Chapter 9: Reuse and Patterns

(Textbook Chapter 8)

Course instructor : TA.Nada Alamoudi

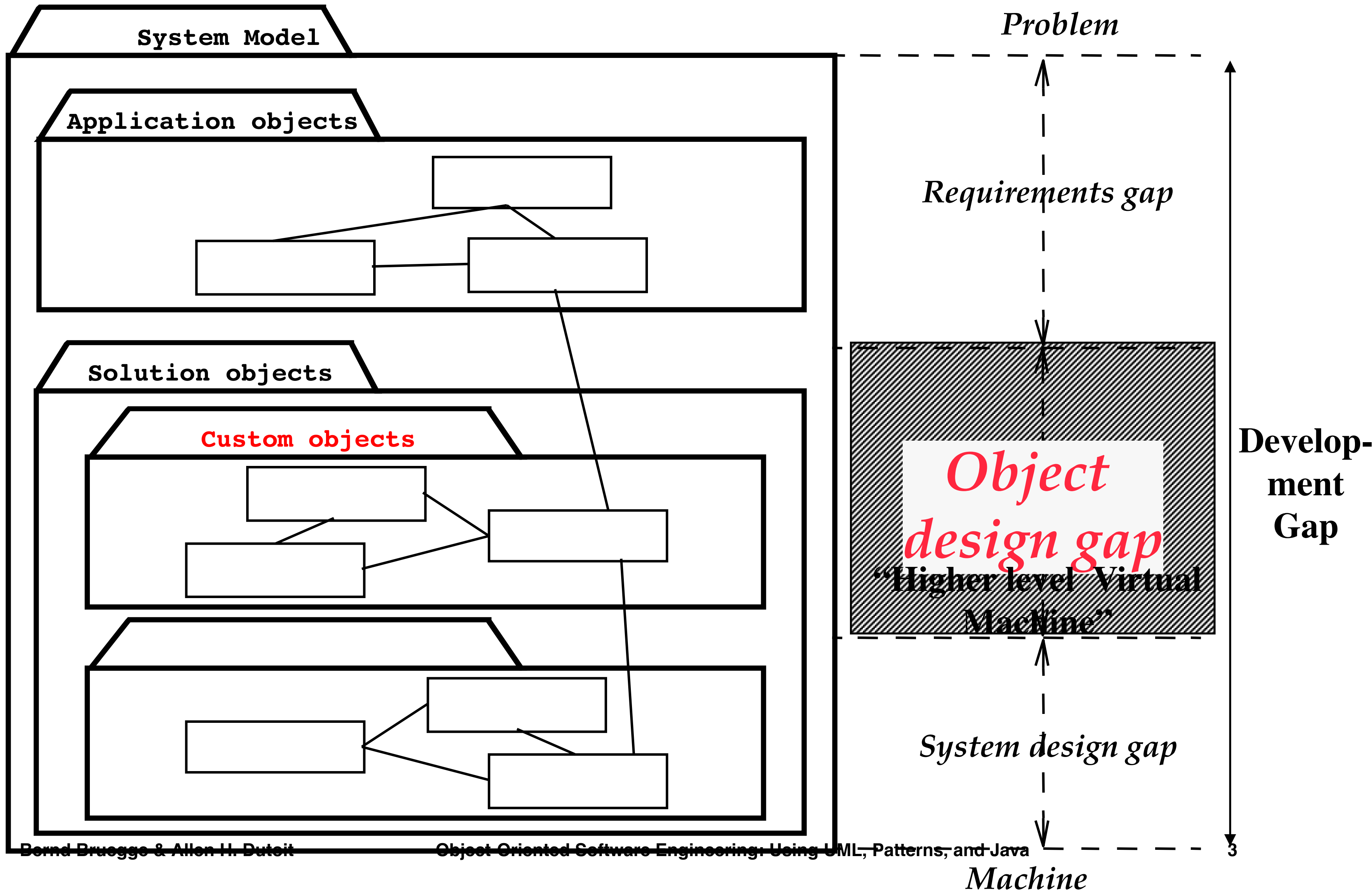
Object-Oriented Software Engineering

Using UML, Patterns, and Java

Object Design

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

Design means "Closing the Gap"



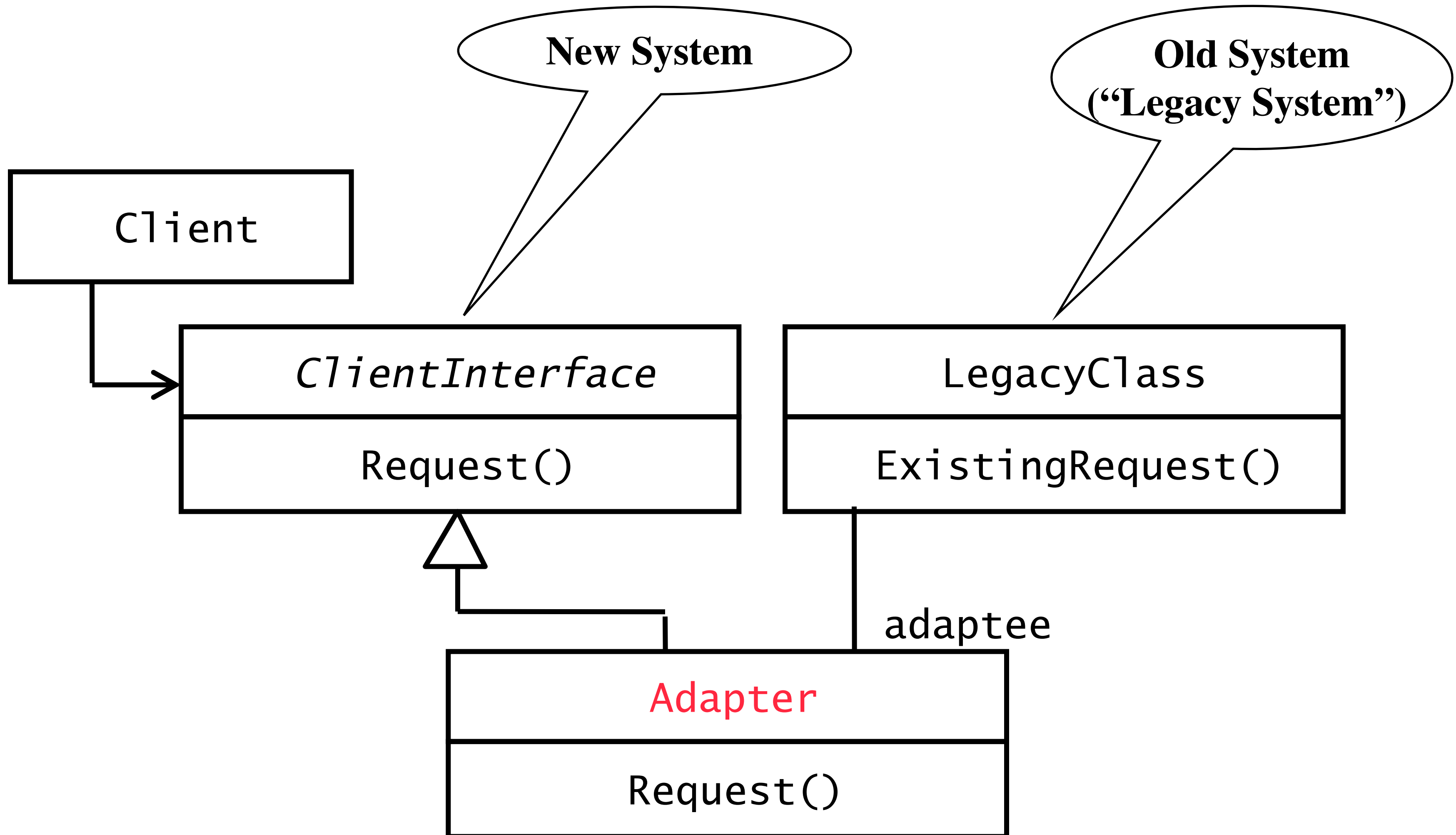
Object Design consists of 4 Activities

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.

Adapter Pattern

- Adapter Pattern: Connects incompatible components.
  - It converts the interface of one component into another interface expected by the other (calling) component
  - Used to provide a new interface to existing legacy components (Interface engineering, reengineering)
- Also known as a wrapper.

Adapter Pattern



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.

Customization: Build Custom Objects

- Problem: Close the object design gap
  - Develop new functionality
- Main 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

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.

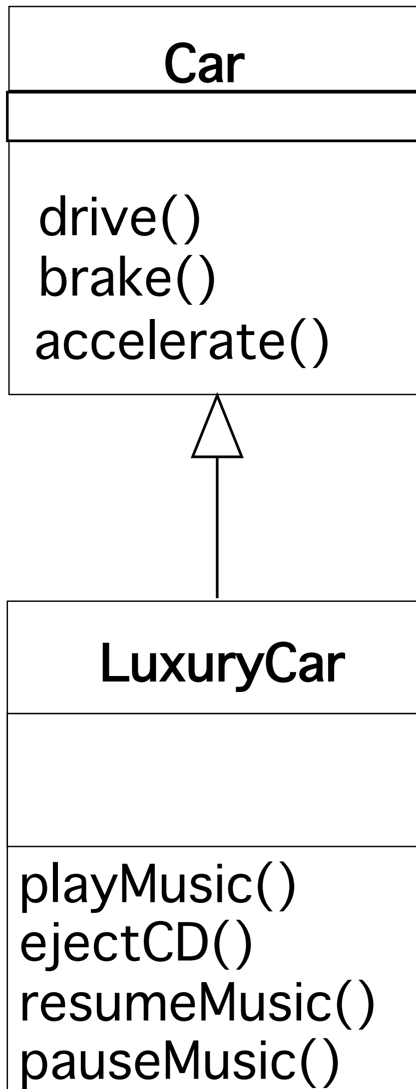
Why Inheritance?

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

Example of Inheritance



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

Inheritance is used in four ways:

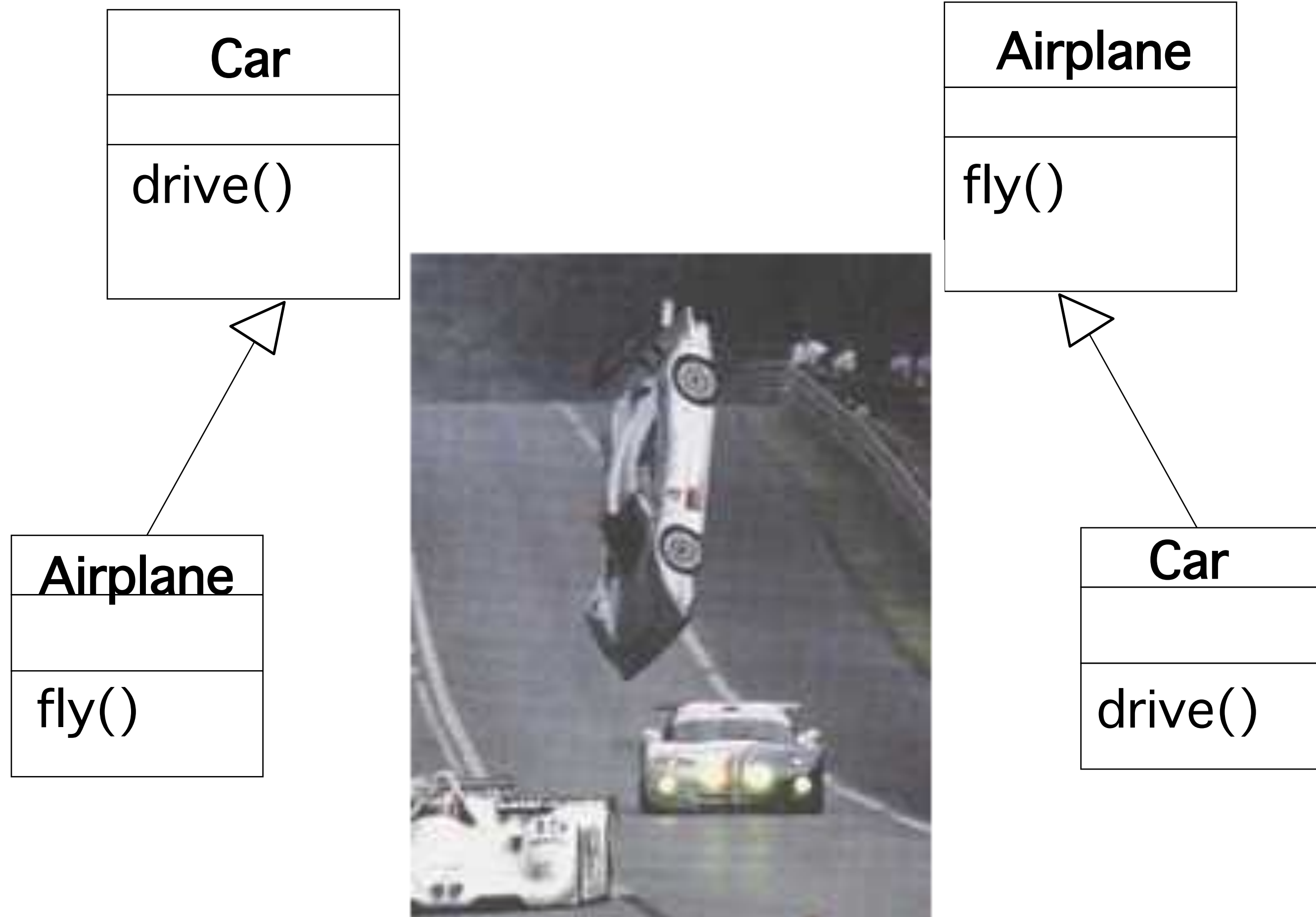
- Specialization
- Generalization
- Specification Inheritance
- Implementation Inheritance.

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.

Which Taxonomy is correct?



Implementation Inheritance and Specification Inheritance

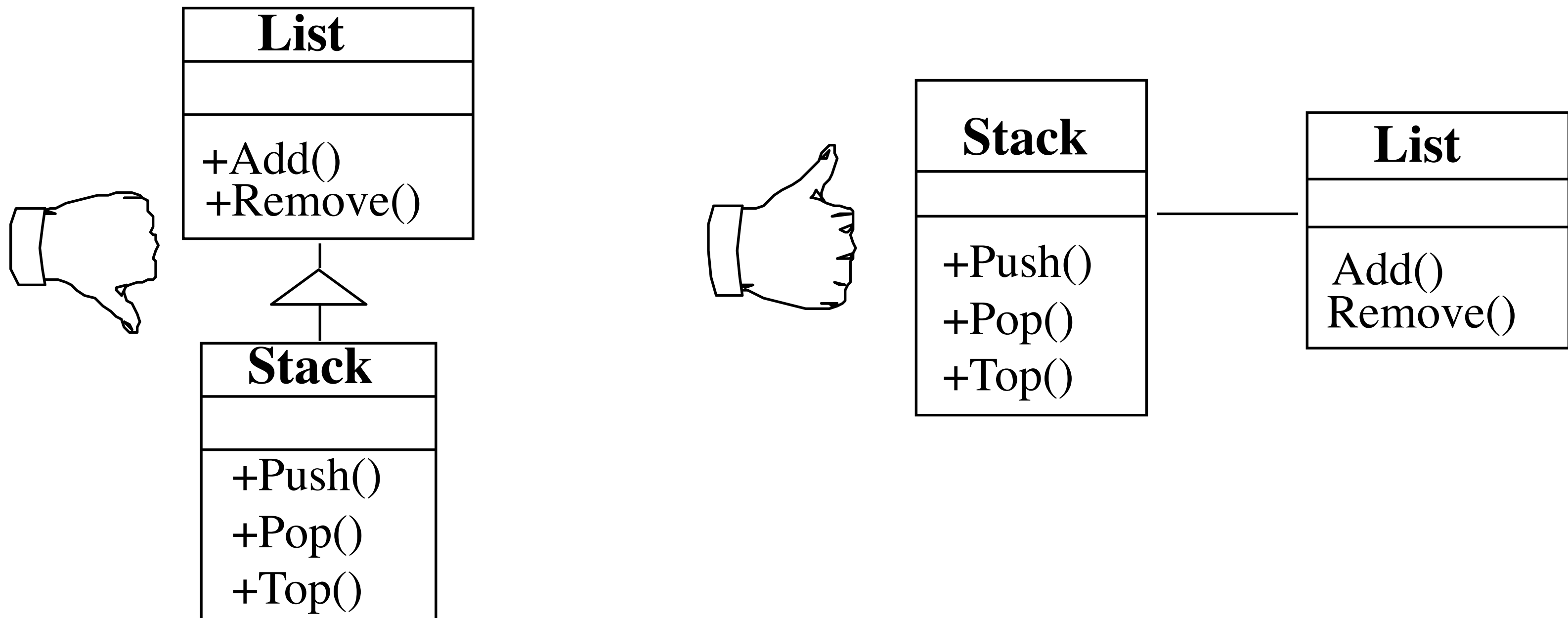
- Implementation inheritance
  - Also called class inheritance
  - Goal:
    - Extend an applications' functionality by reusing functionality from the super class
    - Inherit from an existing class with some or all operations already implemented
- Specification Inheritance
  - Also called subtyping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

Implementation Inheritance vs. 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.

Delegation instead of Implementation Inheritance

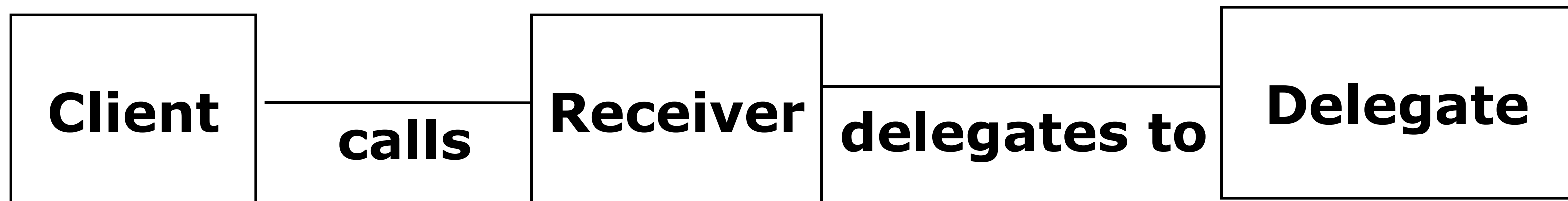
- Inheritance: Extending a Base class by a new operation or overwriting an operation.
- Delegation: Catching an operation and sending it to another object.
- Which of the following models is better?



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.



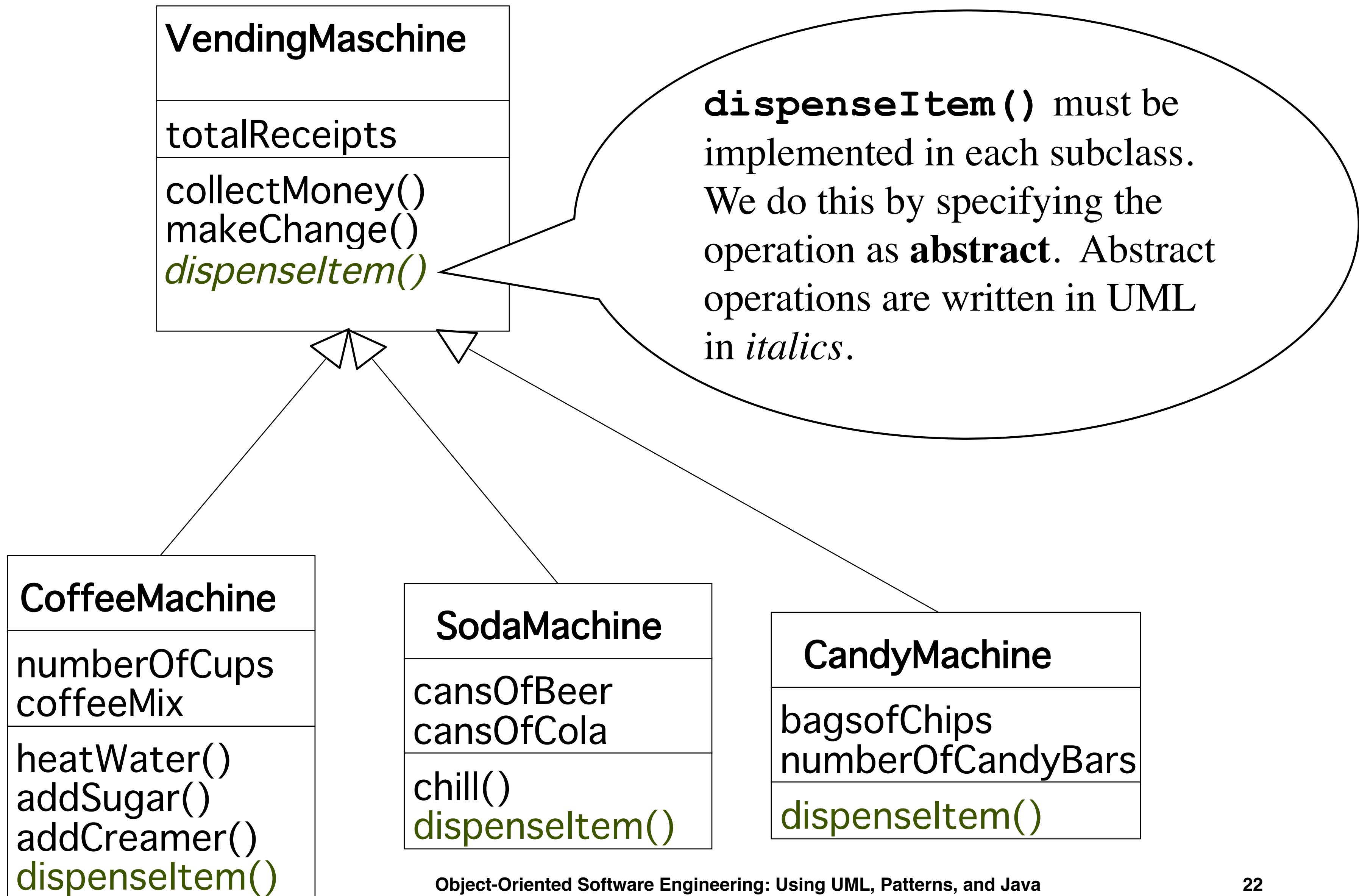
Comparison: Delegation vs Implementation Inheritance

- Delegation
  - ☺ Flexibility: Any object can be replaced at run time by another one (as long as it has the same type)
  - ☹ Inefficiency: Objects are encapsulated.
- Inheritance
  - ☺ Straightforward to use
  - ☺ Supported by many programming languages
  - ☺ Easy to implement new functionality
  - ☹ Inheritance exposes a subclass to the details of its parent class
  - ☹ Any change in the parent class implementation forces the subclass to change (which requires recompilation of both)

Abstract Methods and Abstract Classes

- Abstract method:
  - A method with a signature but without an implementation (also called abstract operation)
- Abstract class:
  - A class which contains at least one abstract method is called abstract class
- Interface: An abstract class which has only abstract methods
  - An interface is primarily used for the specification of a system or subsystem. The implementation is provided by a subclass or by other mechanisms.

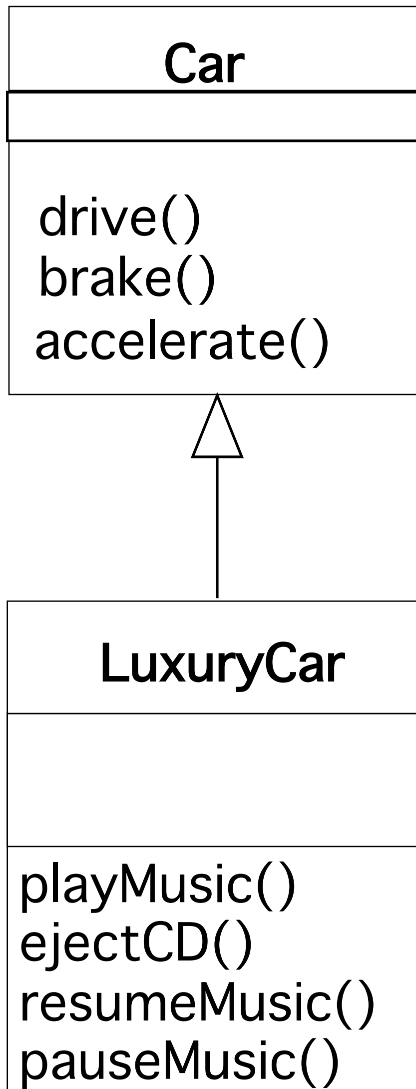
Example of an Abstract Method



Rewriteable Methods and Strict Inheritance

- Rewriteable Method: A method which allow a reimplementation.
  - In Java methods are rewriteable by default, i.e. there is no special keyword.
- Strict inheritance
    - The subclass can only add new methods to the superclass, it cannot over write them
  - If a method cannot be overwritten in a Java program, it must be prefixed with the keyword `final`.

Strict Inheritance



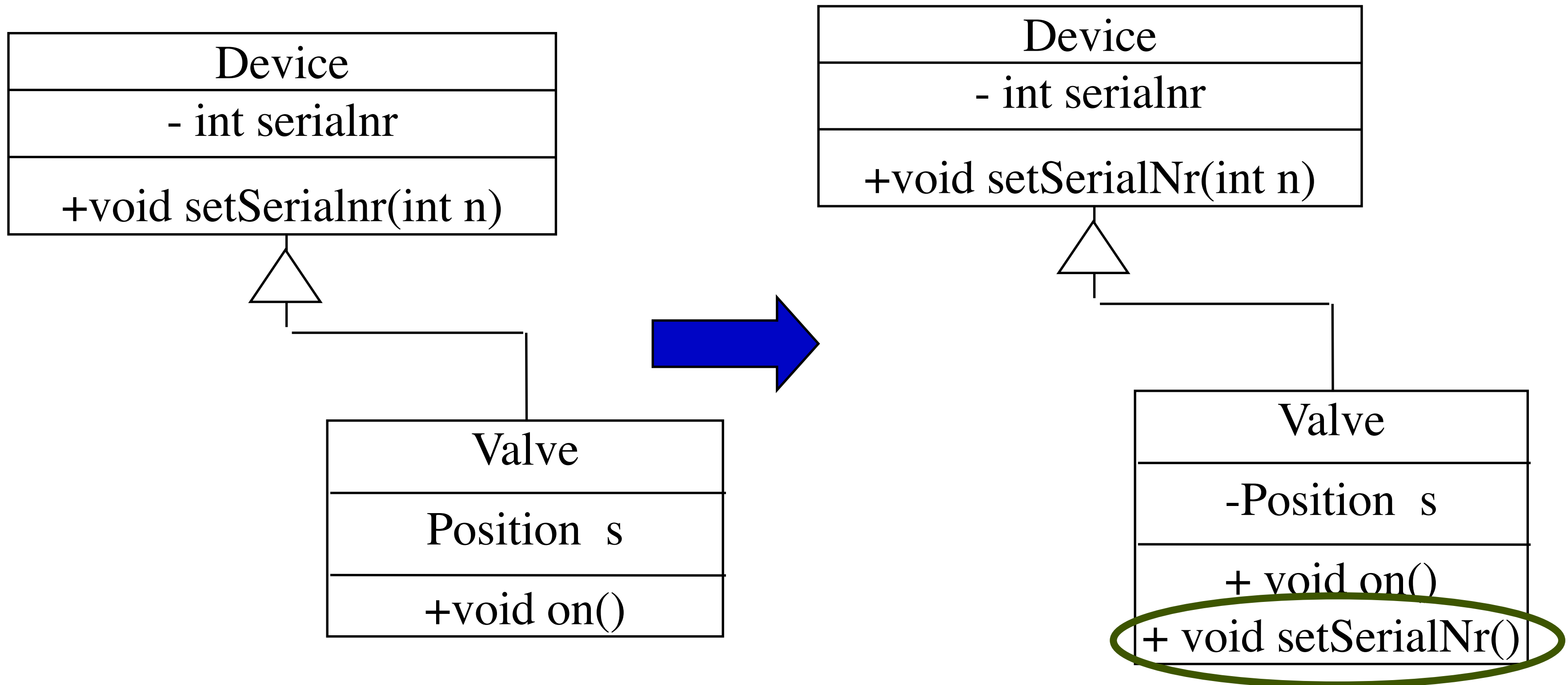
Superclass:

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

Subclass:

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

UML Class Diagram



Contraction

- **Contraction:** Implementations of methods in the super class are overwritten with empty bodies in the subclass to make the super class operations “invisible“
- Contraction is a special type of inheritance
- It should be avoided at all costs, but is is used often.

Contraction must be avoided by all Means

A contracted subclass delivers the desired functionality expected by the client, but:

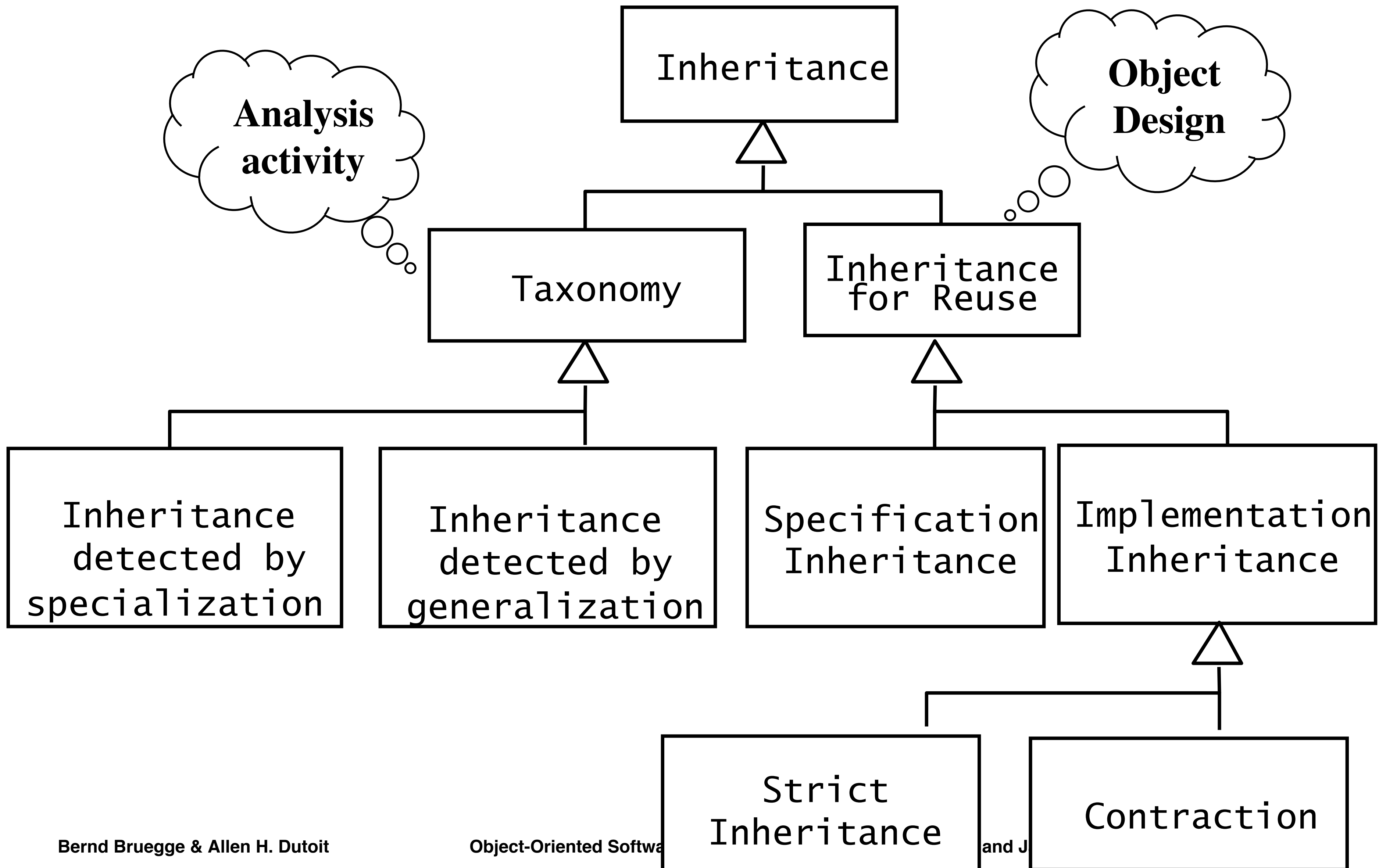
- The interface contains operations that make no sense for this class
- What is the meaning of the operation brake() for a BoomBox?

The subclass does not fit into the taxonomy

A BoomBox ist not a special form of Auto

- The subclass violates Liskov's Substitution Principle:
  - I cannot replace Auto with BoomBox to drive to work.

Revised Metamodel for Inheritance



Frameworks

- A framework is a reusable partial application that can be specialized to produce custom applications.
- The key benefits of frameworks are reusability and extensibility:
  - Reusability leverages of the application domain knowledge and prior effort of experienced developers
  - Extensibility is provided by hook methods, which are overwritten by the application to extend the framework.

Classification of Frameworks

- Frameworks can be classified by their position in the software development process:
  - Infrastructure frameworks
  - Middleware frameworks

- Frameworks can also be classified by the techniques used to extend them:
  - Whitebox frameworks
  - Blackbox frameworks

Frameworks in the Development Process

- Infrastructure frameworks aim to simplify the software development process
  - Used internally, usually not delivered to a client.
- Middleware frameworks are used to integrate existing distributed applications
  - Examples: MFC, DCOM, Java RMI, WebObjects, WebSphere, WebLogic Enterprise Application [BEA].
- Enterprise application frameworks are application specific and focus on domains
  - Example of application domains: telecommunications, avionics, environmental modeling, manufacturing, financial engineering, enterprise business activities.

White-box and Black-box Frameworks

- White-box frameworks:
  - Extensibility achieved through *inheritance* and dynamic binding.
  - Existing functionality is extended by subclassing framework base classes and overriding specific methods (so-called hook methods)
- Black-box frameworks:
  - Extensibility achieved by defining interfaces for components that can be plugged into the framework.
  - Existing functionality is reused by defining components that conform to a particular interface
  - These components are integrated with the framework via *delegation*.

Class libraries vs. Frameworks

- Class Library:
  - Provide a smaller scope of reuse
  - Less domain specific
  - Class libraries are passive; no constraint on the flow of control
- Framework:
  - Classes cooperate for a family of related applications.
  - Frameworks are active; they affect the flow of control.

Components vs. Frameworks

- Components:
  - Self-contained instances of classes
  - Plugged together to form complete applications
  - Can even be reused on the binary code level
    - The advantage is that applications do not have to be recompiled when components change
- Framework:
  - Often used to develop components
  - Components are often plugged into blackbox frameworks.

Documenting Object Design: ODD Conventions

- Each subsystem in a system provides a service
  - Describes the set of operations provided by the subsystem
- Specification of the service operations
  - Signature: Name of operation, fully typed parameter list and return type
  - Abstract: Describes the operation
  - Pre: Precondition for calling the operation
  - Post: Postcondition describing important state after the execution of the operation
- Use JavaDoc and Contracts for the specification of service operations
  - Contracts are covered in the next lecture.

Summary

- Object design closes the gap between the requirements and the machine
- Object design adds details to the requirements analysis and makes implementation decisions
- Object design activities include:
  - ✓ Identification of Reuse
  - ✓ Identification of Inheritance and Delegation opportunities
  - ✓ Component selection
  - Interface specification (Next lecture)
  - Object model restructuring
  - Object model optimization

  } Lectures on Mapping Models to Code
- Object design is documented in the Object Design Document (ODD).

Reuse

- Main goal:
  - Reuse knowledge from previous experience to current problem
  - 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 components
- Inheritance (also called White-box Reuse)
  - New functionality is obtained by inheritance.
- Three ways to get new functionality:
  - Implementation inheritance
  - Interface inheritance
  - Delegation