Session 2 – The Object-Oriented Paradigm

Dao Tran Hoang Chau

Modified from Introduction to Object-Oriented Analysis and Design with UML and Unified Process, Stephen R. Schach

Chapter Overview

- The Traditional Paradigm versus the Object-Oriented Paradigm
- Objects and Classes
- Inheritance
- Generalization, Aggregation, and Association
- Examples of UML Class Modeling
- Information Hiding
- The Unified Process
- Iteration and Incrementation within the Unified Process

Operations and Data

- All programming languages perform *operations* on *data*

- Example 1: Banking information system
  - Data: accountBalance
  - Operations: deposit
    withdraw

Traditional versus Object-Oriented Paradigm

- In the past, systems analysis was a creative skill
  - Individuality was highly prized
- Around 1970, large computers became affordable
  - Information systems began to be developed by teams
  - A systematic approach was needed
- The *traditional methodology* or *traditional paradigm* was developed
  - Also known as the *structured methodology* or *structured paradigm*

Traditional v. Object-Oriented Paradigm (contd)

- Initially, the traditional paradigm was successful
  - Any approach is better than no approach at all
- During the 1980s
  - Larger computers became affordable
  - Information systems grew in size
- But the larger the information system, the less successful the traditional paradigm was
- What had gone wrong?

Traditional v. Object-Oriented Paradigm (contd)

- The traditional paradigm worked well with *small-scale* information systems
  - About 5,000 lines of code
- But not with *medium-scale* information systems
  - About 50,000 lines of code
- Or with *large-scale* information systems
  - At least 500,000 lines of code
- Why?

Traditional v. Object-Oriented Paradigm (contd)

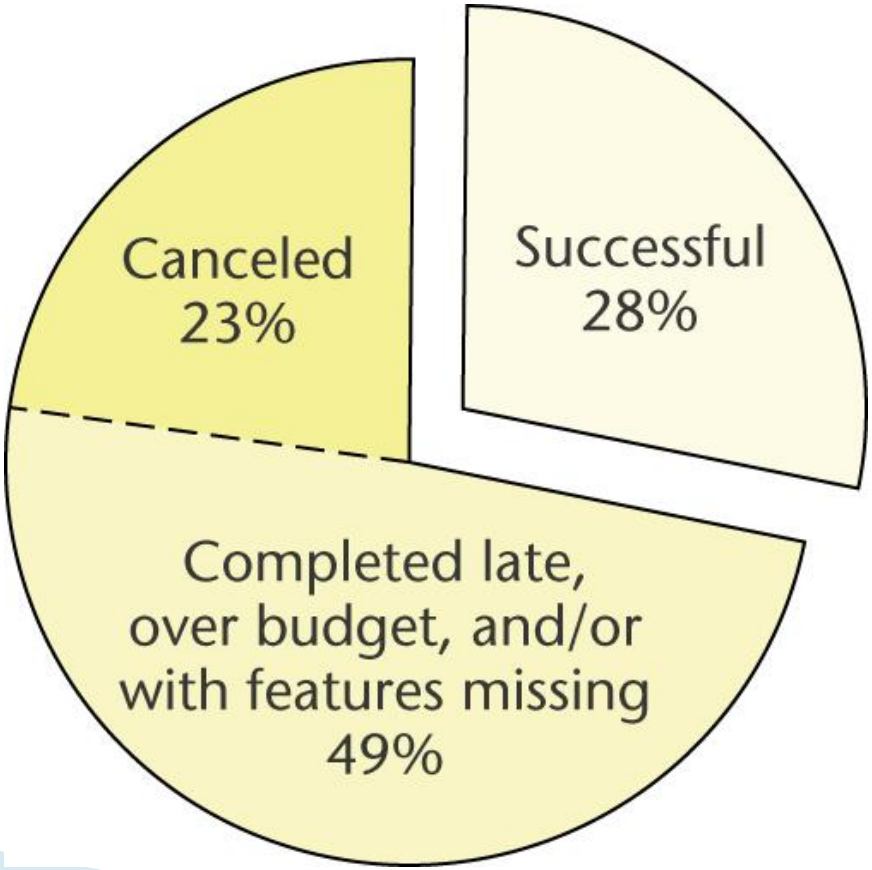
- Recall: The two basic building blocks of information systems are
  - Operations; and
  - Data
- The traditional paradigm
  - Concentrates on operations; and
  - Largely ignores data
- The *object-oriented* paradigm gives equal weight to operations and data

The Current Situation

- Just how serious is the current situation?
- Do we really need to adopt the object-oriented paradigm?

The Current Situation (contd)

- Study of 280,000 projects completed in 2000
  - Only about 1 in 4 was a success



Legal Implications of the Current Situation

- 2002 survey of information technology organizations
  - 78% have been involved in disputes that ended in litigation
- For the organizations that entered into litigation:
  - In 67% of the disputes, the functionality of the information system as delivered did not meet not up to the claims of the developers
  - In 56% of the disputes, the promised delivery date slipped several times
  - In 45% of the disputes, the defects were so severe that the information system was unusable

Objects and Classes

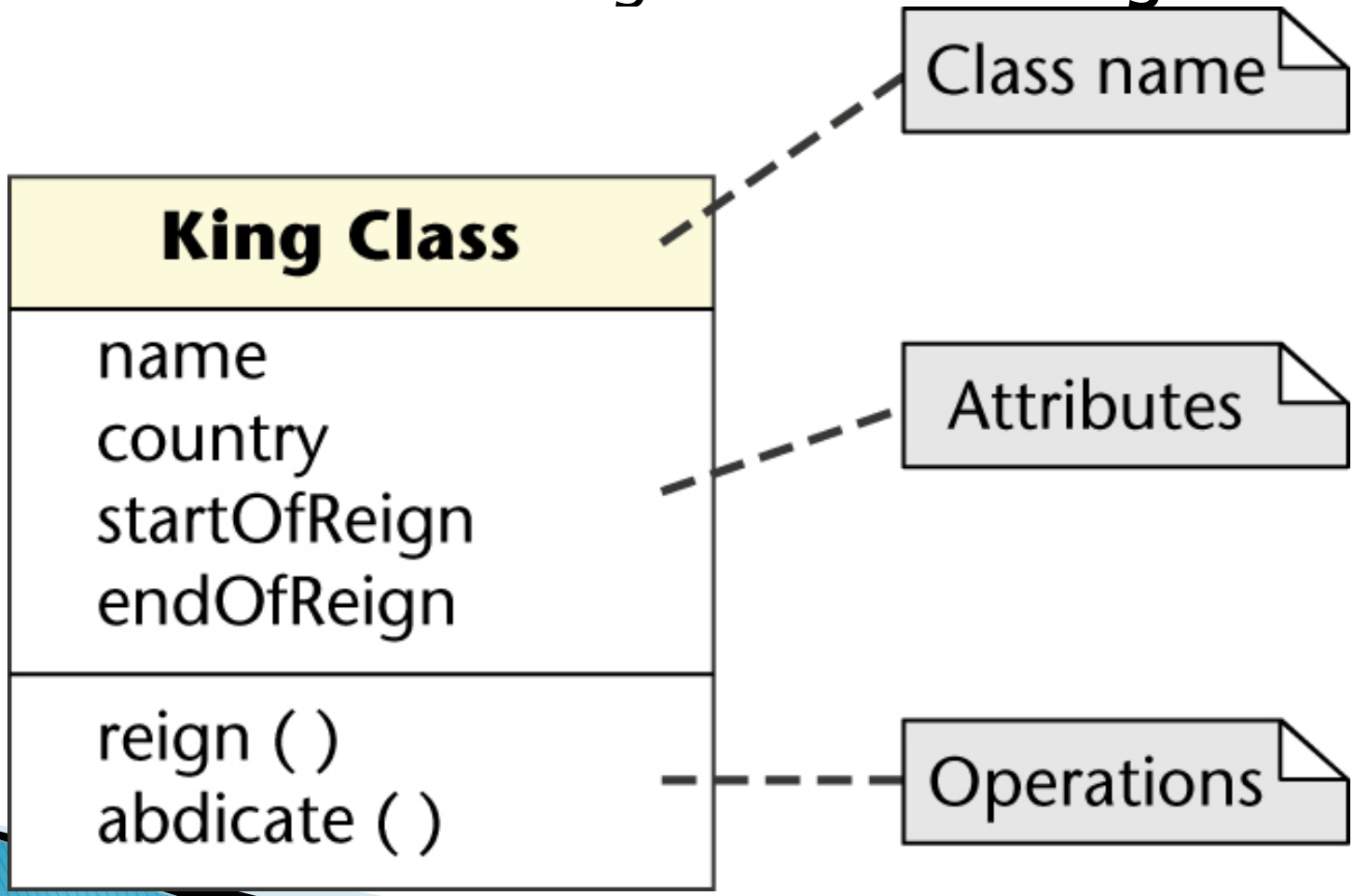
- Kings have something in common

name:	King George III
country:	Great Britain
startOfReign:	1760
endOfReign:	1820

name:	King Louis XVI
country:	France
startOfReign:	1774
endOfReign:	1792

King Class

- Let the set of all kings be called **King Class**



Classes and Objects

- King George III *is an instance of* **King Class**, and so is King Louis XVI

- A *class* is a set of related objects
  - Example: **King Class** is a set of kings

- Conversely, an *object* is an instance of a class
  - Example: King George III is an instance of **King Class**

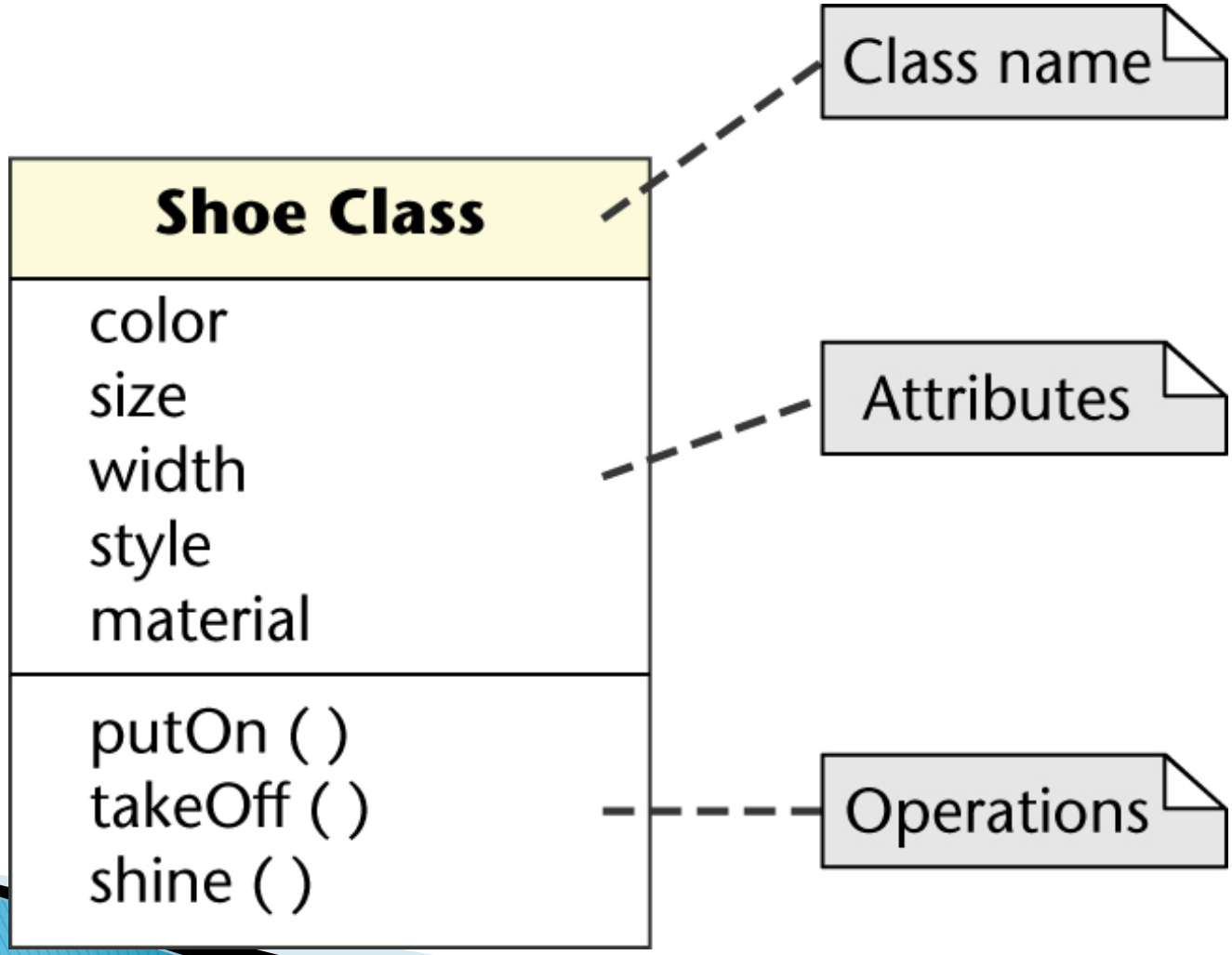
Shoes

- A shoe has a
  - Color (black, brown);
  - Size (3, 8 ½ , 9½ , 14);
  - Width (AAA, C, E);
  - Style (stiletto heel, wingtip, moccasin, penny loafer); and
  - Material (leather, plastic)

- Color, size, width, style, and material are *attributes* of shoes

Shoe Class

- **Shoe Class** is the name of the set of all shoes

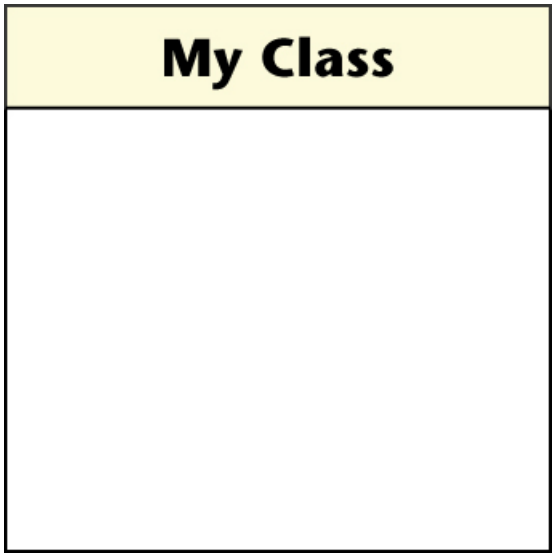


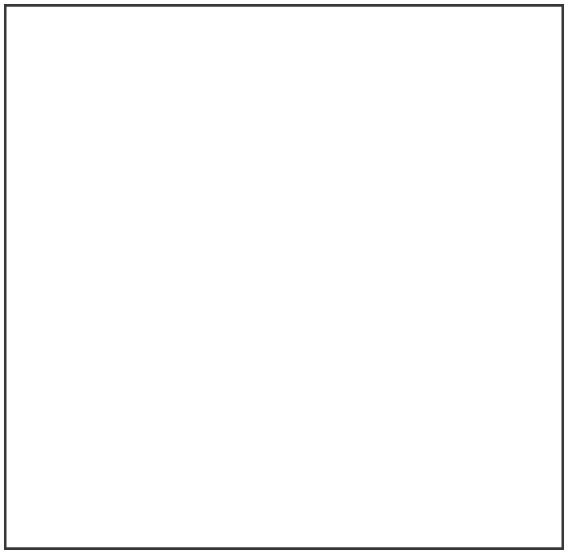
UML Notation for Classes

- There are three boxes
  - Top box: Class name
    - **Boldface With the Words in the Name Starting with Uppercase Letters**
  - Middle box: Attributes
  - Lowest box: Operations

- All the boxes are optional—see next slide

Valid UML Representations of a Class





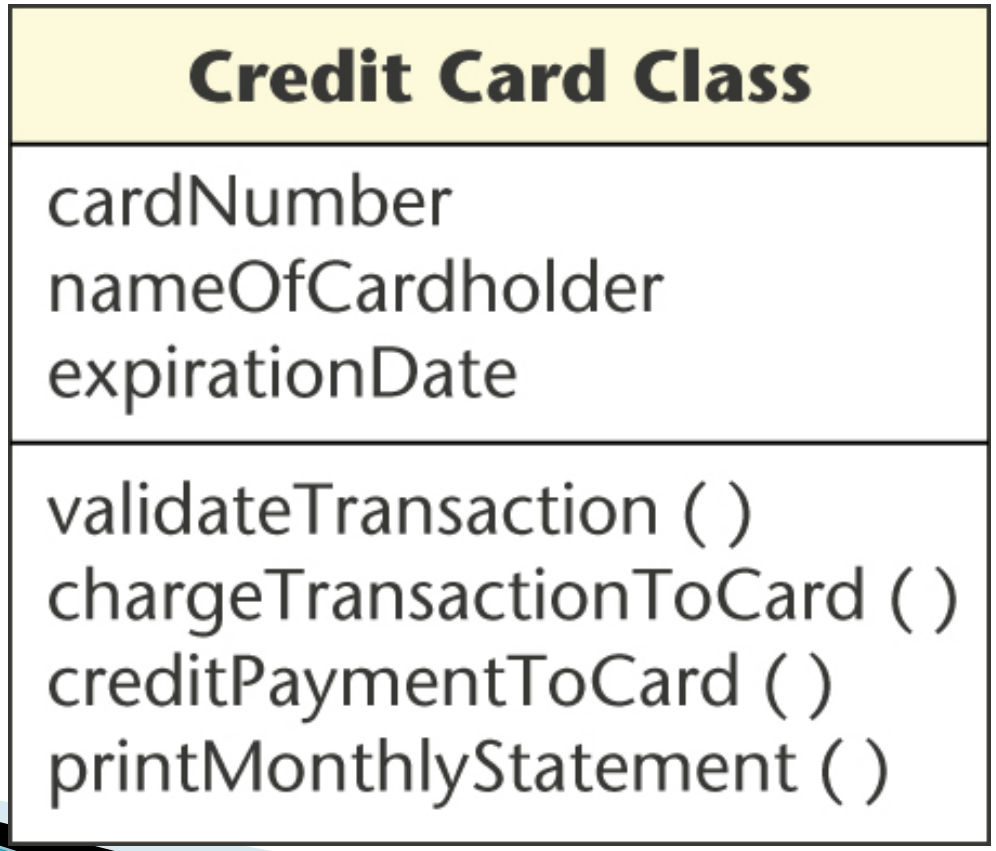
UML Convention for Objects

- The name of an object is written in
  - **boldface underlined but all in lowercase letters**

- Examples:
  - **A king** is an instance of **King Class**
  - A **shoe** is an instance of **Shoe Class**

Inheritance

- Example: Cardholder Clothing Company
  - The e-commerce company allows customers to purchase clothes using a credit card

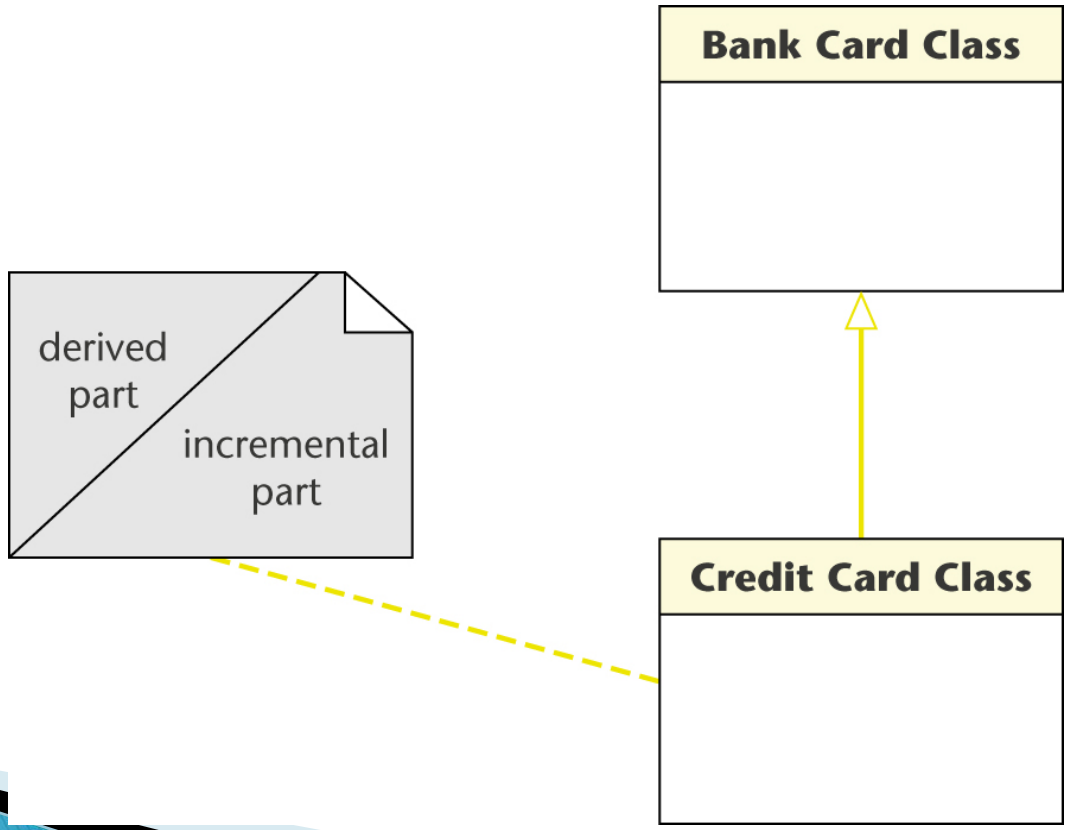


Inheritance (contd)

- The information system must be extended to allow purchasers to charge purchases to a debit card
- Credit and debit cards have many features in common, including attributes:
  - cardNumber;
  - nameOfCardholder; and
  - expirationDate
- The major difference:
  - A credit card has a credit limit
  - A debit card has a current balance

Inheritance (contd)

- Make **Debit Card Class** and **Credit Card Class** special cases of a more general class, **Bank Card Class**



Inheritance (contd)

- The figure on the previous slide shows that:
  - **Bank Card Class** is a class
  - **Credit Card Class** is a subclass of **Bank Card Class**

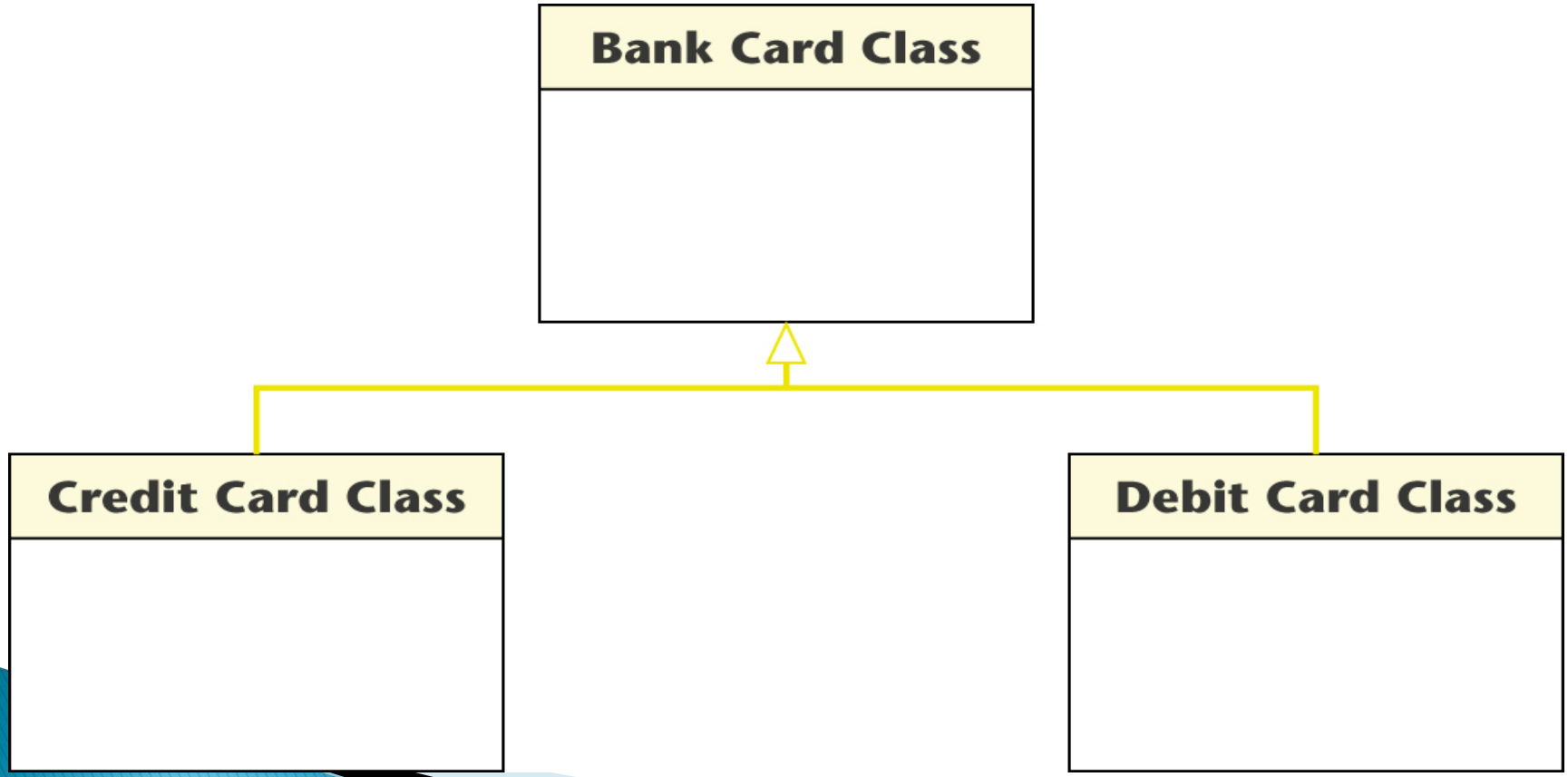
- The open triangle below **Bank Card Class** indicates that **Credit Card Class** *inherits* from **Bank Card Class.** This means that
  - **Credit Card Class** has all the features of **Bank Card Class** (the *derived part*)
  - It also has features of its own that are specific to **Credit Card Class** (the *incremental part*), such as a creditLimit

Cardholder Clothing Company (contd)

- The following statements are equivalent:
  - **Credit Card Class** is a *subclass* of **Bank Card Class**
  - **Bank Card Class** is a *superclass* of **Credit Card Class**
  - **Credit Card Class** is a *specialization* of **Bank Card Class**
  - **Bank Card Class** is a *generalization* of **Credit Card Class**
  - **Credit Card Class** *is a* **Bank Card Class**
  - **Bank Card Class** is the *base class*, **Credit Card Class** is the *derived class*
  - **Bank Card Class** is the *parent class*, **Credit Card Class** is the *child class*
  - **Credit Card Class** *inherits* from **Bank Card Class**

Inheritance (contd)

- **Credit Card Class** and **Debit Card Class** are both subclasses of class **Bank Card Class**



Inheritance (contd)

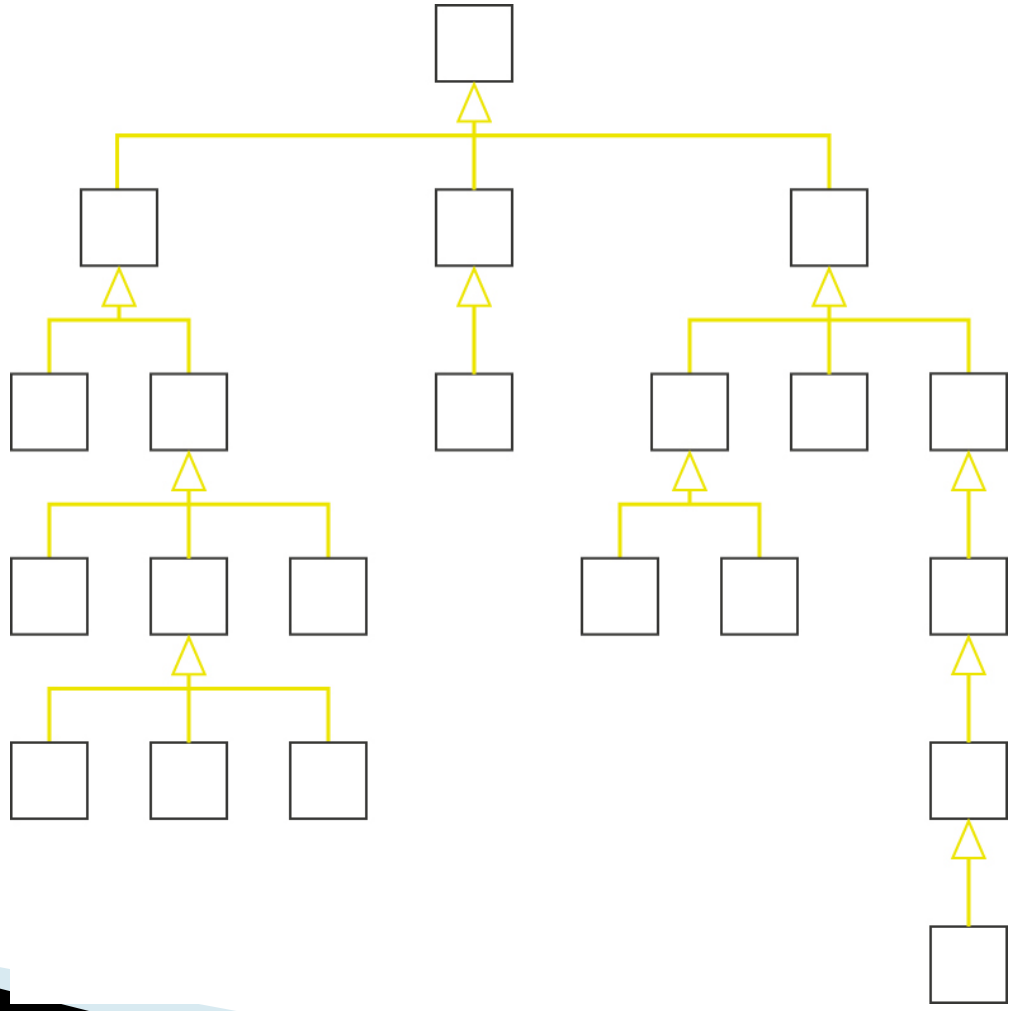
- The figure on the previous slides shows that:
  - **Credit Card Class** and **Debit Card Class** both inherit all the attributes and operations of **Bank Card Class**, such as
    - nameOfCardholder
    - validateTransaction
- In addition, **Credit Card Class** and **Debit Card Class** can have attributes and operations of their own
  - For example:
    - **Credit Card Class** has attribute creditLimit
    - **Debit Card Class** has operation determineAccountBalance

Inheritance (contd)

- Inheritance is a required feature of object orientation
  - Therefore, every object-oriented programming language supports inheritance, including
    - Smalltalk
    - C++
    - Ada 95
    - Java

Inheritance Hierarchy

- Inheritance can involve more than two classes



Relationships between Classes

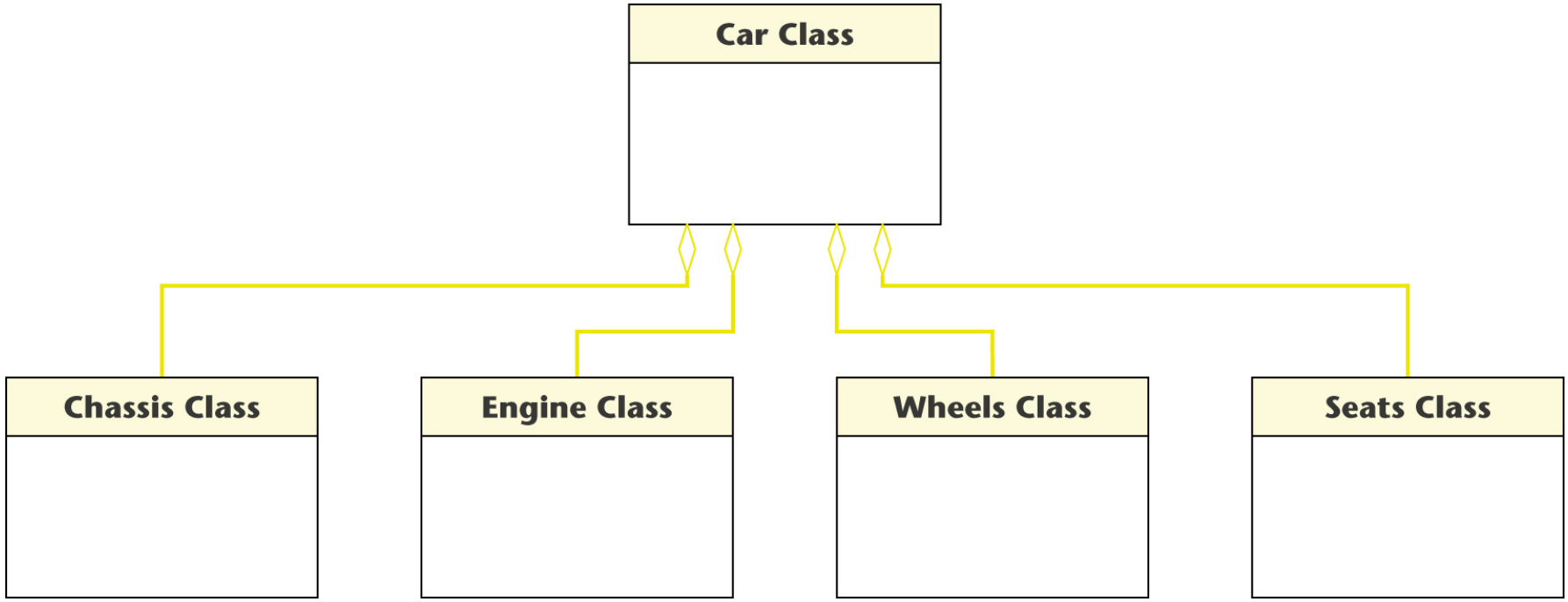
- Inheritance is an implementation of the *generalization* relationship

- Other possible relationships between classes include
  - Aggregation
  - Association

  These two relationships are now described

Aggregation

- Example: A car is constructed from a chassis, engine, wheels, and seats



Aggregation (contd)

- The *aggregation* relationship is appropriate when class **X** "consists of" classes **Y** and **Z**

- In UML, aggregation is represented by an open diamond
  - See the figure in the previous slide

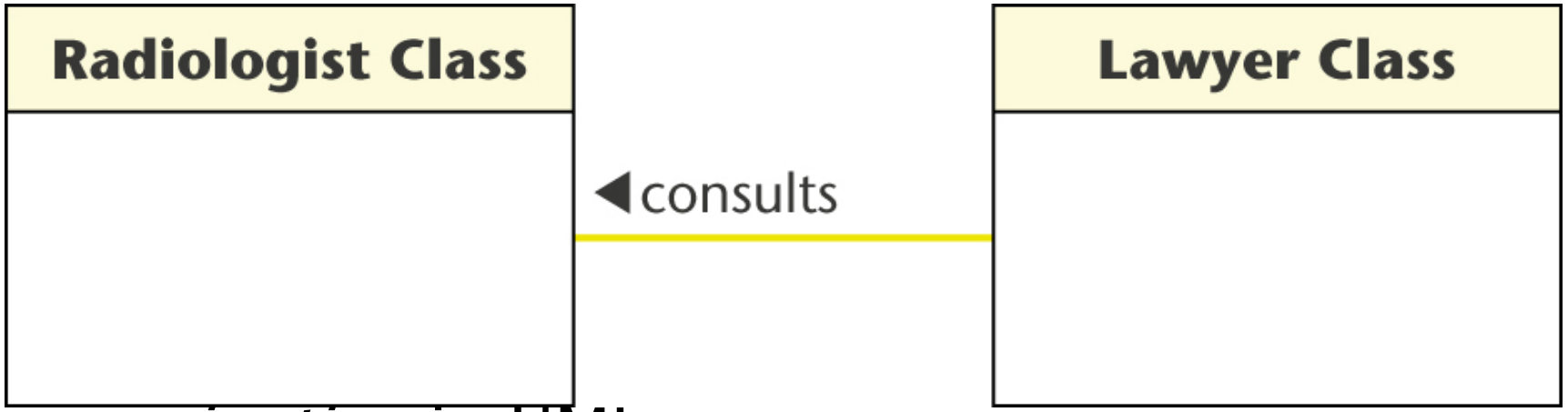
Association

- *Association* is an arbitrary relationship between two classes

- Example: A radiologist may consult a lawyer regarding the interpretation of a contract

Association

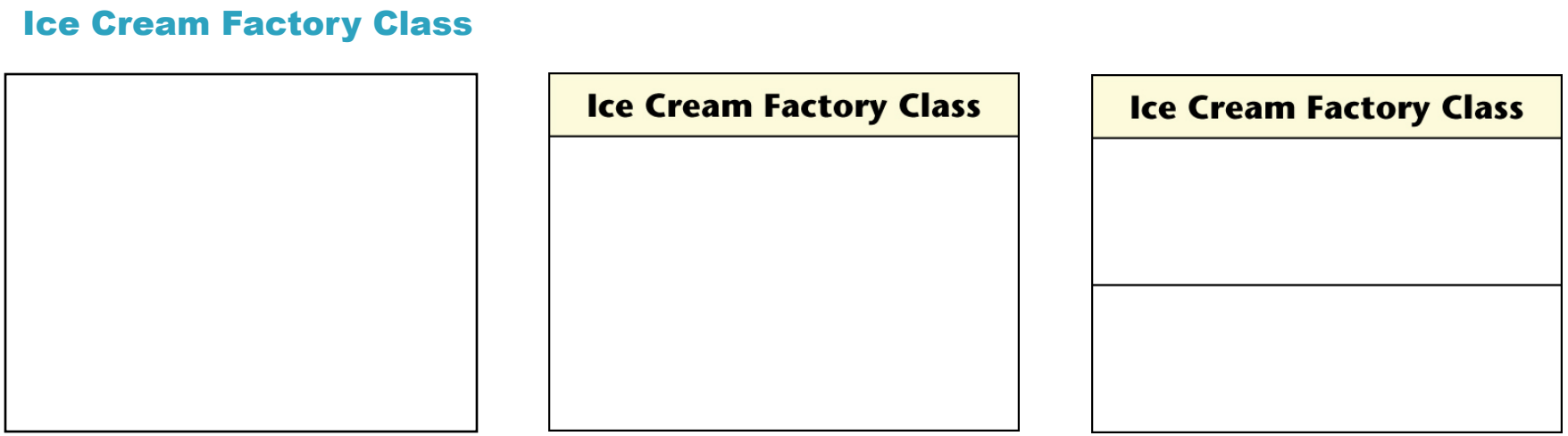
- Example: A lawyer with a broken leg consults a radiologist
  - The solid triangle points in the other direction



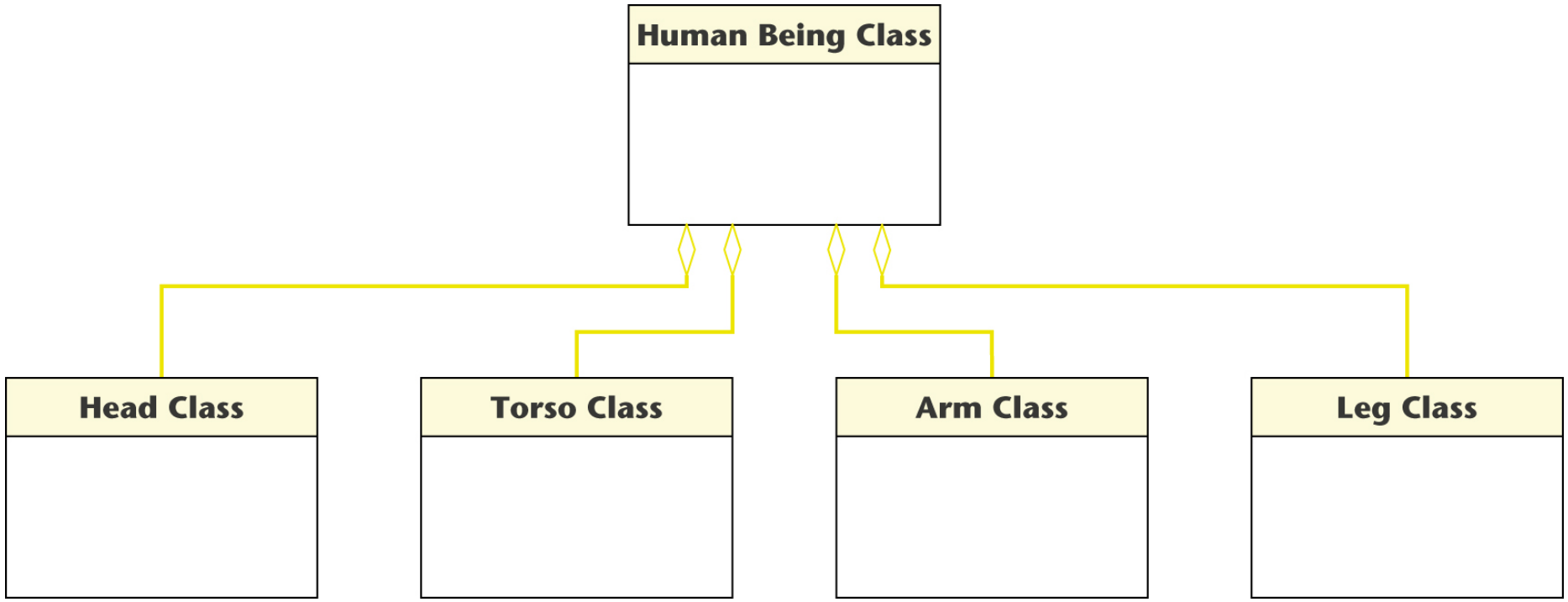
navigation in UML

UML Modeling Example: Ice-Cream Factories

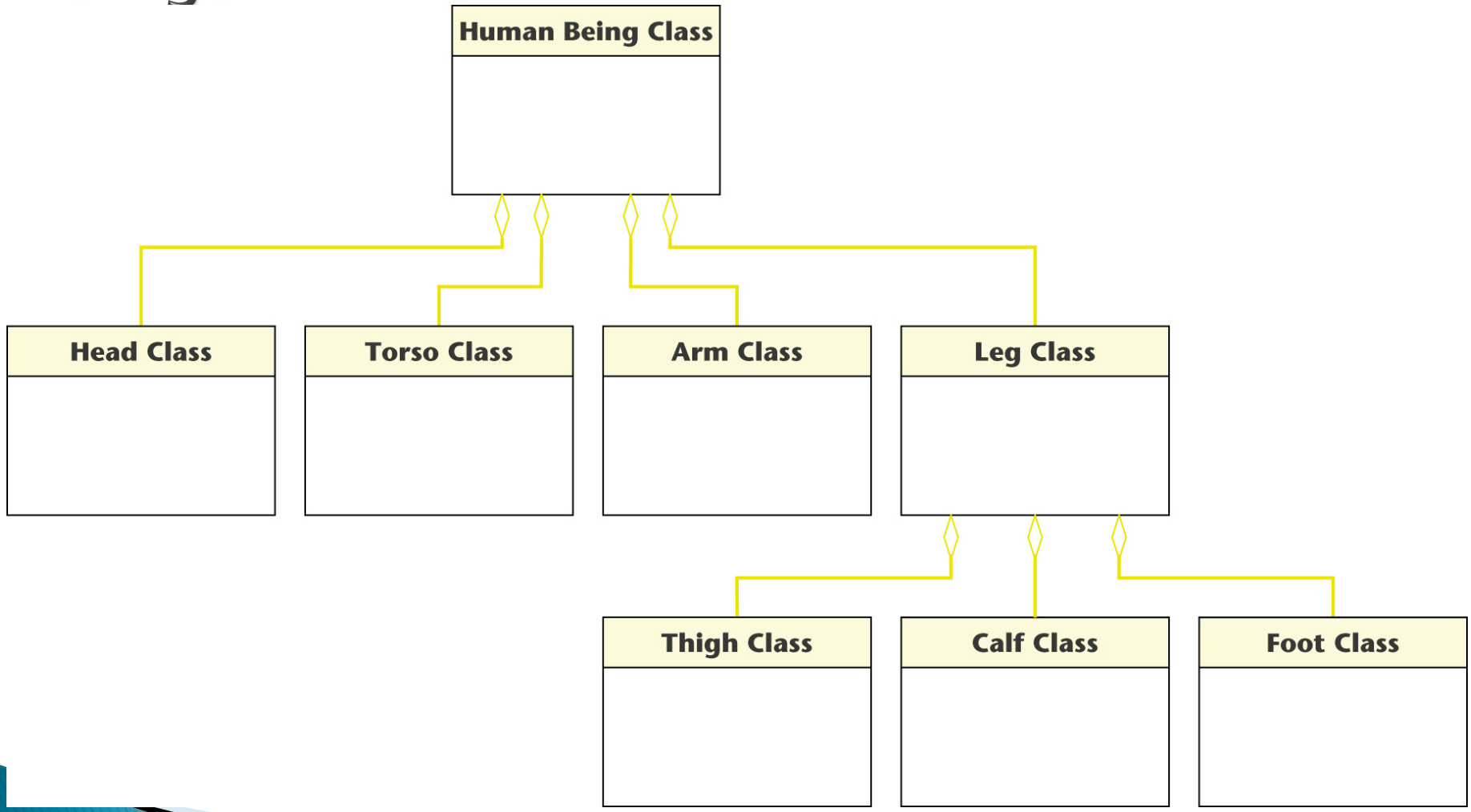
- Three equally valid UML *class diagrams*



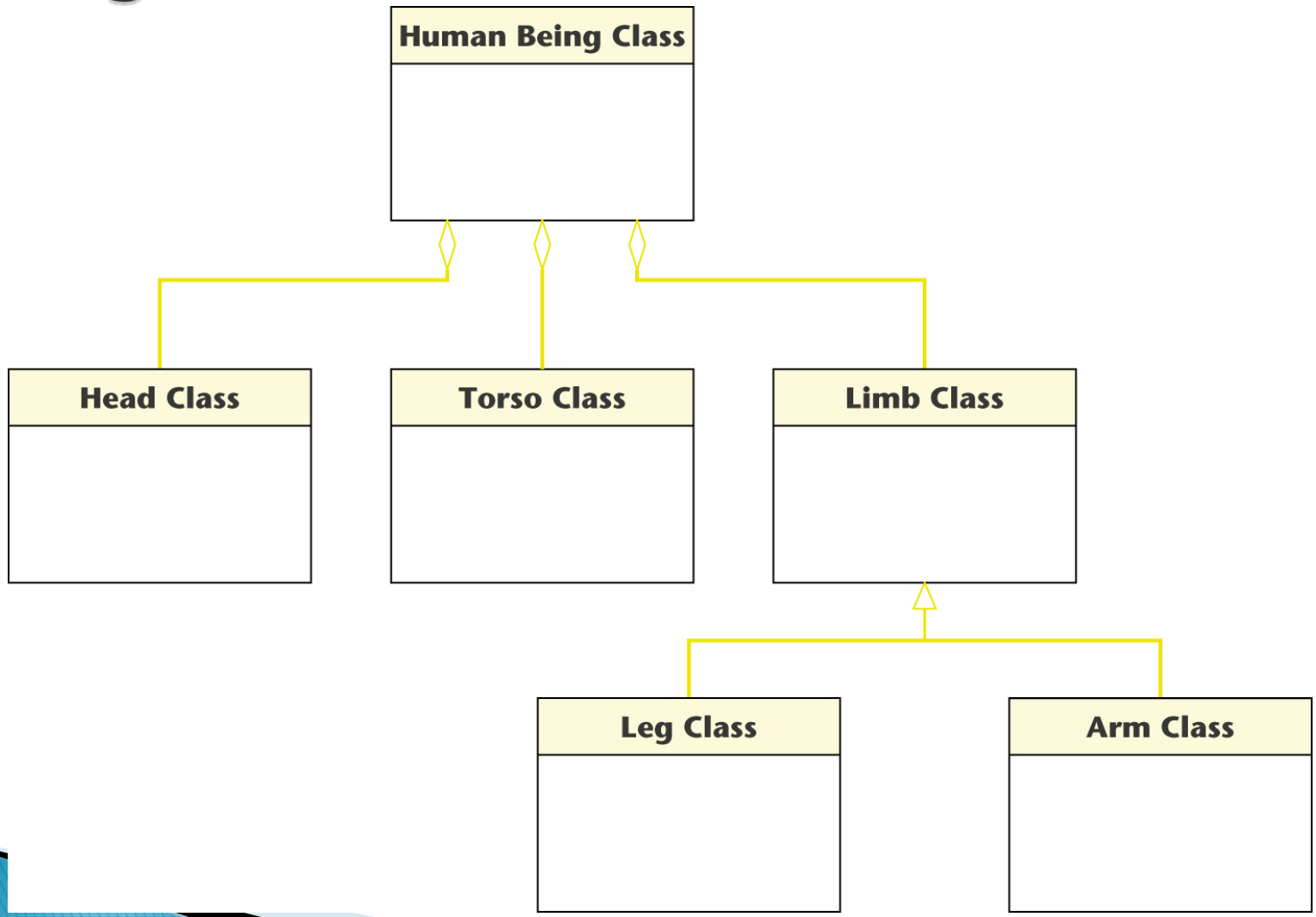
UML Modeling Example: Human Beings. 1



UML Modeling Example: Human Beings. 2



UML Modeling Example: Human Beings. 3

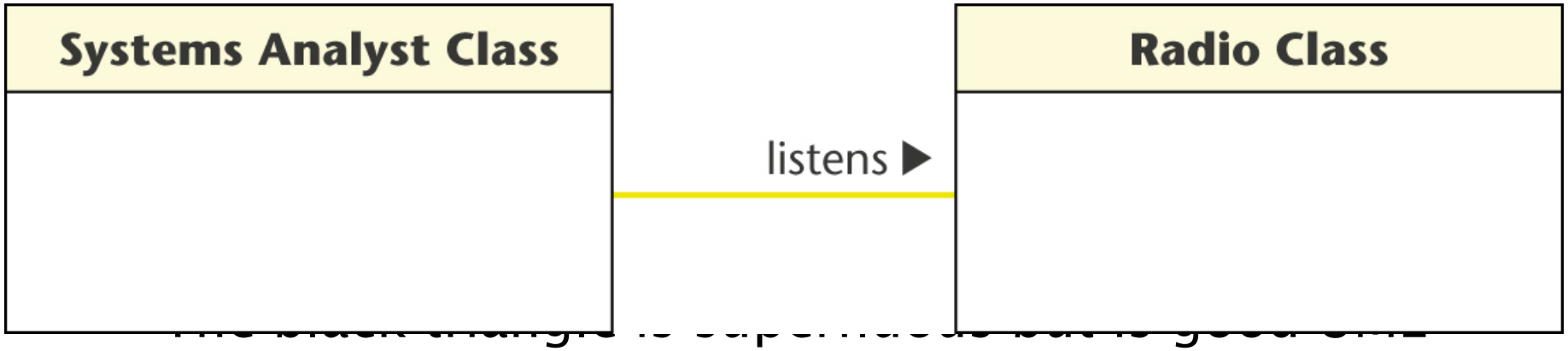


Alternative Models

- How can all three models all be correct?
  - Each model highlights different aspects
- Information systems can be modeled in a number of different but equally correct ways
  - The systems analyst must decide which model is the most appropriate

Systems Analysts Listening to the Radio

- At first sight there is no connection between systems analysts and radios
  - Model using association



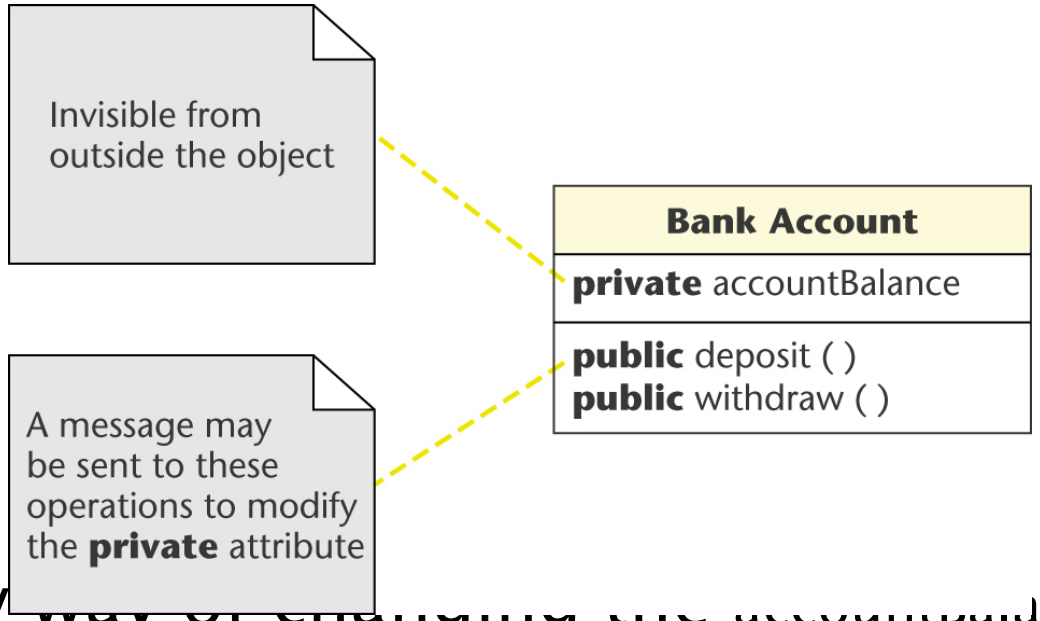
practice

Information Hiding

- *Information hiding* enhances maintainability
  - Implementation details are invisible outside an object
- Example: **Bank Account Class**
  - Attribute accountBalance can be implemented in many different ways:
    - As an integer (the total amount in cents); or
    - As a dollar amount with two decimal places for the cents
  - With information hiding, these details are invisible
- Object-oriented languages allow an attribute to be described as **private** (invisible outside the object)

Information Hiding (contd)

- For example, suppose that the two operations of **Bank Account Class** are deposit and withdraw



- The only way of changing the accountBalance of a **bank account** object is
  - To *send a message* to deposit or withdraw

Information Hiding (contd)

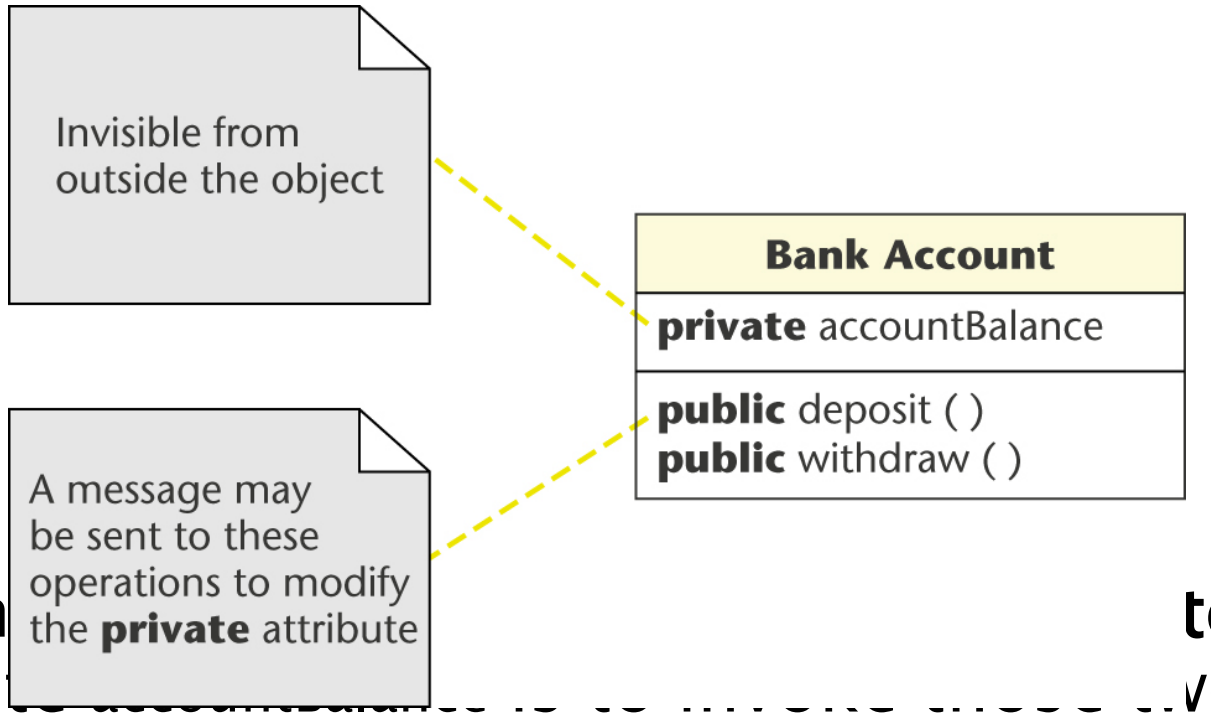
- Why do we do this?
  - Surely it is easier to modify accountBalance directly?
- When we change the way accountBalance is implemented
  - If we have information hiding, this change cannot affect any other part of the information system
- When there is information hiding, there cannot be regression faults
  - A *regression fault* is a fault in one part of the code caused by a change in an apparently unrelated part of the code

Information Hiding (contd)

- With information hiding
  - A system consists of essentially independent classes
  - They communicate by sending messages
- A **private** attribute
  - Can be accessed only from inside the class in which is declared
- A **public** attribute
  - Can be accessed from anywhere inside the information system
- The same rules apply to operations

Example: Bank Account Class

- **public** operations deposit and withdraw can be invoked from anywhere in the information system



- The on te attribu vo operations

Responsibility-Driven Design

- *Responsibility-driven design* goes beyond information hiding
  - Total responsibility for carrying out all aspects of the message rests with the object receiving the message

- Example: 1-800-flowers.com
  - It is irrelevant where 1-800-flowers.com is located
  - It is irrelevant which florist will be given your order to deliver
  - The company will not divulge that information (information hiding)

The Unified Process

- As pointed out earlier in this chapter:
  - There are problems when the traditional paradigm is used with medium- and large-scale information systems
  - The object-oriented paradigm solved these problems
- Between 1985 and 1995, over 50 different object-oriented methodologies were published
  - Three of the most successful were
    - Booch's method
    - Jacobson's Objectory
    - Rumbaugh's OMT

The Unified Process (contd)

- Booch, Jacobson, and Rumbaugh published a complete object-oriented analysis and design methodology that unified their three separate methodologies
  - Original name: *Rational Unified Process* (RUP)
  - Next name: *Unified Software Development Process* (USDP)
  - Name used today: *Unified Process* (for brevity)

The Unified Process (contd)

- The Unified Process is not a series of steps for constructing an information system
  - No such single "one size fits all" methodology could exist
  - There is such a wide variety of different types of information systems
- The Unified Process is an adaptable methodology
  - It has to be modified for the specific information system to be developed

The Unified Process (contd)

- The version of the Unified Process in this book is for
  - Information systems small enough to be developed by a team of three students during the semester or quarter
- However, the modifications for developing a large information system are also discussed
- The goals of this book:
  - A thorough understanding of how to develop smaller information systems
  - An appreciation of the issues that need to be addressed when larger information systems are constructed

The Unified Process (contd)

- The Unified Process is a modeling technique
  - A *model* is a set of UML diagrams that represent various aspects of the information system we want to develop
- UML stands for unified *modeling* language
  - UML is the tool that we use to represent (model) the target information system

The Unified Process (contd)

- UML is graphical
  - A picture is worth a thousand words
- UML diagrams enable information technology professionals to communicate quickly and accurately

Iteration and Incrementation

- The Unified Process is iterative and incremental
  - This is a consequence of Miller's Law (see Session 1)
- Each workflow consists of a number of steps
  - The steps are repeated until the UML model is accurate
- We can never get the object-oriented analysis and design right the first time

The Complete Unified Process

- We cannot learn the complete Unified Process in one semester or quarter
  - Extensive study and unending practice are needed
  - The Unified Process has too many features
  - A case study of a large-scale information system is huge

The Complete Unified Process (contd)

- In this book, we therefore cover most, but not all, of the Unified Process
  - The topics covered are adequate for smaller systems
- To work on larger systems, experience is needed
  - This must be followed by training in the more complicated aspects of the Unified Process