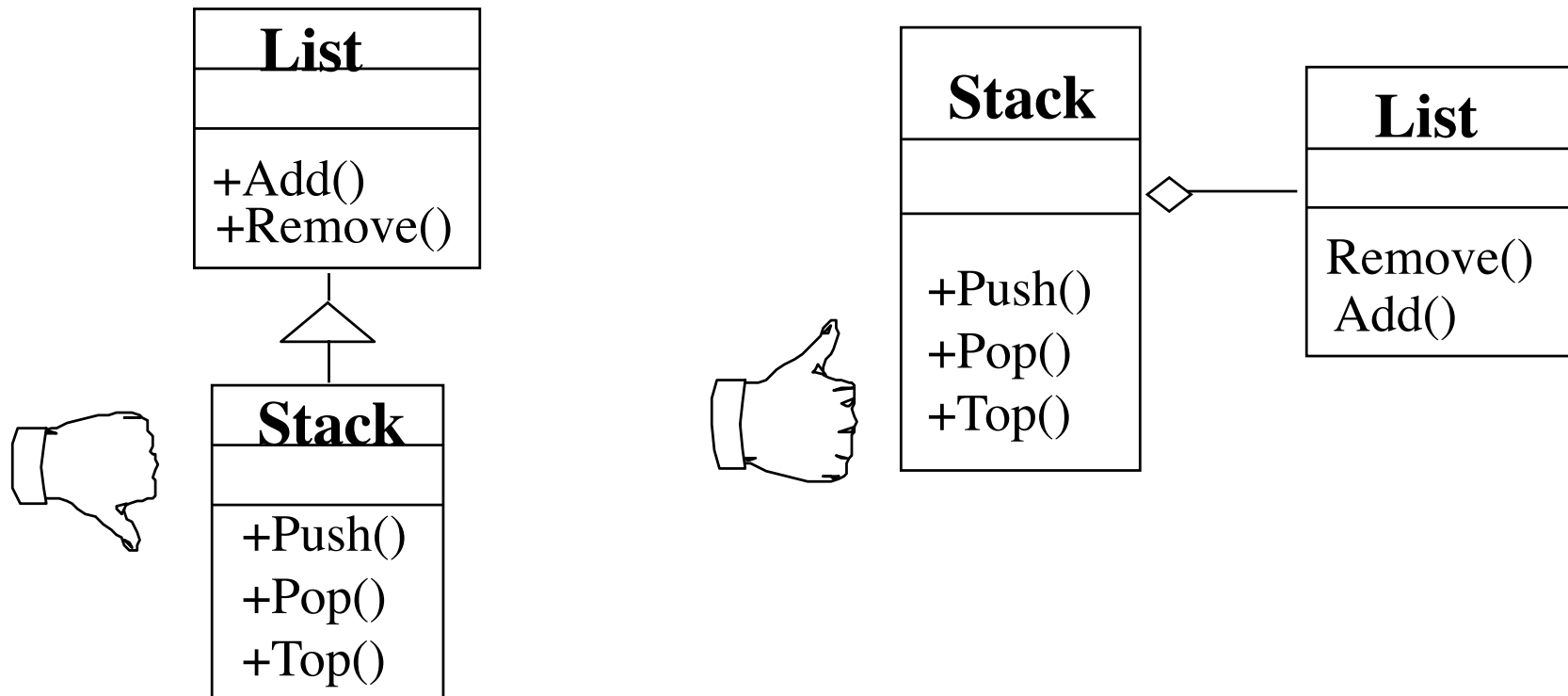
- I have a **List** class, I need a **Stack** class
- How about subclassing the **Stack** class from the **List** class and implementing **Push()**, **Pop()**, **Top()** with **Add()** and **Remove()**?



- ❖ Problem with implementation inheritance:
  - The inherited operations might exhibit unwanted behavior.
  - Example: What happens if the Stack user calls **Remove()** instead of **Pop()**?

# Better Code Reuse: Delegation

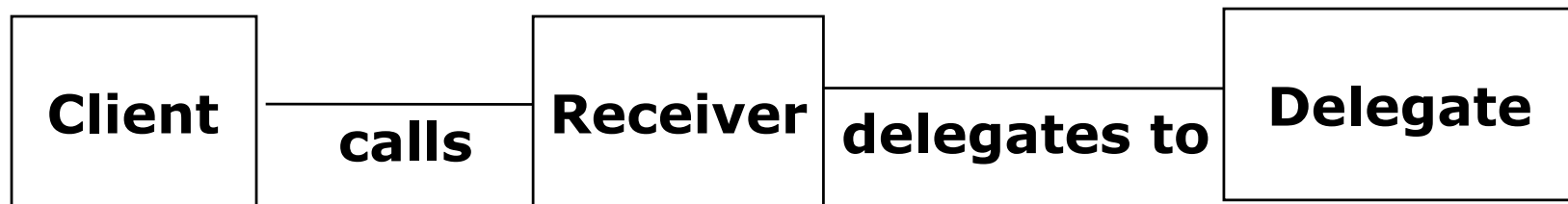
- **Implementation-Inheritance:** Using the implementation of super class operations
- **Delegation:** Catching an operation and sending it to another object that implements the operation





# Delegation

- Delegation is a way of making composition as powerful for reuse as inheritance
- In delegation two objects are involved in handling a request from a Client
  - The Receiver object delegates operations to the Delegate object
  - The Receiver object makes sure, that the Client does not misuse the Delegate object.



# Comparison: Delegation v. Inheritance

- Code-Reuse can be done by delegation as well as inheritance
- Delegation
  - Flexibility: Any object can be replaced at run time by another one
  - Inefficiency: Objects are encapsulated
- Inheritance
  - Straightforward to use
  - Supported by many programming languages
  - Easy to implement new functionality
  - Exposes a subclass to details of its super class
  - Change in the parent class requires recompilation of the subclass.

# Finally: Pack up the design

- Goal: Pack up design into discrete physical units that can be edited, compiled, linked, reused
- Two design principles for packaging
  - Minimize coupling:
    - Example: Classes in client-supplier architectures are usually loosely coupled
    - Large number of parameters ( $> 4-5$ ) in some methods mean high coupling
  - Maximize cohesion:
    - Classes closely connected by associations  $\Rightarrow$  same package

# Design Heuristics for Packaging

- Each subsystem service is made available by one or more interface objects within the package
- Start with one interface object for each subsystem service
  - Try to limit the number of interface operations (7+-2)
- If the service has too many operations, reconsider the number of interface objects
- If you have too many interface objects, reconsider the number of subsystems

# Summary

- Object design closes the gap between the requirements and the system design/machine.
- Object design adds details to the requirements analysis and prepares for implementation decisions
- Object design activities include:
  - Identification of Reuse
  - Identification of interface and implementation inheritance
  - Identification of opportunities for delegation