Vietnam and Japan
Joint ICT HRD Program

ITSS Software Development

Chapter 3. Introduction to OOAD

Nguyen Thi Thu Trang
trangntt-fit@mail.hut.edu.vn

Content

Content

1.1. Object Technology

- Object technology is used for creating models that reflect a specific domain using the terminology of the domain.
- Models created using object technology should be easy to create, change, expand, validate, and verify.
- Systems built using object technology are flexible to change, have well-defined architectures, and have the opportunity to create and implement reusable components

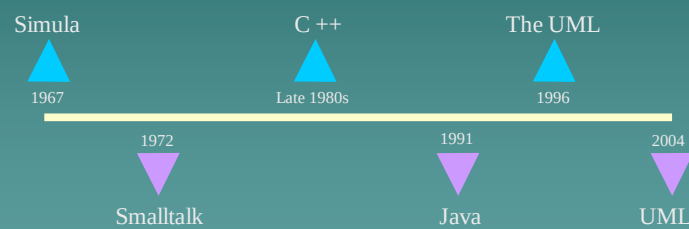
1.1.1. The Strengths of Object Technology

- Reflects a single paradigm
- Facilitates architectural and code reuse
- Reflects real world models more closely
- Encourages stability
- Is adaptive to change

5

1.1.2. The History of Object Technology

- Major object technology milestones



6

1.1.3. Where Is Object Technology Used?

- Client/Server Systems and Web Development
  - Object technology allows companies to encapsulate business information in objects and helps to distribute processing across the Internet or a network.

1.1.3. Where Is Object Technology Used (2)?

- Real-time systems
  - Object technology enables real-time systems to be developed with higher quality and flexibility.

Discussion

- What is your perception of object technology?
- What do you perceive as object technology's strengths? Its weaknesses?
- Why are you making the shift to object technology?

1.2. Modeling

- A model is a simplification of reality.

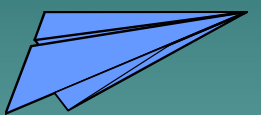
Why Model?

- Modeling achieves four aims [1]:
  - Helps you to “visualize a system as you want it to be”.
  - Permits you to “specify the structure or behavior of a system”.
  - Gives you “a template that guides you in constructing a system”.
  - “Documents the decisions you have made”.
- You build models of complex systems because you cannot comprehend such a system in its entirety.
- You build models to better understand the system you are developing.

[1]: Chapter 1, Section 1.1

The Importance of Modeling

Less Important — More Important



Paper Airplane

Fighter Jet

1.3. Unified Modeling Language (UML)

- "The UML is a language for
  - Visualizing
  - Specifying
  - Constructing
  - Documenting

  the artifacts of a software-intensive system" [1].

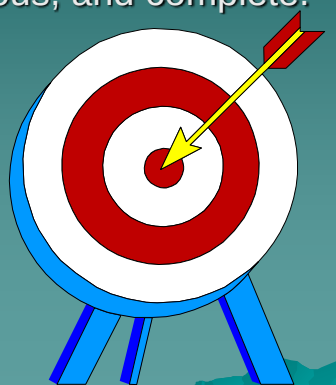
[1]: Chapter 2, Section 2.1

The UML Is a Language for Visualizing

- Communicating conceptual models to others is prone to error unless everyone involved speaks the same language.
- There are things about a software system you can't understand unless you build models.
- An explicit model facilitates communication.

The UML Is a Language for Specifying

- The UML builds models that are precise, unambiguous, and complete.

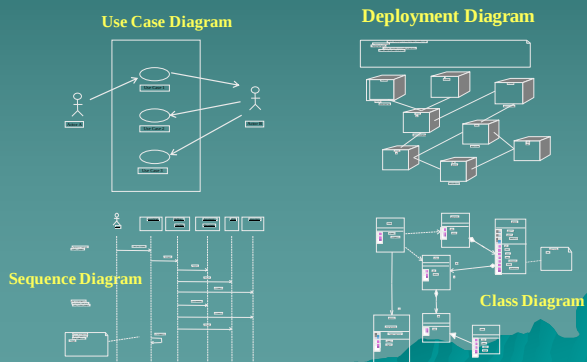


The UML Is a Language for Constructing

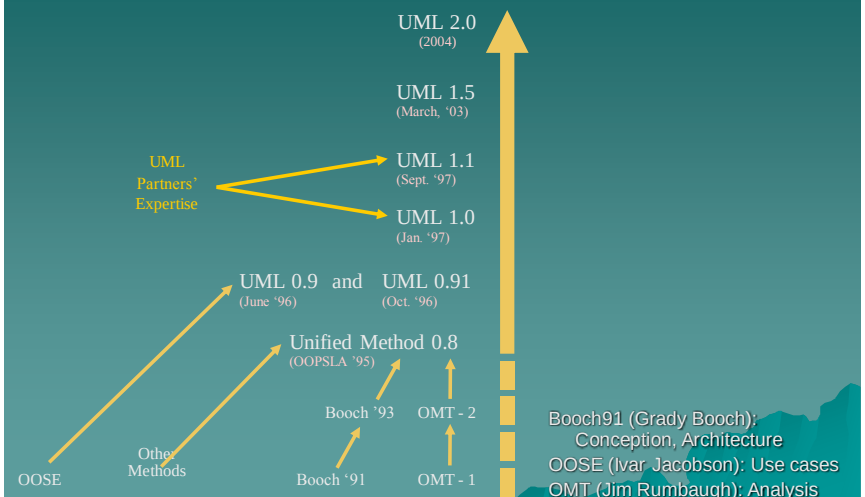
- UML models can be directly connected to a variety of programming languages.
  - Maps to Java, C++, Visual Basic, and so on
  - Tables in a RDBMS or persistent store in an OODBMS
  - Permits forward engineering
  - Permits reverse engineering

The UML Is a Language for Documenting

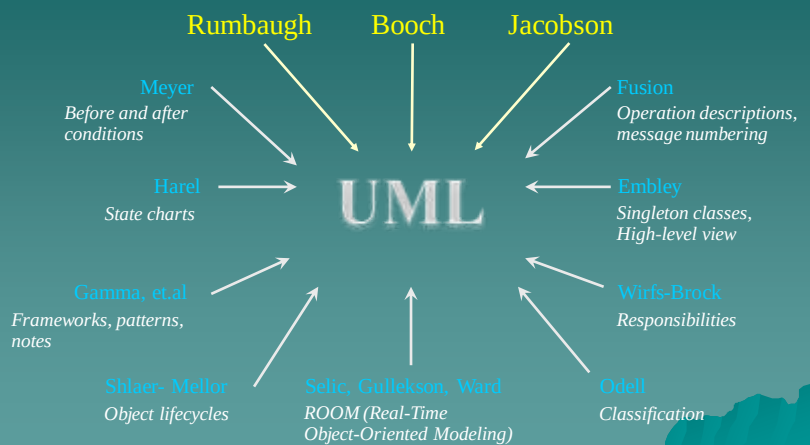
- The UML addresses documentation of system architecture, requirements, tests, project planning, and release management



History of the UML



Inputs to the UML



UML Views

- "A view is simply a subset of UML modeling constructs that represents one aspect of a system" [2]
- Four areas
  - structural classification,
  - dynamic behavior
  - physical layout,
  - and model management.

[2]: Part 2, Chapter 3, Section 3.1

20

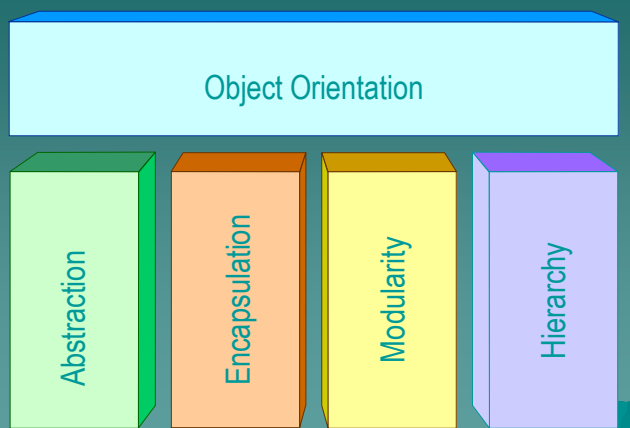
UML Views [2]

Major area	View	Diagram
Structural	Static view	Class diagram
	Design view	Internal structure, Collaboration diagram Component diagram
	Use case view	Use case diagram
Dynamic	State machine view	State machine diagram
	Activity view	Activity diagram
	Interaction View	Sequence Diagram Communication Diagram
Physical	Deployment View	Deployment Diagram
Model Management	Model Management View	Package diagram
	Profile	Package Diagram

[2]: Part 2, Chapter 3, Section 3.1, Table 3.1 (extracted)

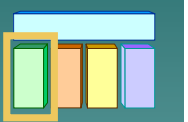
Content

2. Basic Principles of Object Orientation



2.1. What Is Abstraction?

- "The act of identifying the essential characteristics of a thing that distinguish it from all other kinds of things and omitting details that are unimportant from a certain viewpoint".
- "Abstraction involves looking for similarities across sets of things by focusing on their essential common characteristics".
- "An abstraction always involves the perspective and purpose of the viewer; different purposes result in different abstractions for the same things".



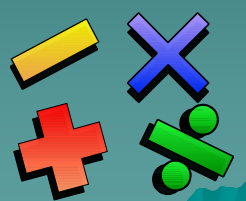
[2]: Part 3, Abstraction, pp 134.

Example: Abstraction

Student

Professor

Course Offering (9:00 a.m., Monday-Wednesday-Friday)



Course (e.g. Algebra)

2.2. What Is Encapsulation?

- The physical localization of features (for example, properties, behaviors) into a single blackbox abstraction that hides their implementation (and associated design decisions) behind a public interface.
- Hides implementation from clients
  - Clients depend on interface.

Improves Resiliency

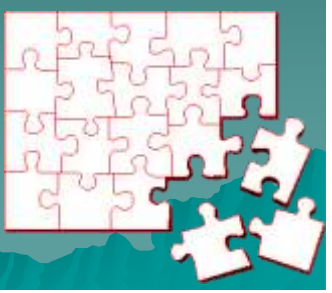
Encapsulation Illustrated

- Professor Clark needs to be able to teach four classes in the next semester.

SetMaxLoad(4)

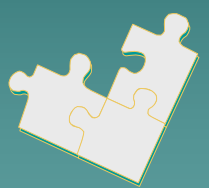
2.3. What Is Modularity?

- The logical and physical decomposition of things (for example, responsibilities and software) into small, simple groupings (for example, requirements and classes, respectively), which increase the achievements of software-engineering goals.
- Helps people understand complex systems.



Example: Modularity

- For example, break complex systems into smaller modules.



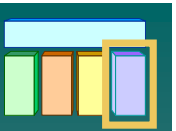
University System

Billing System

Course Registration System

Student Management System

2.4. What Is Hierarchy?



- Any ranking or ordering of abstractions into a tree-like structure. Kinds: Aggregation hierarchy, class hierarchy, containment hierarchy, inheritance hierarchy, partition hierarchy, specialization hierarchy, type hierarchy.

30

Example: Hierarchy?

Elements at the same level of the hierarchy should be at the same level of abstraction.

Content

1. Object Technology and UML
2. Basic Principles of Object Orientation
3. Basic Concepts of Object Orientation
4. Object-Oriented Analysis and Design
5. OOAD Tools

3.1. Class and Object

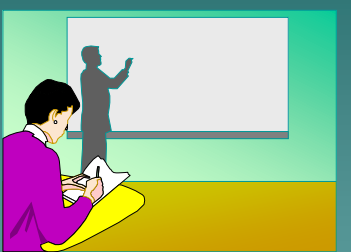
- "A class is a description of a set of objects that share the same *attributes*, *operations*, *relationships*, and *semantics"* [1].
  - An object is an instance of a class.
- A class is an abstraction in that it
  - Emphasizes relevant characteristics.
  - Suppresses other characteristics.

[1]: Chapter 4 (Overview part)

A Sample Class

Class
Course

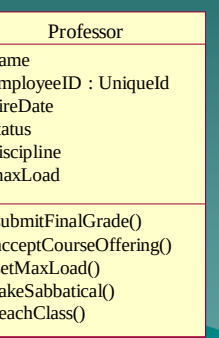
Properties
Name
Location
Days offered
Credit hours
Start time
End time



Behavior
Add a student
Delete a student
Get course roster
Determine if it is full

Representing Classes in the UML

- A class is represented using a rectangle with three compartments:
  - The class name
  - The structure (attributes)
  - The behavior (operations)

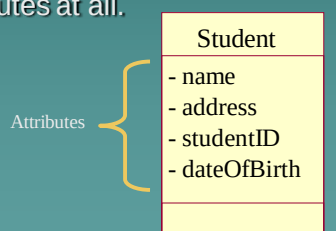


The Relationship between Classes and Objects

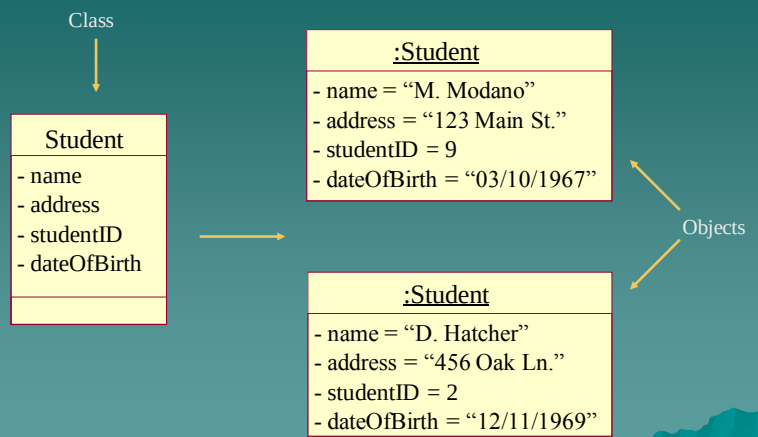
- A class is an abstract definition of an object.
  - It defines the structure and behavior of each object in the class.
  - It serves as a template for creating objects.
- Classes are not collections of objects.

What Is an Attribute?

- An attribute is a named property of a class that describes the range of values that instances of the property may hold.
  - A class may have any number of attributes or no attributes at all.

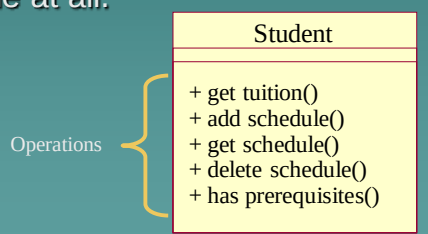


Attributes in Classes and Objects



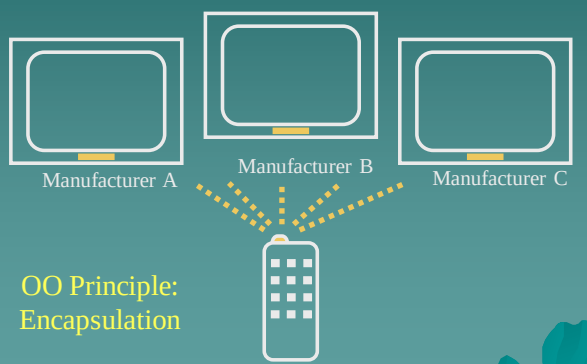
What Is an Operation?

- A service that can be requested from an object to effect behavior. An operation has a signature, which may restrict the actual parameters that are possible.
- A class may have any number of operations or none at all.

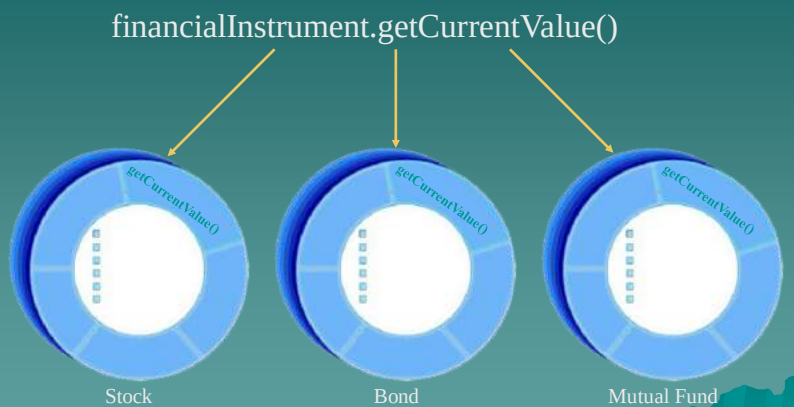


3.2. What Is Polymorphism?

- The ability to hide many different implementations behind a single interface.



Example: Polymorphism

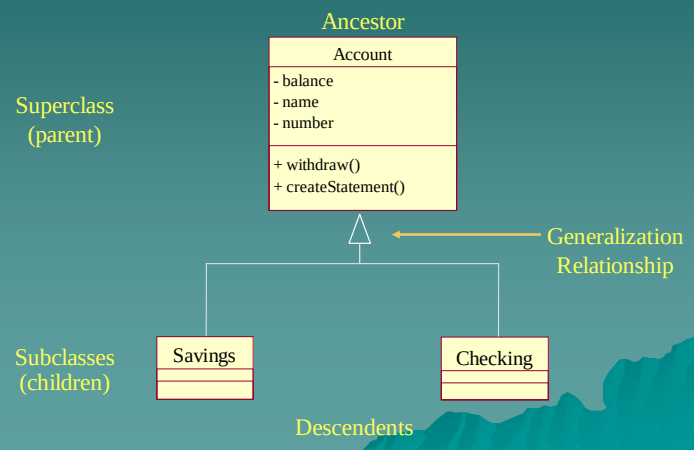


3.2. What Is Generalization?

- A relationship among classes where one class shares the structure and/or behavior of one or more classes.
- Defines a hierarchy of abstractions in which a subclass inherits from one or more superclasses.
  - Single inheritance.
  - Multiple inheritance.
- Is an “is a kind of” relationship.

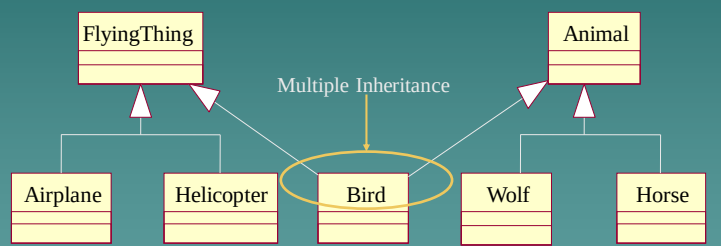
Example: Single Inheritance

- One class inherits from another.



Example: Multiple Inheritance

- A class can inherit from several other classes.



Use multiple inheritance only when needed and always with caution!

What Is Inherited?

- A subclass inherits its parent's attributes, operations, and relationships.
- A subclass may:
  - Add additional attributes, operations, relationships.
  - Redefine inherited operations. (Use caution!)
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy.

Inheritance leverages the similarities among classes.

3.4. What Is a Package?

- A general purpose mechanism for organizing elements into groups.
- A model element that can contain other model elements.
- A package can be used:
  - To organize the model under development.
  - As a unit of configuration management.

A Package Can Contain Classes

- The package, University Artifacts, contains one package and five classes.

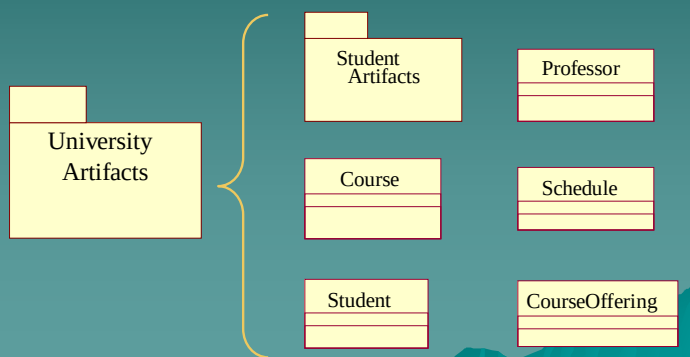
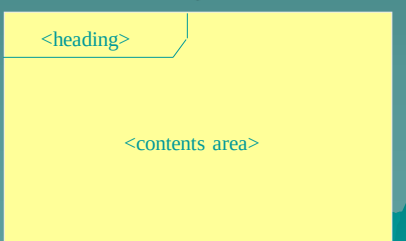


Diagram Depiction

- Each diagram has a frame, a heading compartment in the upper left corner, and a contents area.
  - If the frame provides no additional value, it may be omitted and the border of the diagram area provided by the tool will be the implied frame.



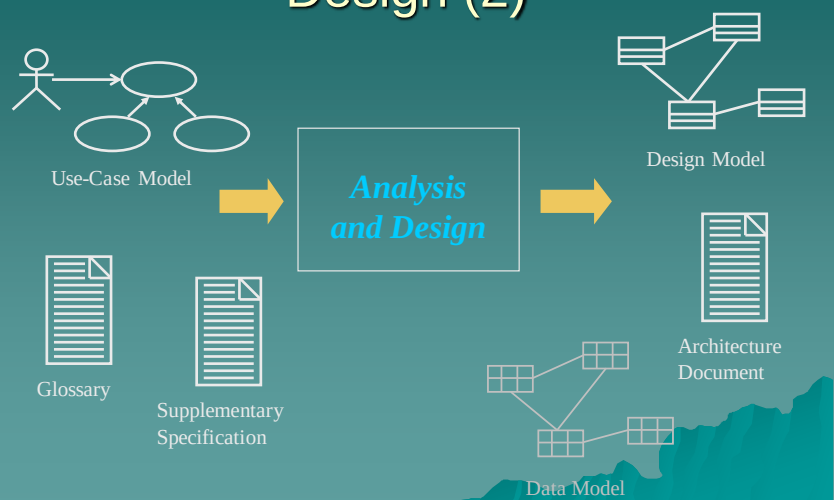
Content

1. Object Technology and UML
2. Basic Principles of Object Orientation
3. Basic Concepts of Object Orientation
4. Object-Oriented Analysis and Design
5. OOAD Tools

4.1. Purpose of Analysis and Design

- Transform the requirements into a design of the system-to-be.
- Evolve a robust architecture for the system.
- Adapt the design to match the implementation environment, designing it for performance.

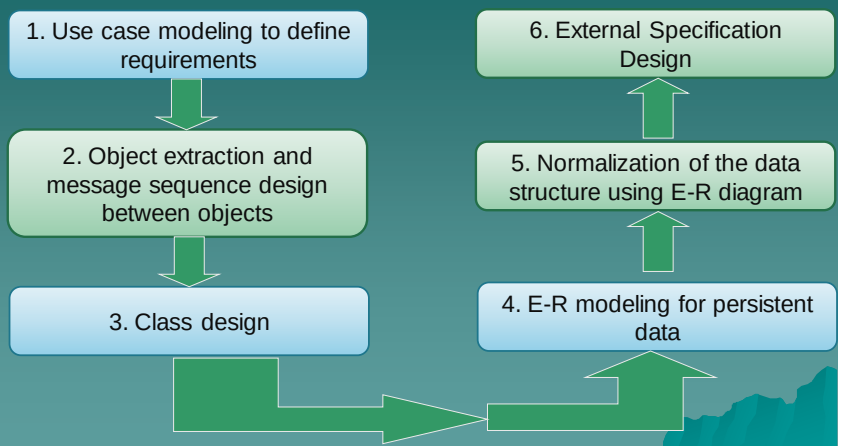
4.1. Purpose of Analysis and Design (2)



4.2. Analysis vs. Design

Analysis	Design
▪ Focus on understanding the problem ▪ Idealized design ▪ Behavior ▪ System structure ▪ Functional requirements ▪ A small model	▪ Focus on understanding the solution ▪ Operations and attributes ▪ Performance ▪ Close to real code ▪ Object lifecycles ▪ Nonfunctional requirements ▪ A large model

4.3. OOAD Steps



Content

1. Object Technology and UML
2. Basic Principles of Object Orientation
3. Object-Oriented Analysis and Design
4. OOAD Tools

4. OOAD Tools

- Open source tools
  - EclipseUML
  - UmlDesigner
  - ArgoUML...
- Commercial tools
  - **Enterprise Architect**
  - IBM Rational Software Architect
  - Microsoft Visio
  - Visual Paradigm for UML
  - SmartDraw...

Enterprise Architect

- Commercial product of Sparx System
  - A robust tool for developing and modeling software
  - Ligh-weigth (~40MB)
  - Can be integrated with Visual Studio, Eclipse
  - The latest version, Enterprise Architect 7.5, supports UML 2.1 and related standards.

References

[1] Unified Modeling Language User Guide, The (2nd Edition); Grady Booch, James Rumbaugh, Ivar Jacobson; Addison-Wesley Professional; 2005.

[2] The Unified Modeling Language Reference Manual (2nd Edition); James Rumbaugh, Ivar Jacobson, Grady Booch; Addison-Wesley Professional; 2005.

Question?