Introduction to UML

About this lecture…

- Will attempt to introduce you to UML and Rational Rose
- Not possible to teach everything
- Requires that you study on you own after
- Goal is to get you familiar
- Make use of on-line tutorials, books, etc.

Introduction to UML

- What is UML?
- Motivations for UML
- Types of UML diagrams
- UML syntax
- Descriptions of the various diagram types
  - Rational Rose and UML
- UML pitfalls

What is UML?

- A standardized, graphical "modeling language" for communicating software design.

- Allows implementation-independent specification of:
    - user/system interactions (required behaviors)
    - partitioning of responsibility (OO)
    - integration with larger or existing systems
    - data flow and dependency
    - operation orderings (algorithms)
    - concurrent operations

- Pretty pictures.

- UML is not "process". (That is, it doesn't tell you how to do things, only what you should do.)

Motivations for UML

- UML is a fusion of ideas from several precursor modeling languages.
- We need a modeling language to:
  - help develop efficient, effective and correct designs, particularly Object Oriented designs.
  - communicate clearly with project stakeholders (concerned parties: developers, customer, etc).
  - give us the "big picture" view of the project.

Types of UML diagrams

- There are different types of UML diagram, each with slightly different syntax rules:
    - use cases.
    - class diagrams.
    - sequence diagrams.
    - package diagrams.
    - state diagrams
    - activity diagrams
    - deployment diagrams.

UML syntax, 1

- Actors: a UML actor indicates an interface (point of interaction) with the system.
  - We use actors to group and name sets of system interactions.
  - Actors may be people, or other systems.
  - An actor is NOT part of the system you are modeling. An actor is something external that your system has to deal with.
- Boxes: boxes are used variously throughout UML to indicate discrete elements, groupings and containment.

UML syntax, 2

- Arrows: arrows indicate all manner of things, depending on which particular type of UML diagram they're in. Usually, arrows indicate flow, dependency, association or generalization.

- Cardinality: applied to arrows, cardinalities show relative numerical relationships between elements in a model: 1 to 1, 1 to many, etc.

UML syntax, 3

- Constraints: allow notation of arbitrary constraints on model elements. Used, for example, to constrain the value of a class attribute (a piece of data).

- Stereotypes: allow us to extend the semantics of UML with English. A stereotype is usually a word or short phrase that describes what a diagram element does. That is, we mark an element with a word that will remind us of a common (stereotypical) role for that sort of thing. Stereotypes should always be applied consistently (with the same intended meaning in all instances).

UML diagrams: use cases

- A use case encodes a typical user interaction with the system. In particular, it:
  - captures some user-visible function.
  - achieves some concrete goal for the user.
- A complete set of use cases largely defines the requirements for your system: everything the user can see, and would like to do.
- The granularity of your use cases determines the number of them (for you system). A clear design depends on showing the right level of detail.
- A use case maps actors to functions. The actors need not be people.

Use case examples, 1
(High-level use case for powerpoint.)
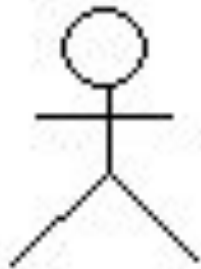
Create slide presentation

About the last example...

- Although this is a valid use case for powerpoint, and it completely captures user interaction with powerpoint, it's too vague to be useful.

Use case examples, 2
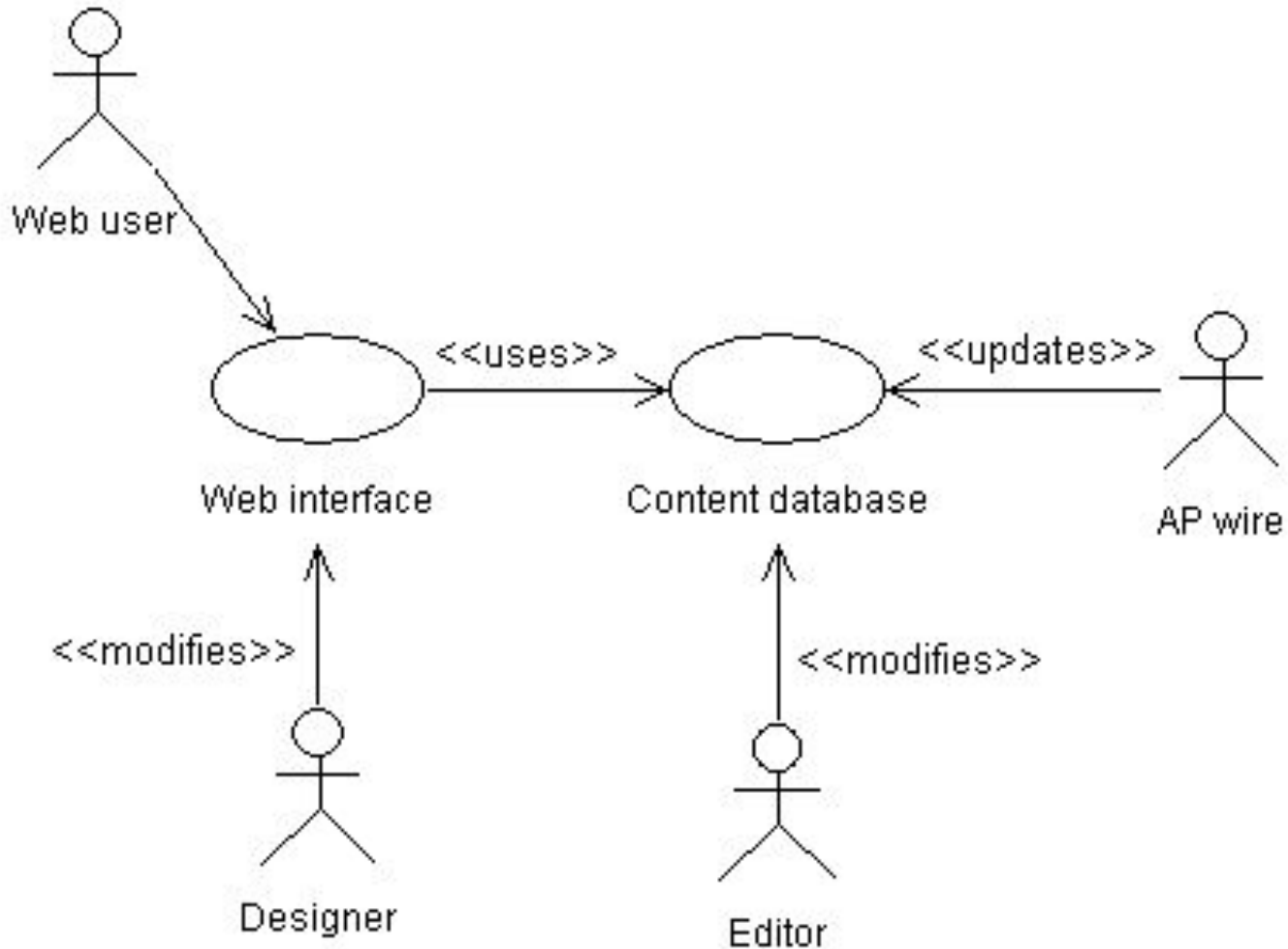## (Finer-grained use cases for powerpoint.)

13
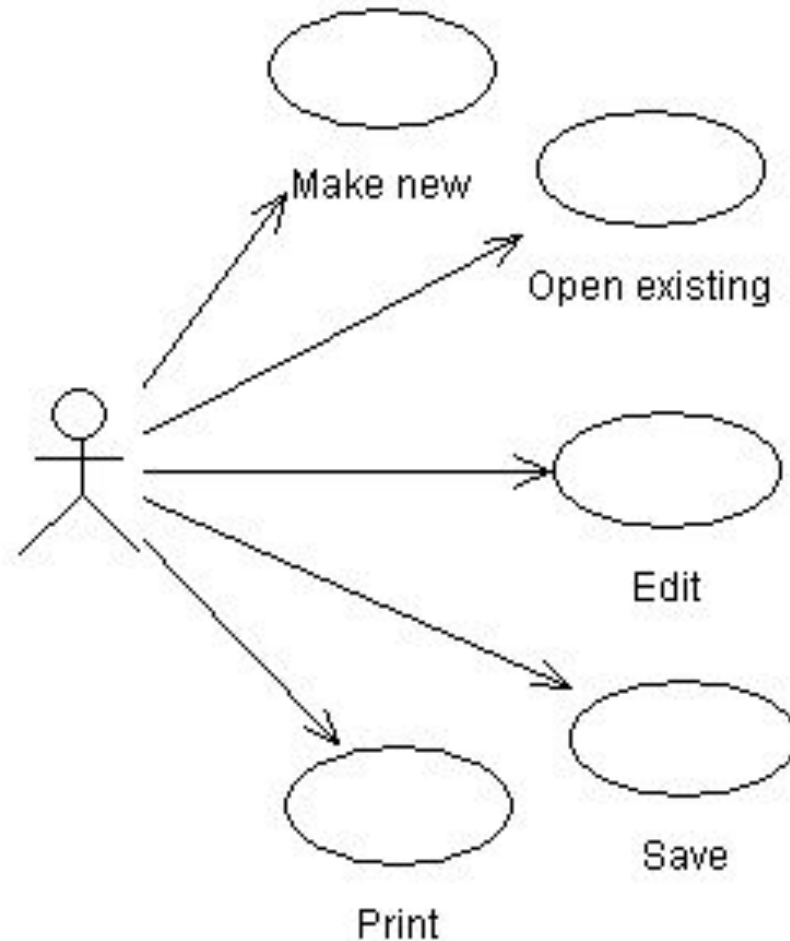
About the last example...

- The last example gives a more useful view of powerpoint (or any similar application).

- The cases are vague, but they focus your attention the key features, and would help in developing a more detailed requirements specification.

- It still doesn't give enough information to characterize powerpoint, which could be specified with tens or hundreds of use cases (though doing so might not be very useful either).

Use case examples, 3
(Relationships in a news web site.)

Web user

<<uses>>

<<updates>>

Web interface

Content database

AP wire

<<modifies>>

<<modifies>>

Designer

Editor

About the last example...

- The last is more complicated and realistic use case diagram. It captures several key use cases for the system.

- Note the multiple actors. In particular, 'AP wire' is an actor, with an important interaction with the system, but is not a person (or even a computer system, necessarily).

- The notes between << >> marks are *stereotypes:* identifiers added to make the diagram more informative. Here they differentiate between different roles (ie, different meanings of an arrow in this diagram).

More UML later,

now on to…

Using Rational Software

IBM Rational resources

- Getting the program:
  - http://www-01.ibm.com/software/rational/
  - http://en.wikipedia.org/wiki/Rational_Software

- Installed in A32

What is Rational Rose?

- An expensive CASE (Computer-Aided Software Engineering) tool for object-oriented modeling.
- Based on UML (more or less).
- Provides semantics (a 'compiler') for UML.
- Has a reasonably intuitive GUI similar to standard drawing programs, like Illustrator. Is available for Windows and other platforms.
- Makes creating and maintaining your UML diagrams easier (or at least more consistent).
- Has many bizarre features, including generation of C++ (and other) code from your diagrams.

What is a Rose Model?

- A Rose "model" is a representation of the problem domain and system software
  - Each model contains views, diagrams, and specifications to visualize and manipulate the elements in the model
  - There are many views of each underlying element
  - Every "object" in the design is represented once in the Rose "model"
  - Rose maintains a consistent semantic representation in the "model"

Rational
unifying software teams

About the next 2 slides…

- The next slides introduce the Rose UI.
- Main points are the:
  - "Browser", a drop-down list of things in your model.
  - "Documentation window", where you can add notes to a thing in your model.
  - "Diagram windows", where you draw your pictures.

The Rose User Interface

- Rose consists of:
  - Standard toolbar
  - Diagram toolbar
  - Browser
  - Documentation window
  - Diagram windows
  - Specifications
  - Status bar

23

Rati⊙nal
unifying software teams

Rational Rose

Standard Toolbar

Browser

Documentation Window

Status Bar
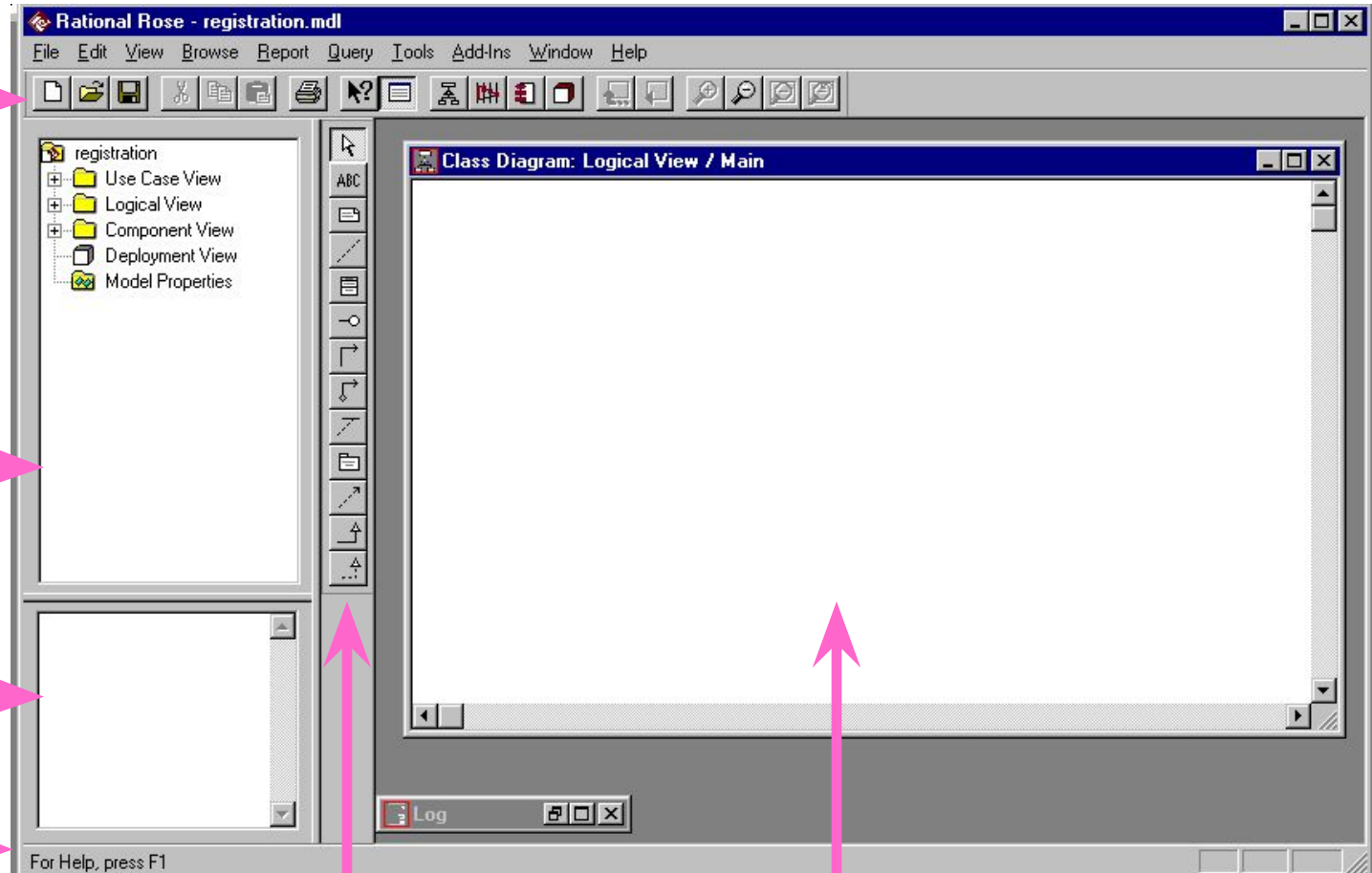
Diagram Toolbar

Diagram Window

24

About the next slide…

- Most things in your model (classes, use cases, actors, etc) have all manner of attributes and parameters. You edit these via the "Specification" dialogue associated with each.

- To get the specifications, right-click a thing in the browser or a diagram and choose "Specification".

Specifications

- Most modeling elements have a Specification that contains additional information about the modeling element

Rational
unifying **software** teams
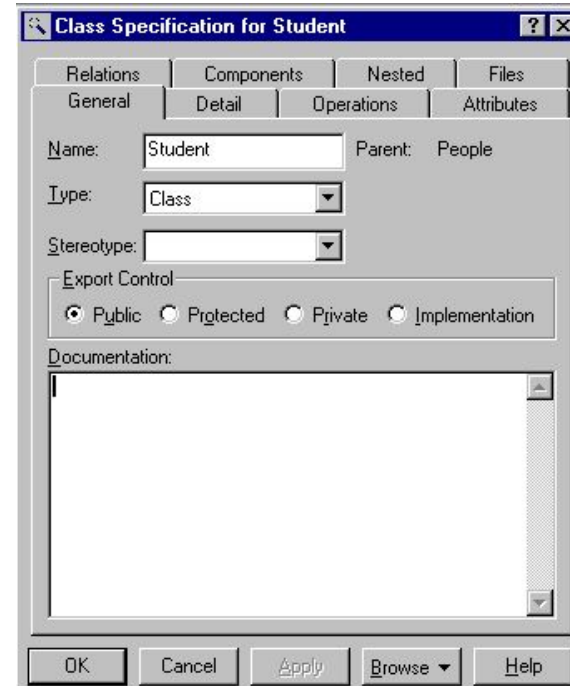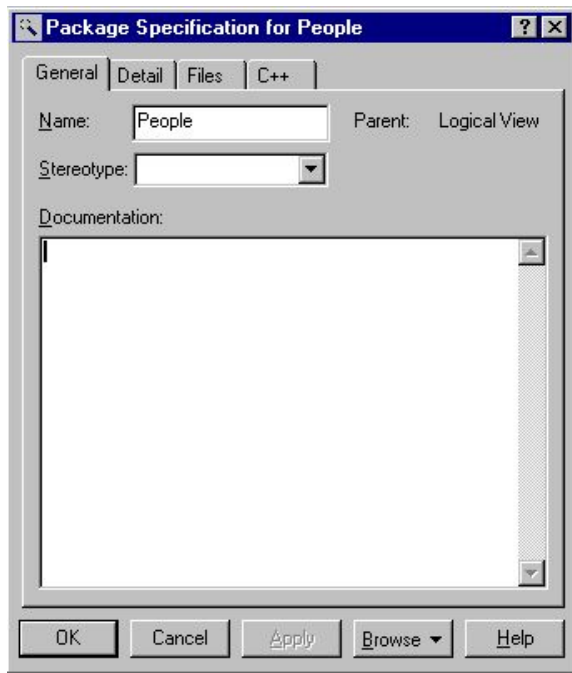
About the next slide…

- The next several slides refer to use cases, a particular type of diagram
- The next slide shows the "Use Case View" section of the browser. Any actors, use cases and use case diagrams each get an entry. "Associations", ie arrows, are grouped together.
- Use the browser to add elements to your model, then draw a picture to show how they go together.

Use Case Diagrams
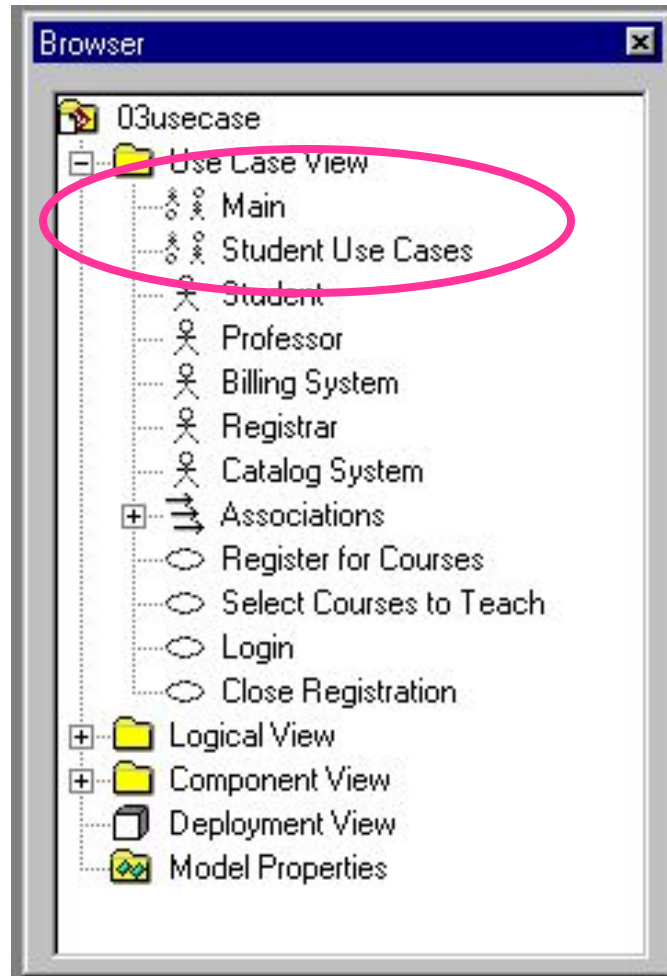
About the next slide…

- The next slide shows a full use case diagram.
- The stick figures denote actors, and the ovals are use cases (a function or behavior or interface your software provides).
- The arrows indicate 'use' or dependency. For example, the "Student" uses the function "Register for Courses", which in turn uses the external "Catalog System".
- The <<uses>> tokens attached to some of the associations (arrows) are *stereotypes*, an indication of what the association means. In this diagram, <<uses>> indicates that the association means a direct software link, ie, that the function "Register for Courses" will directly use the function "Login". This is different than the unmarked arrows, which indicate "use" in the vague sense of manipulating or interacting.

Use Case Relationships
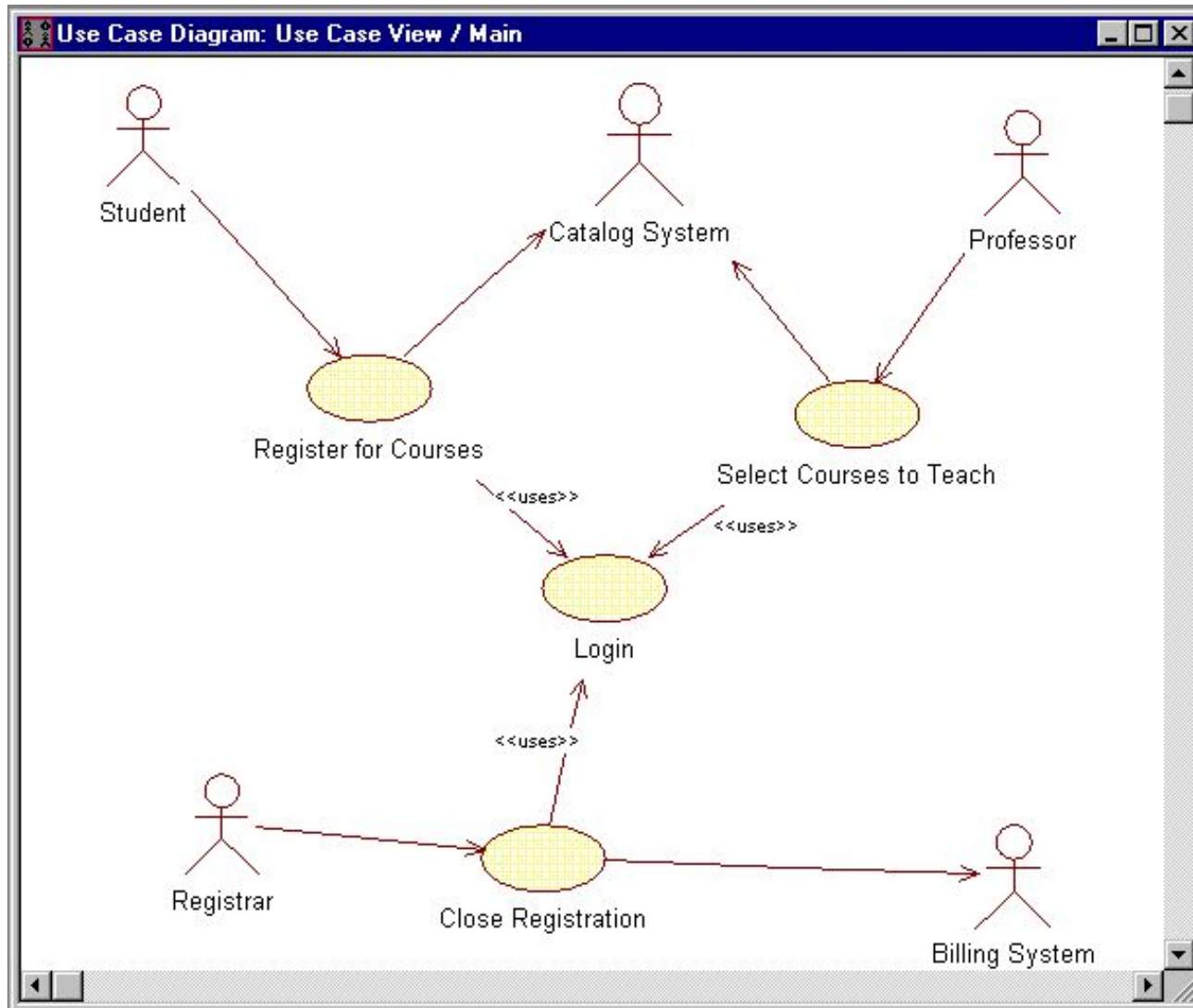
About the next slide…

- The next slide shows how documentation (notes, etc) can be added to a particular element.
- Here, they're adding the documentation via the Specification dialogue.

Brief Description -- Register for Courses

Rational
unifying software teams
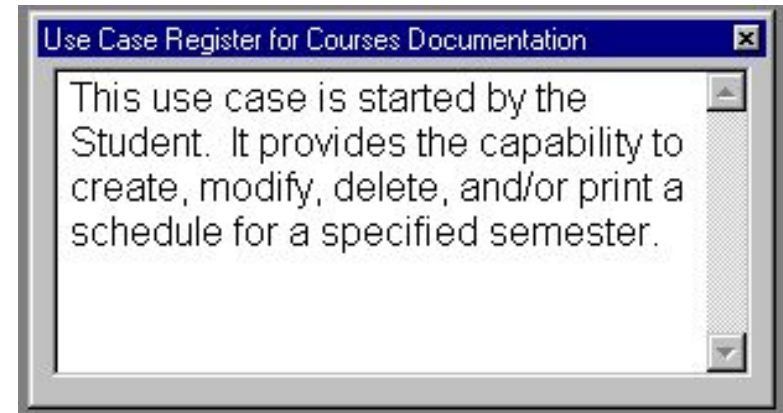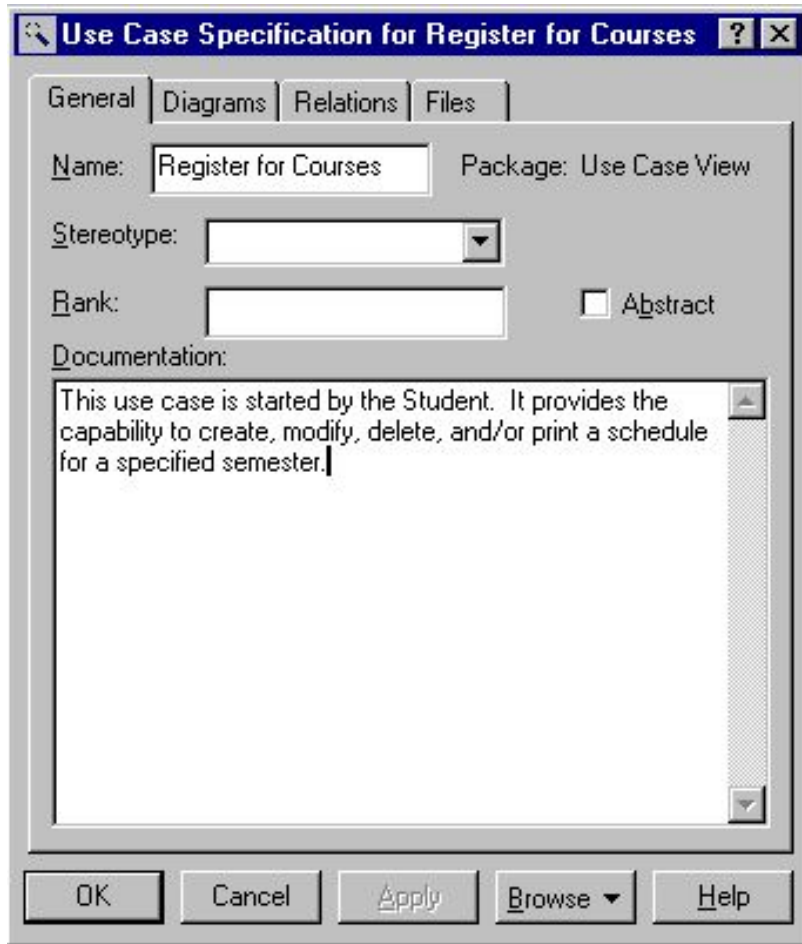
UML in your documents

- UML is intended to ease the task of communicating software designs.
- Typical uses of UML in G64HLL:
  - conceptual component diagrams in the concept document.
  - use cases and class diagrams in the requirements document.
  - class, sequence, state, package and deployment diagrams in the architecture document.

UML pitfalls, 1

- UML is a language, with a (reasonably) rigorous syntax and accepted semantics; that is, the diagrams have a meaning. Thus you have to be careful that the meaning of your diagram is what you intended.
- However, the semantics of UML are less well-defined than a programming language (where the semantics are defined by the compiler). Thus there is some leeway to use UML your own way: but you must be consistent in what you mean by the things you draw.

UML pitfalls, 2

- Arrow happiness: people tend to draw arrows (associations) everywhere in their diagrams, inconsistently without much regard for the UML meaning of a given arrow.

- Diagram fever: it's easy to do too many diagrams. The trick is to get the correct granularity. Eg, the requirements document should leave implementation detail to the architecture.

- General loopiness: be careful about slapping together UML diagrams, or doing a diagram without thoroughly understanding your system. You should always be able to give a clear and concise explanation of your diagram, and why you did it that way.

Exporting Rational diagrams?

- You can use screen captures
- Find out other ways to do this …

Learning Rational …

- Lots of practice
- Read online tutorial

Citation

These slides borrow heavily:

Martin Fowler, *UML Distilled*

Addison-Wesley, 1997

You can also get the material from:

Sinan Si Alhir, *UML In A Nutshell*

O'Reilly, 1998

Back to UML

Class Diagrams

UML diagrams: class diagram

- Motivated by Object-Oriented design and programming (OOD, OOP).

- A class diagram partitions the system into areas of responsibility (classes), and shows "associations" (dependencies) between them.

- Attributes (data), operations (methods), constraints, part-of (navigability) and type-of (inheritance) relationships, access, and cardinality (1 to many) may all be noted.

Class diagram "perspective"

- Class diagrams can make sense at three distinct levels, or perspectives:

  - Conceptual: the diagram represents the concepts in the project domain. That is, it is a partitioning of the relevant roles and responsibilities in the domain.

  - Specification: shows interfaces between components in the software. Interfaces are independent of implementation.

  - Implementation: shows classes that correspond directly to computer code (often Java or C++ classes). Serves as a blueprint for an actual realization of the software in code.

41

Class diagram examples

(A classroom scheduling system: specification perspective.)
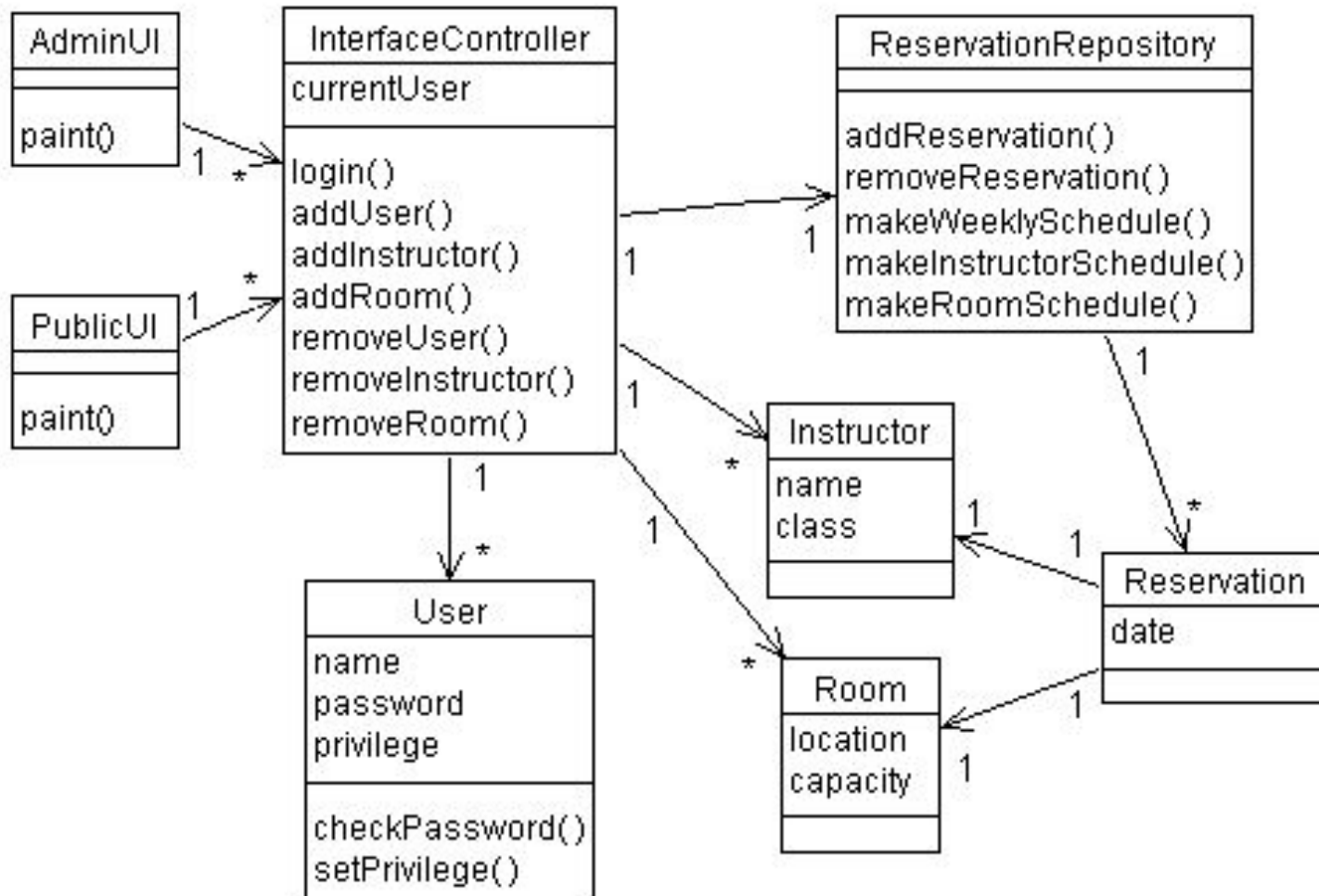
About the last example...

- Each box is a class, with necessary attributes and operations specified.

- Navigability arrows show which classes can reference which others.

- Cardinality marked in bi-directional manner on arrows.

- The classes together represent the complete system; thus the the classes are a *partitioning* of the system.

Rational Software

Class Diagrams

About the next 4 slides…

- The next several slides deal with classes and class diagrams.
- The next two slides show classes and packages in the browser. A package contains some classes.
- The following two slides show adding attributes (a class's data; "operations" are a classes methods) to a class from the Specification dialogue, and from the browser directly.
- As you'd expect, the menus pop up when click the right mouse button.

45

Classes in the Browser
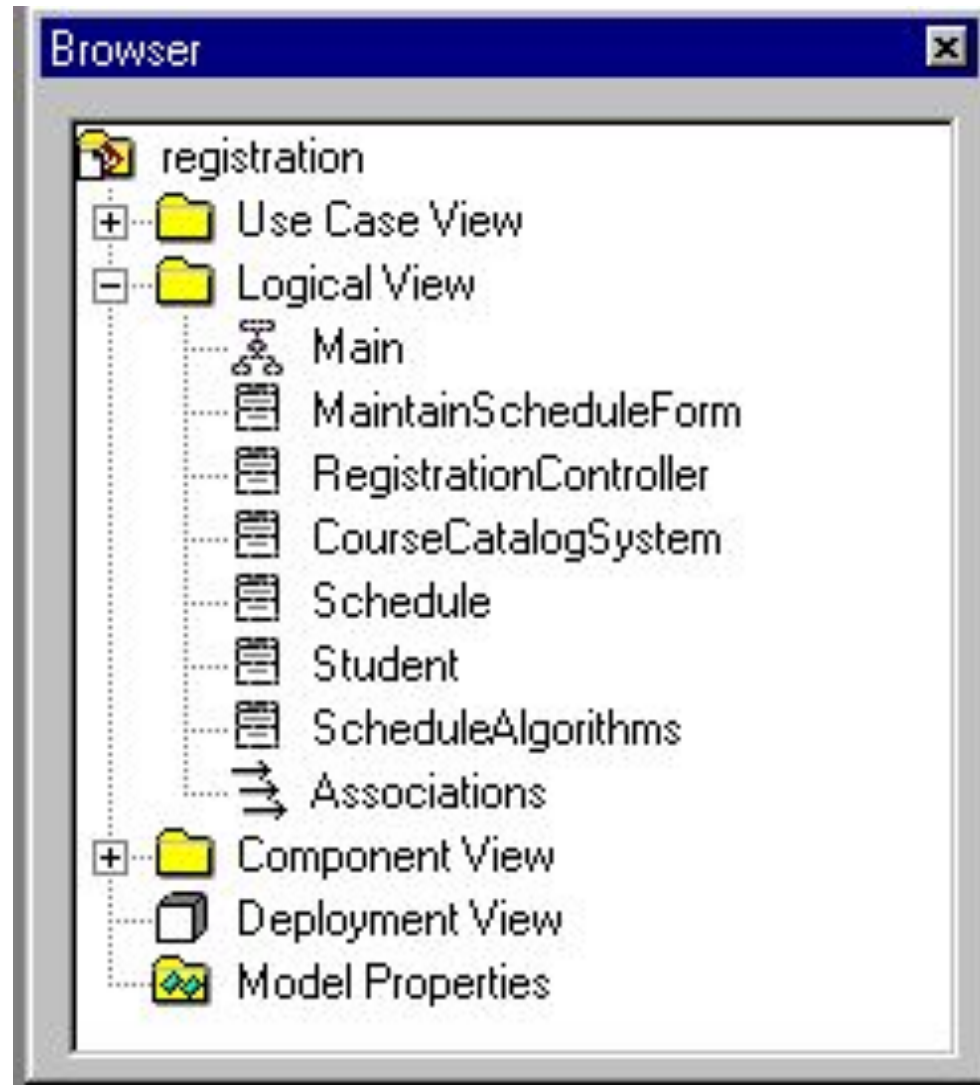
Packages in the Browser
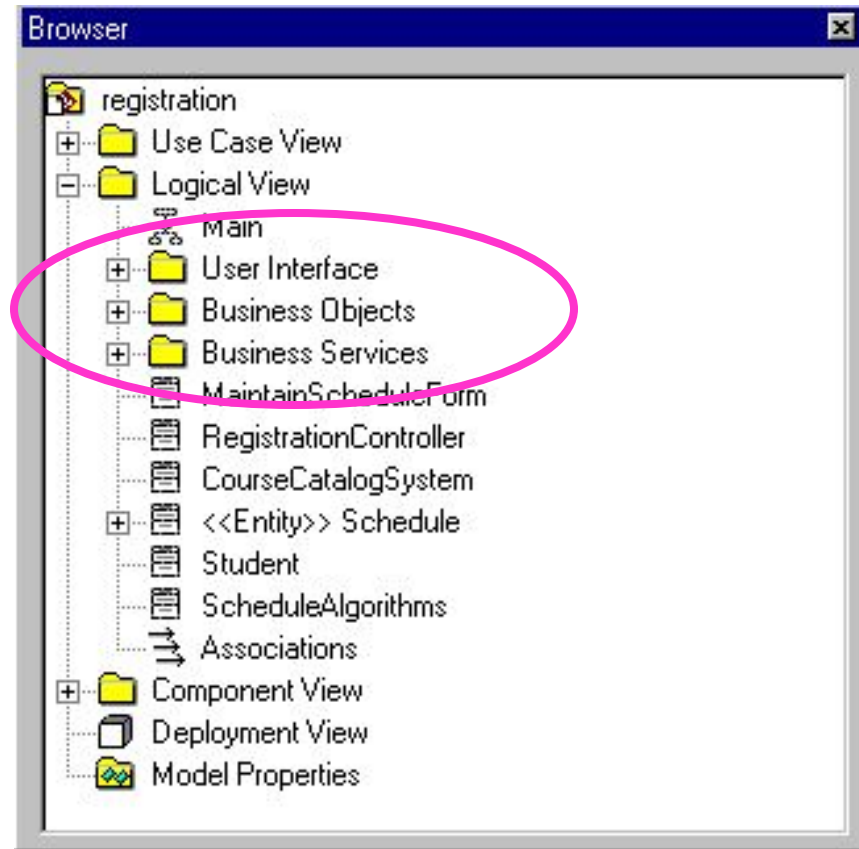
Using the Class Specification - Attributes
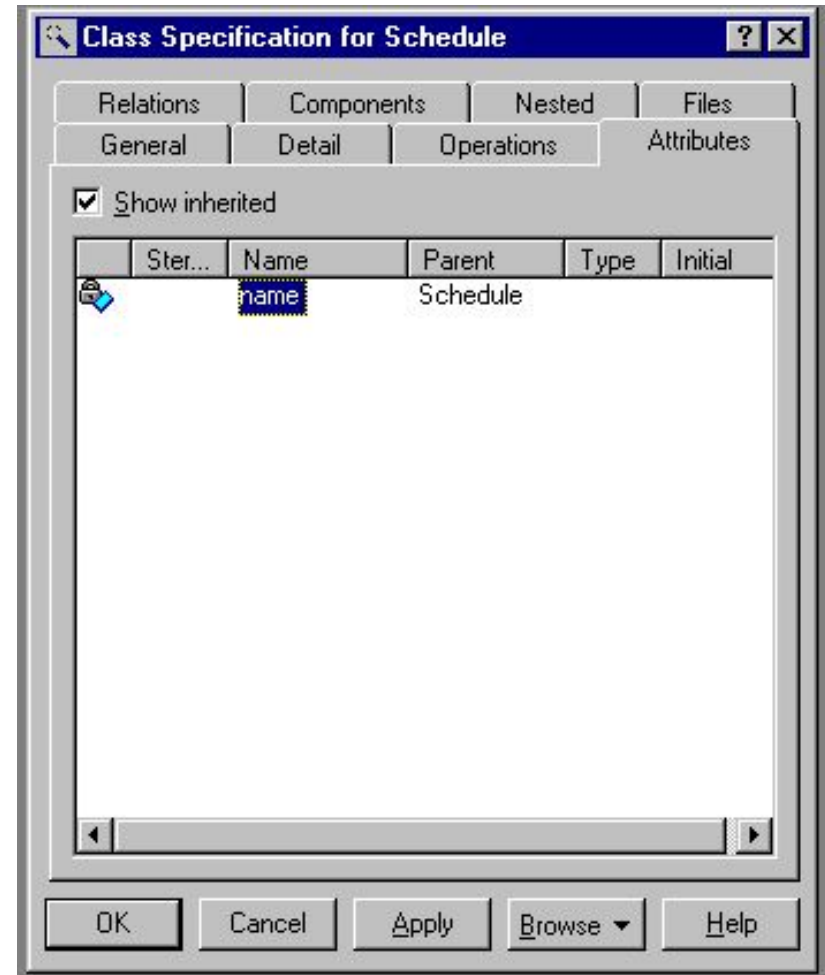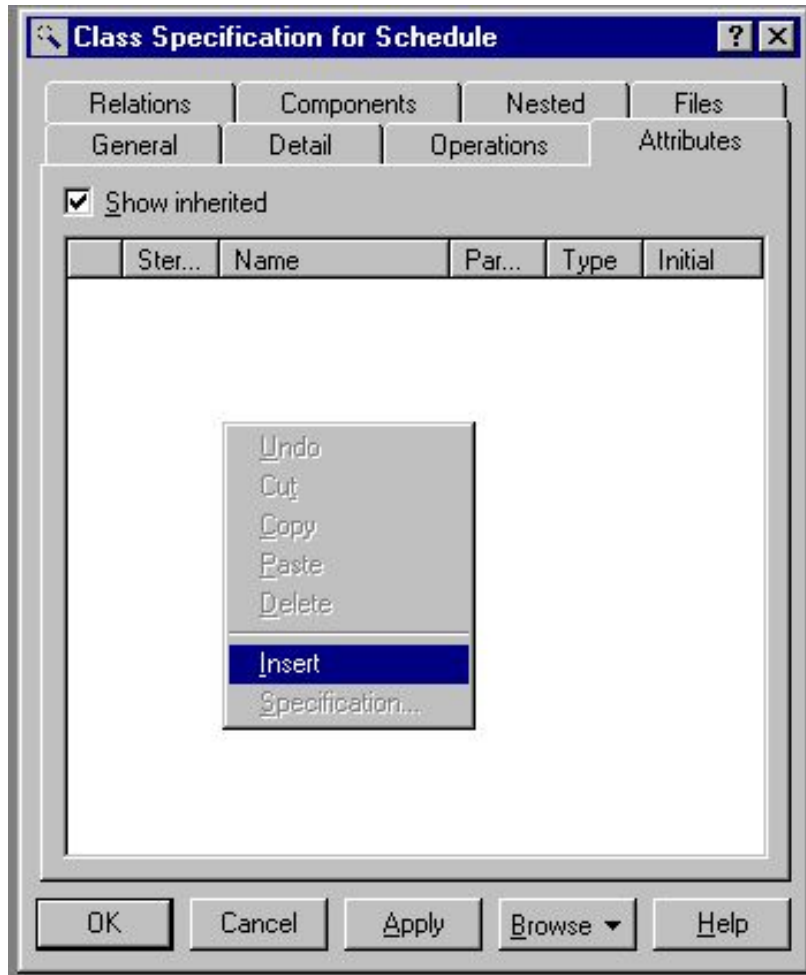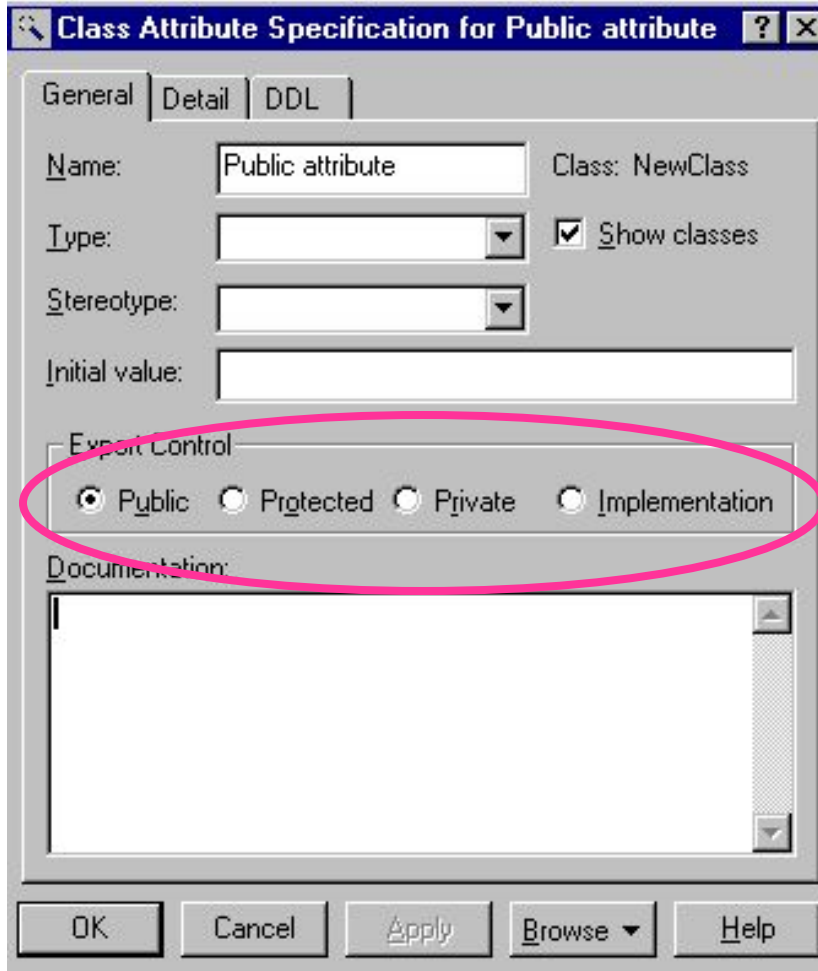
Attributes and Operations and the Browser

Rational
unifying software teams
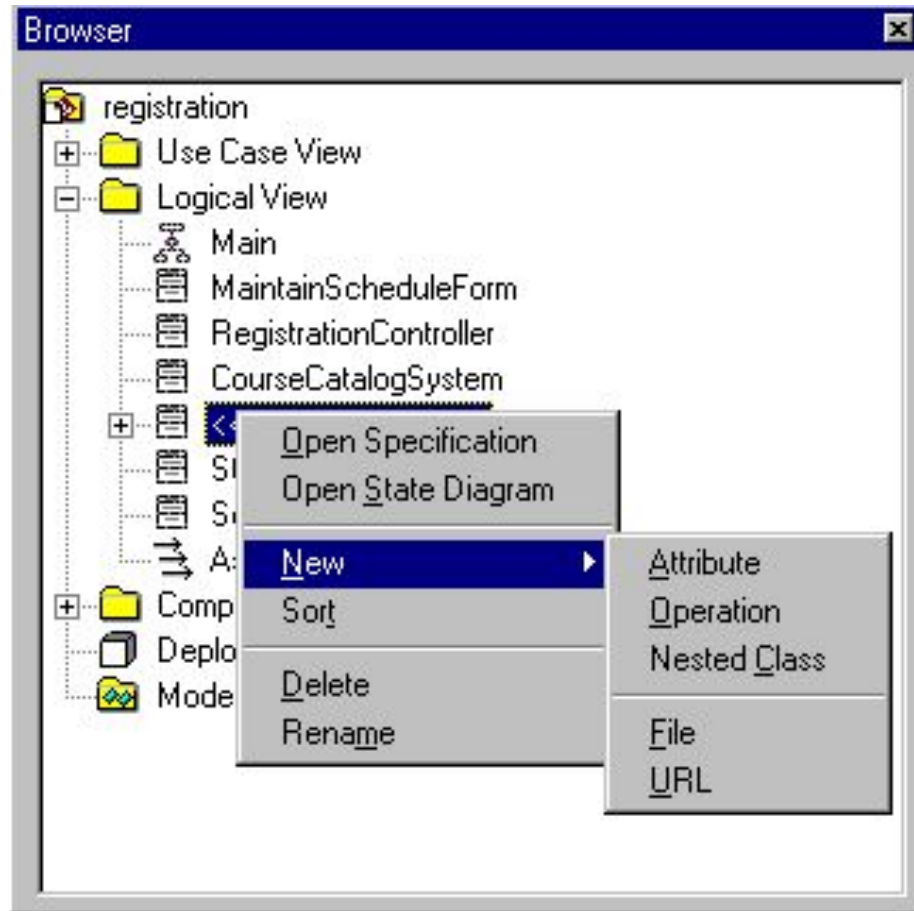
About the next slide…

- The next slide shows how to specify the visibility of class attributes in the model.
- The visibilities correspond to the notions of visibility in Java (public, private, protected, etc).

Attribute Visibility Options

NewClass

◆Public attribute
🔑◆Protected attribute
🔒◆Private attribute

NewClass

+ Public attribute
# Protected attribute
- Private attribute

Rational
unifying software teams

About the next 2 slides…

- The next slide shows the icons for packages and classes in a class diagram.
- The third icon is a class, marked with a stereotype. Here the stereotype indicates a type of class, ie that it is an "interface". This doesn't necessarily mean that the class is a Java-type interface (but that's probably what they mean).
- The following slide shows the types of associations (arrows) Rose allows in a class diagram. They correspond to constructs in OO design and programming.
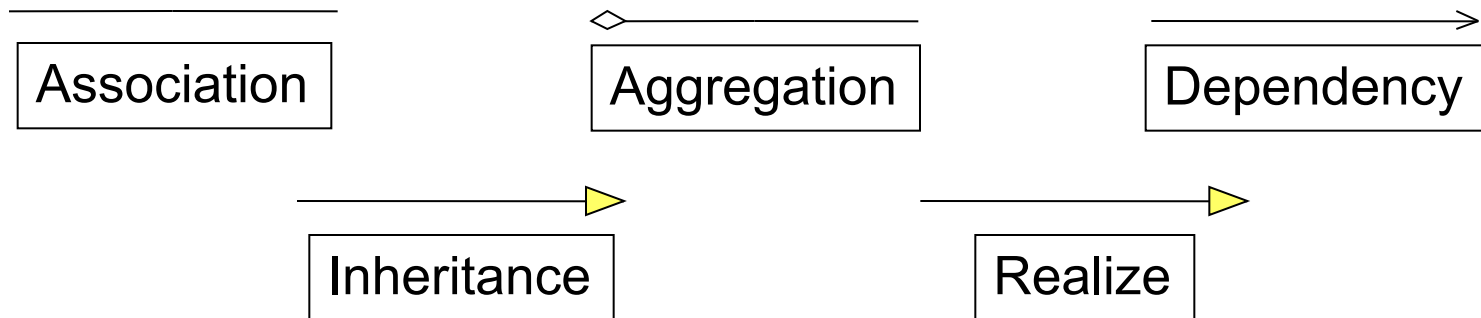
What is a Class Diagram?

- A class diagram is a view of the static structure of a system
  - Models contain many class diagrams
- Class diagrams contain:
  - Packages, classes, interfaces, and relationships
- Notation:

| Package Name |
|---|

Class
Name

<<Interface>>
Interface
Name
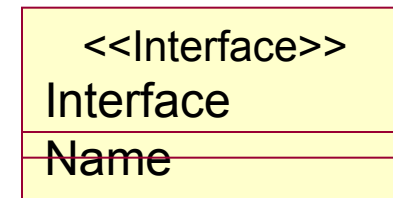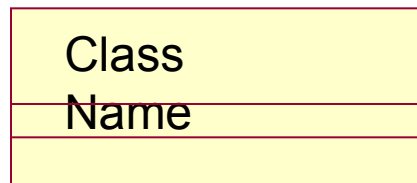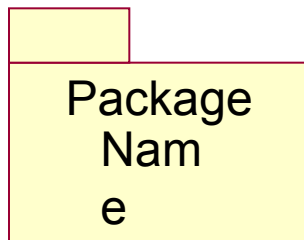
Relationships

- Class diagrams may contain the following relationships:

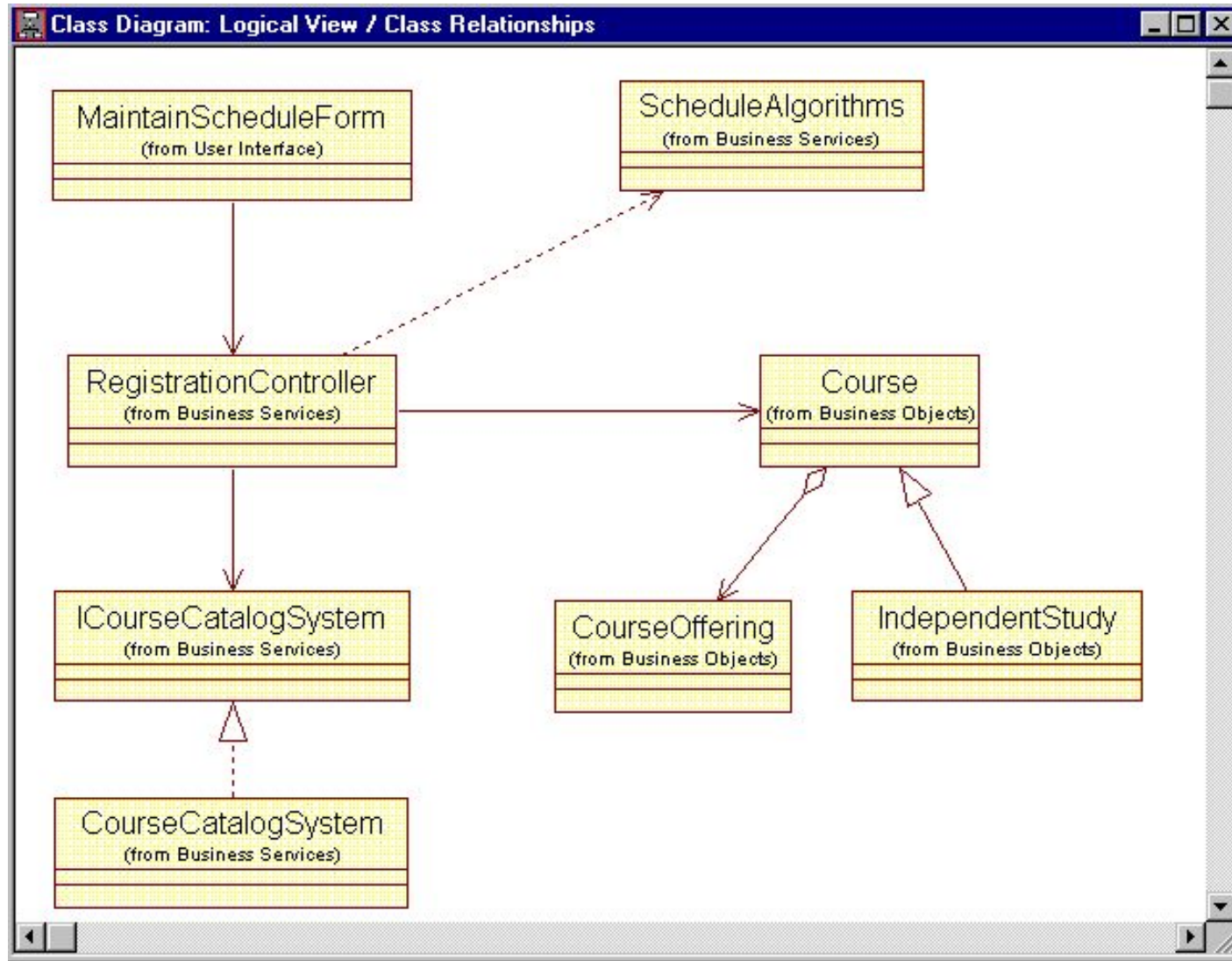 - Association, aggregation, dependency, realize, and inheritance

- Notation:

Rational
unifying software teams

About the next 2 slides…

- The next slide shows a package diagram, with dependencies.
- The following slide shows a class diagram, with various associations between the classes.

Package Relationships
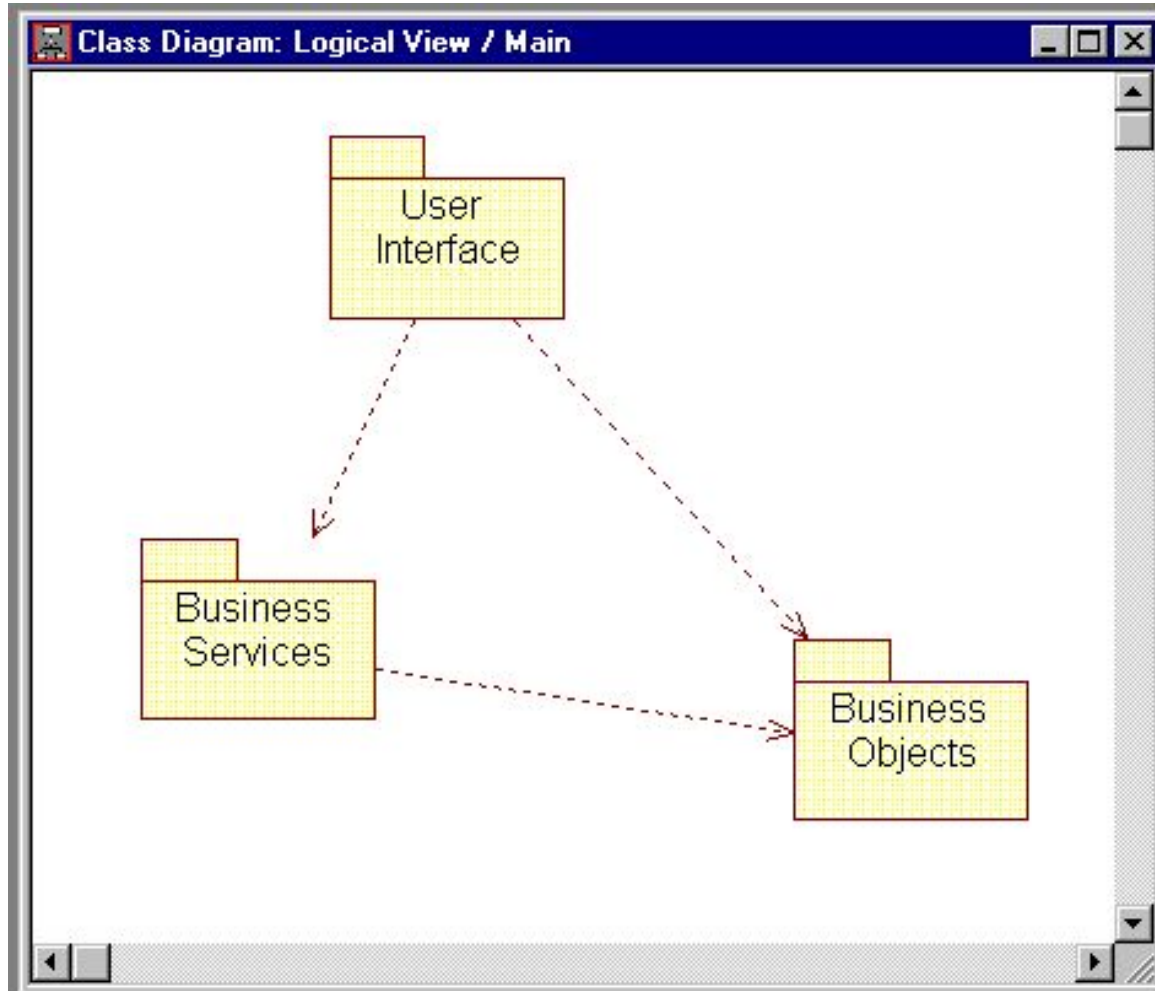
Class Relationships

About the next 2 slides…

- The next slide shows how cardinalities are denoted in Rose.

- The following slide is the class diagram example from before, but this time with cardinalities marked on the associations.

Multiplicity Indicators

- Each end of an association or aggregation contains a multiplicity indicator
  - Indicates the number of objects participating in the relationship

——————— 1	**Exactly one**
——————— 0..*	**Zero or more**
——————— 1..*	**One or more**
——————— 0..1	**Zero or one**
——————— 2..7	**Specified range**

Rational
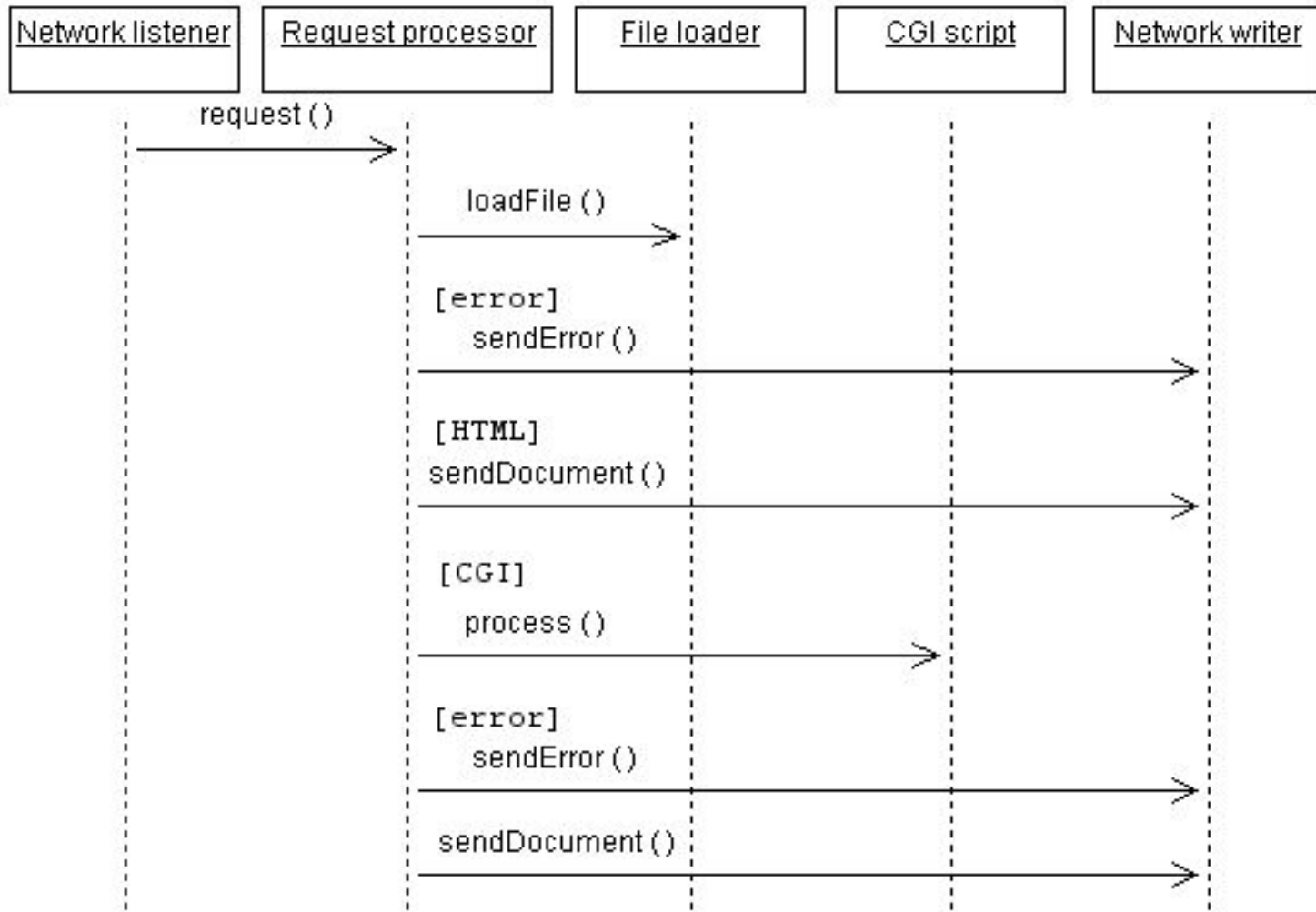unifying software teams

Multiplicity Indicators
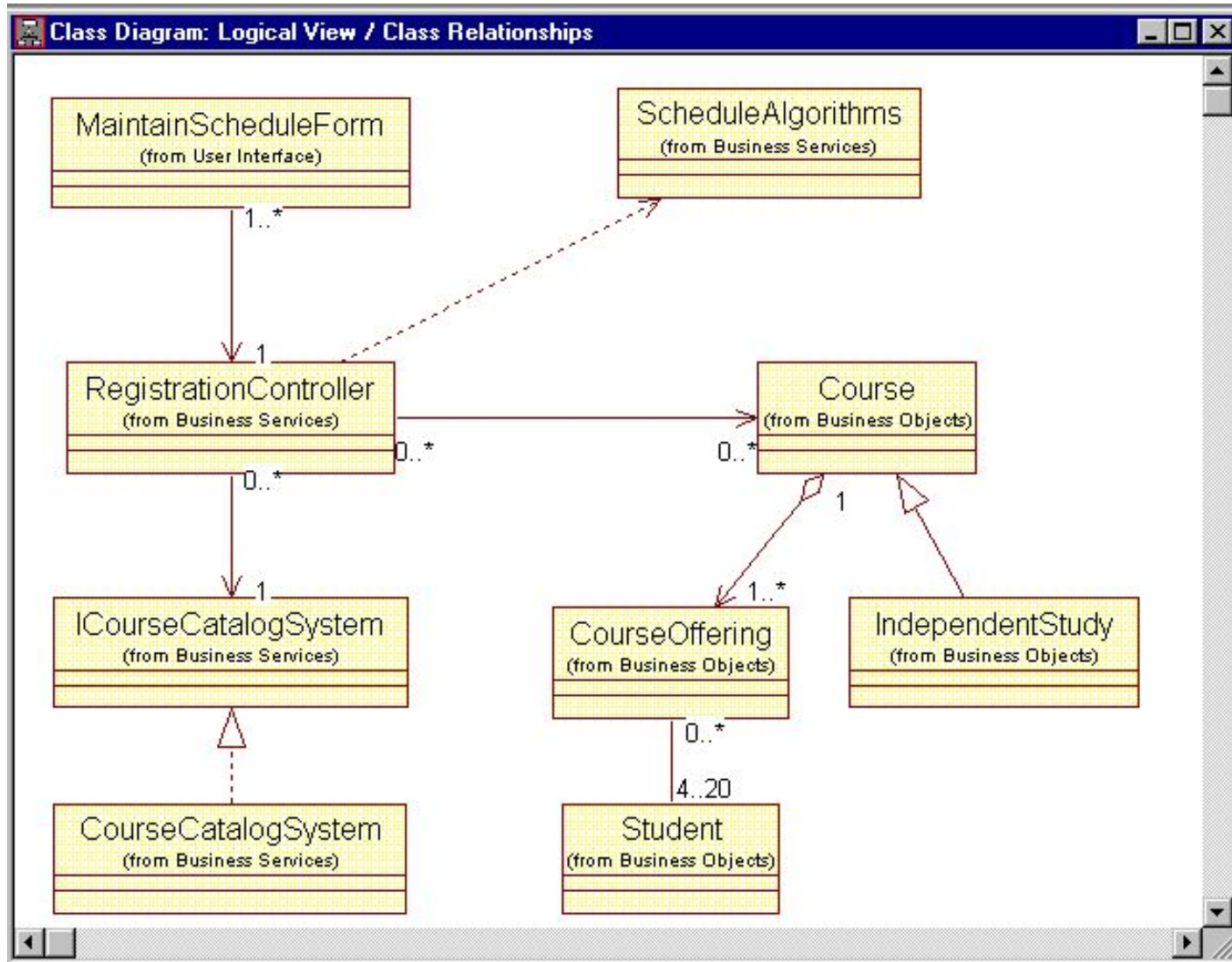
Sequence Diagrams

UML diagrams: sequence diagram

- Sequence diagram describe algorithms, though usually at a high level: the operations in a useful sequence diagram specify the "message passing" (method invocation) between objects (classes, roles) in the system.

- The notation is based on each object's life span, with message passing marked in time-order between the objects. Iteration and conditional operations may be specified.

- May in principle be used at the same three levels as class diagrams, though the specification level will usually be most useful. (At the implementation level, you might better use pseudocode.)

Sequence diagram example

Network listener	Request processor	File loader	CGI script	Network writer

request ()

loadFile ()

[error]
  sendError ()

[HTML]
sendDocument ()

[CGI]
  process ()

[error]
  sendError ()

sendDocument ()

About the last example...

- Each box with connected line represents a distinct thing, where all the things aren't necessarily in the same piece of software, or software at all.

- Arrows indicate *message passing*. That is, an arrow indicates that one thing tells another thing to do something.

- Reverse arrows are implied. If arrow goes from A to B, and then immediately afterward an arrow goes from A to something else, it is understood that B completed it's operation and returned control (and a result, probably) to A.

- Time runs down the page. An comes before an arrow that is below it.

- Bracketed expressions indicate conditions. In the diagram, an error document is returned if the fileLoad() operation returns and error.

64

About the next 3 slides…

- The next several slides are about sequence diagrams (for algorithms, processes).

- The next slide shows how to create a sequence diagram in browser, by associating it with a use case.

- The following slide shows some "objects" in a sequence diagram, and the slide after shows how to associated an object with a class. Objects are a bit more general than classes, but you'll get the best results if you create a one-to-one association between the objects in your sequence diagrams and the classes you've defined (define your classes first, if you can!).
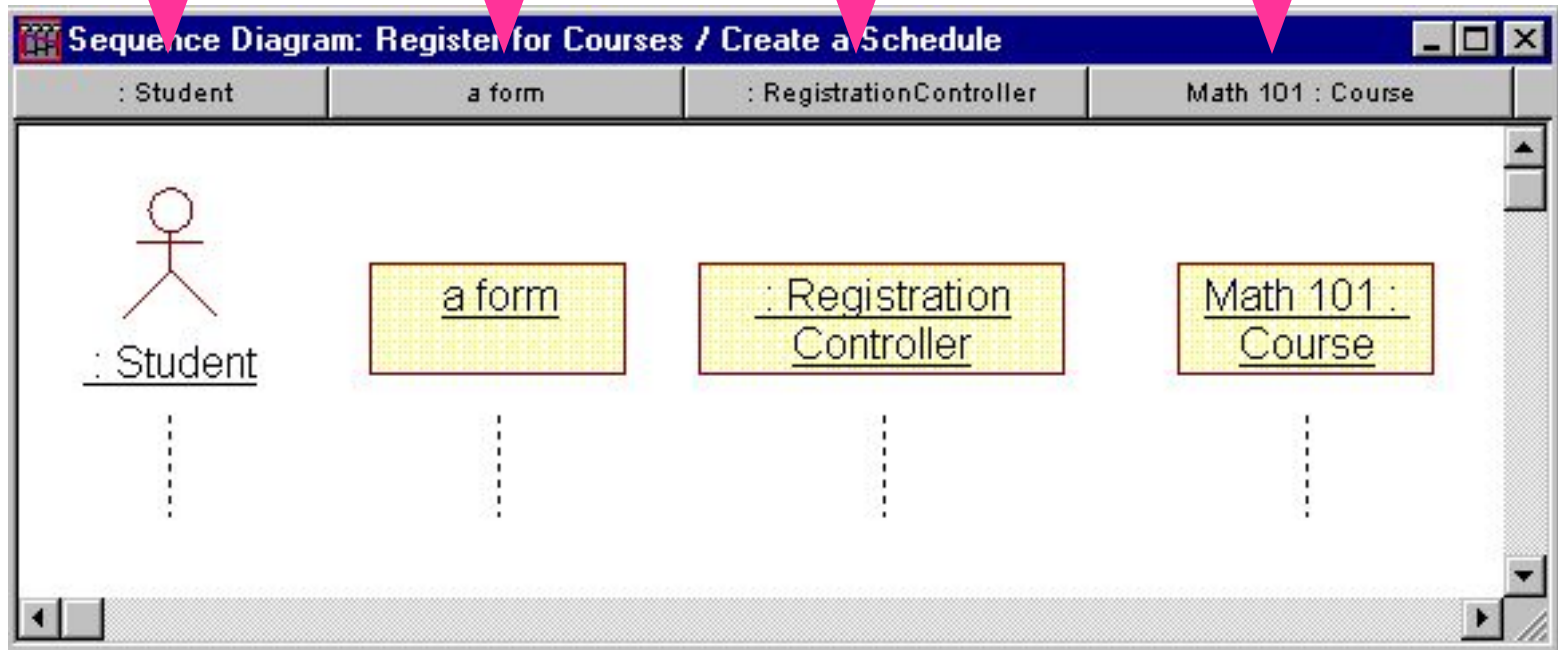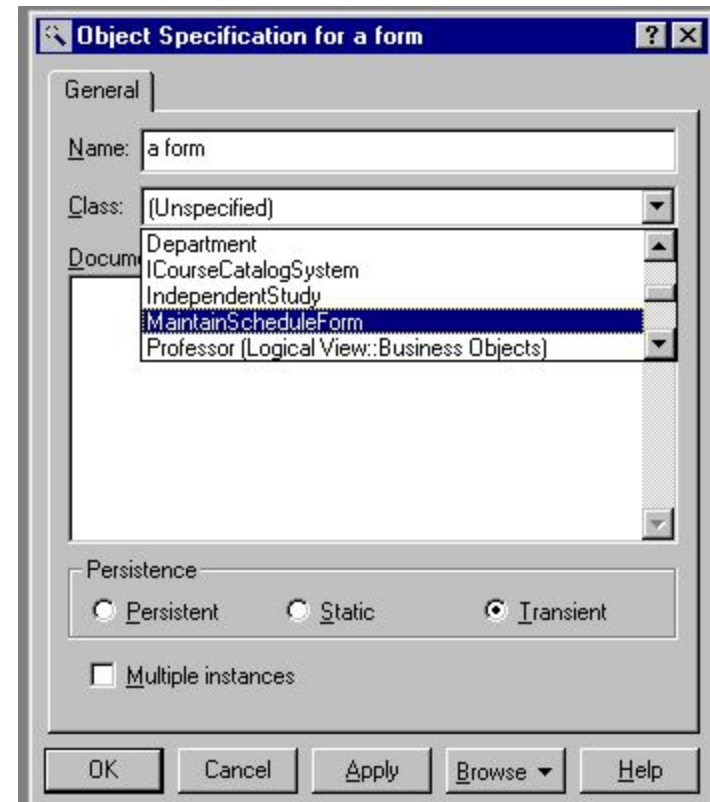
Creating a Sequence Diagram

Rati⊙nal
unifying **software** teams
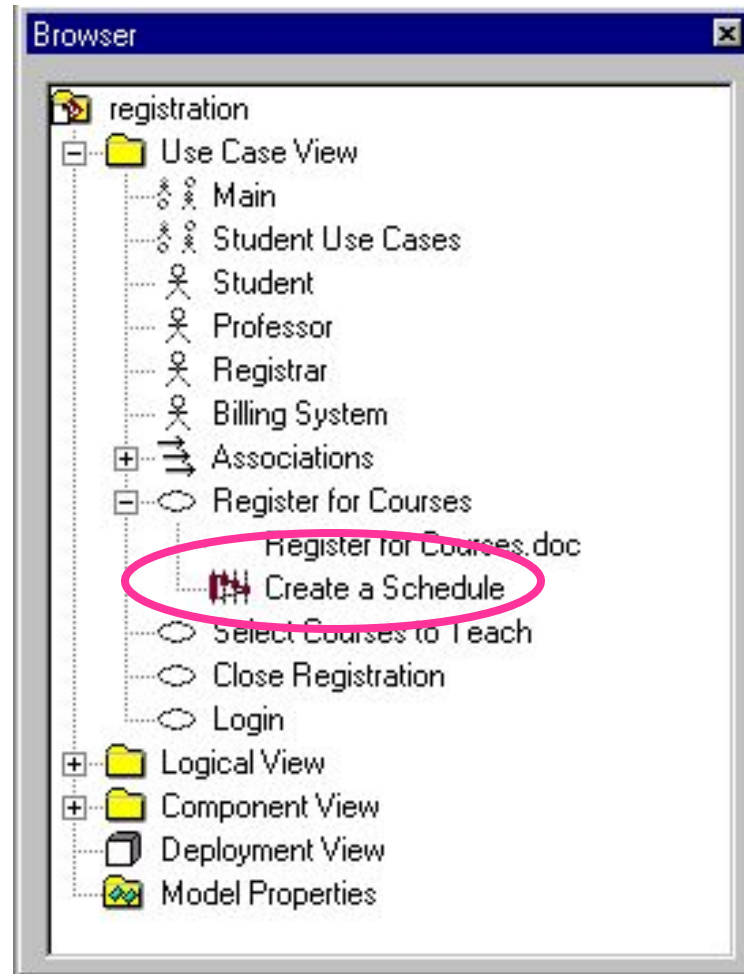
Representing Objects
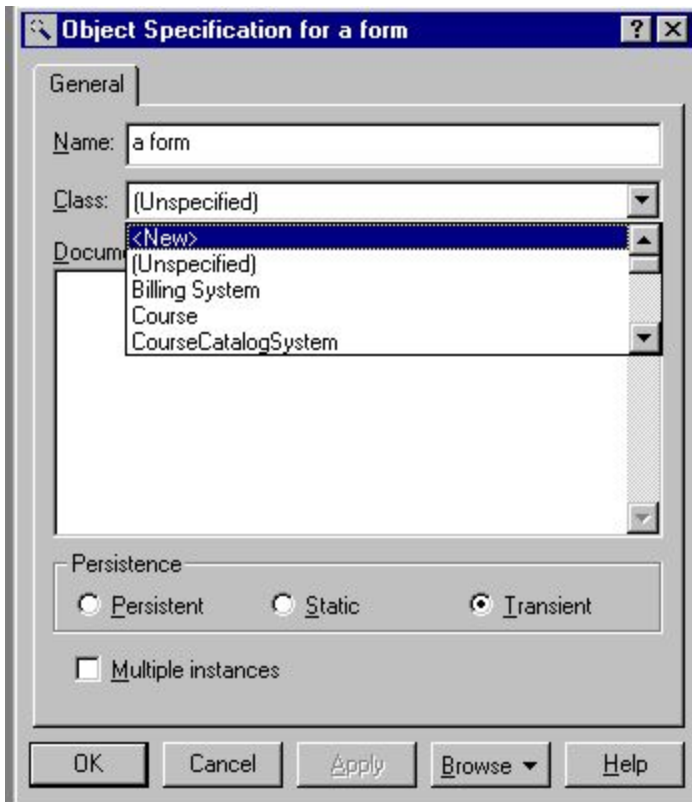
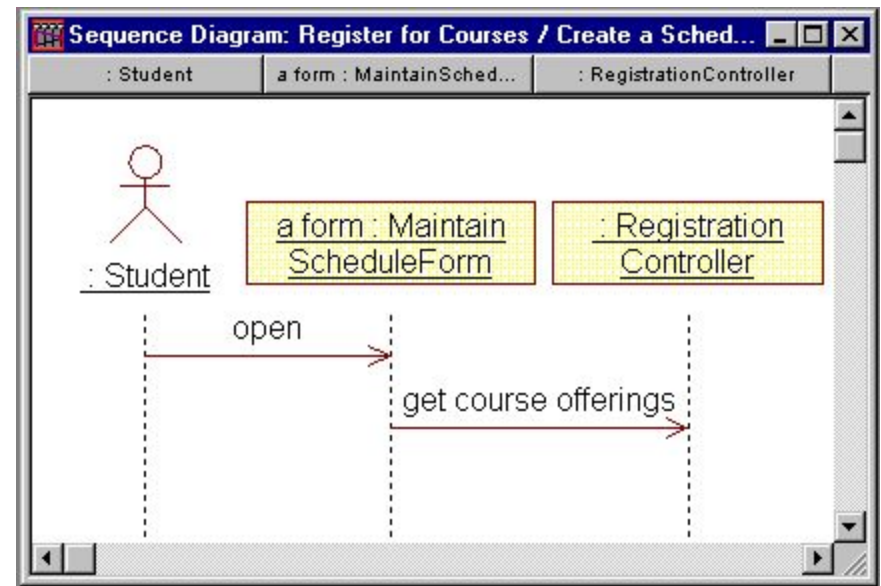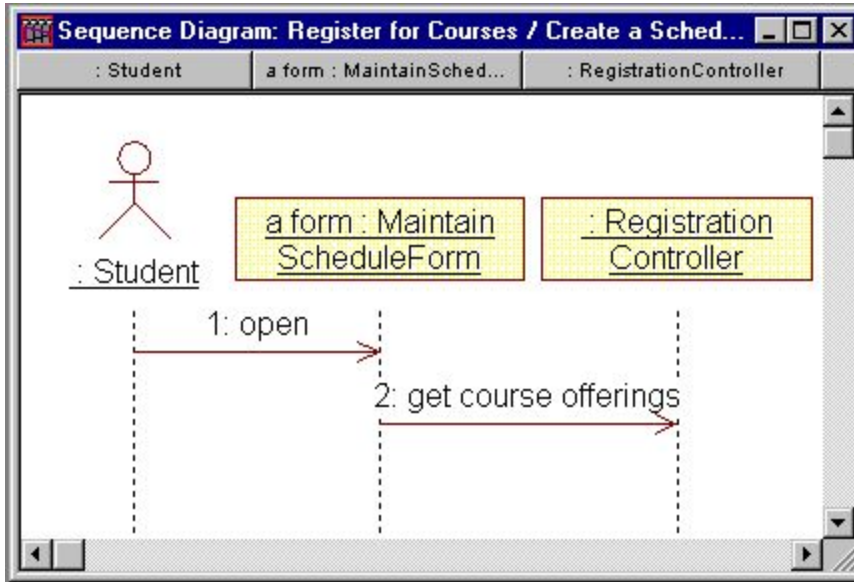# Assigning Objects to Classes

- A new class can be created for the object
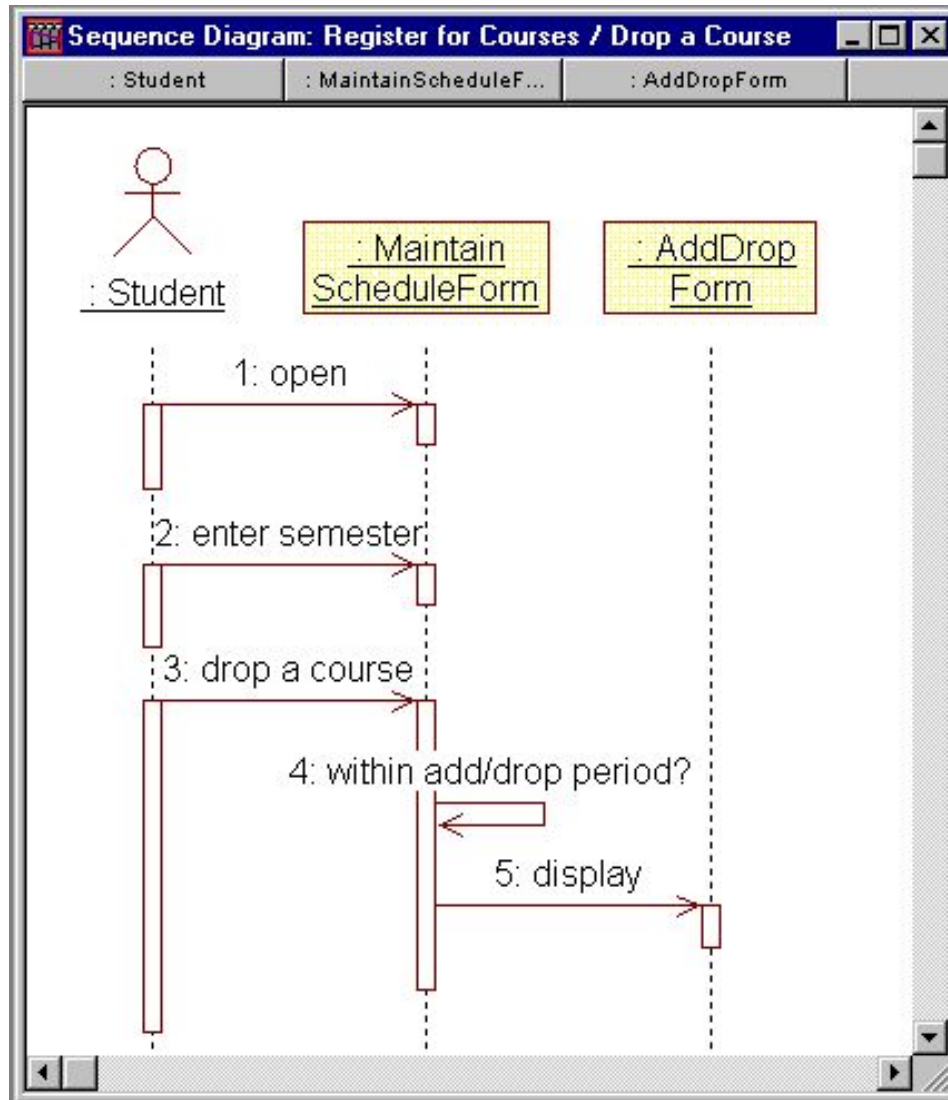
- An object can be assigned to a class already defined

About the next 3 slides…

- The next slide shows how to denote message passing in a sequence diagram. To pass a message is usually to call a method on an object.

- The following slide shows a notation for "focus of control". This means that an object in control when there is a box around its lifeline. The example indicates that "Student" maintains control throughout "drop a course", even while "Maintain schedule form" does its thing. Among other things, this can be used to imply that called methods terminate and return.

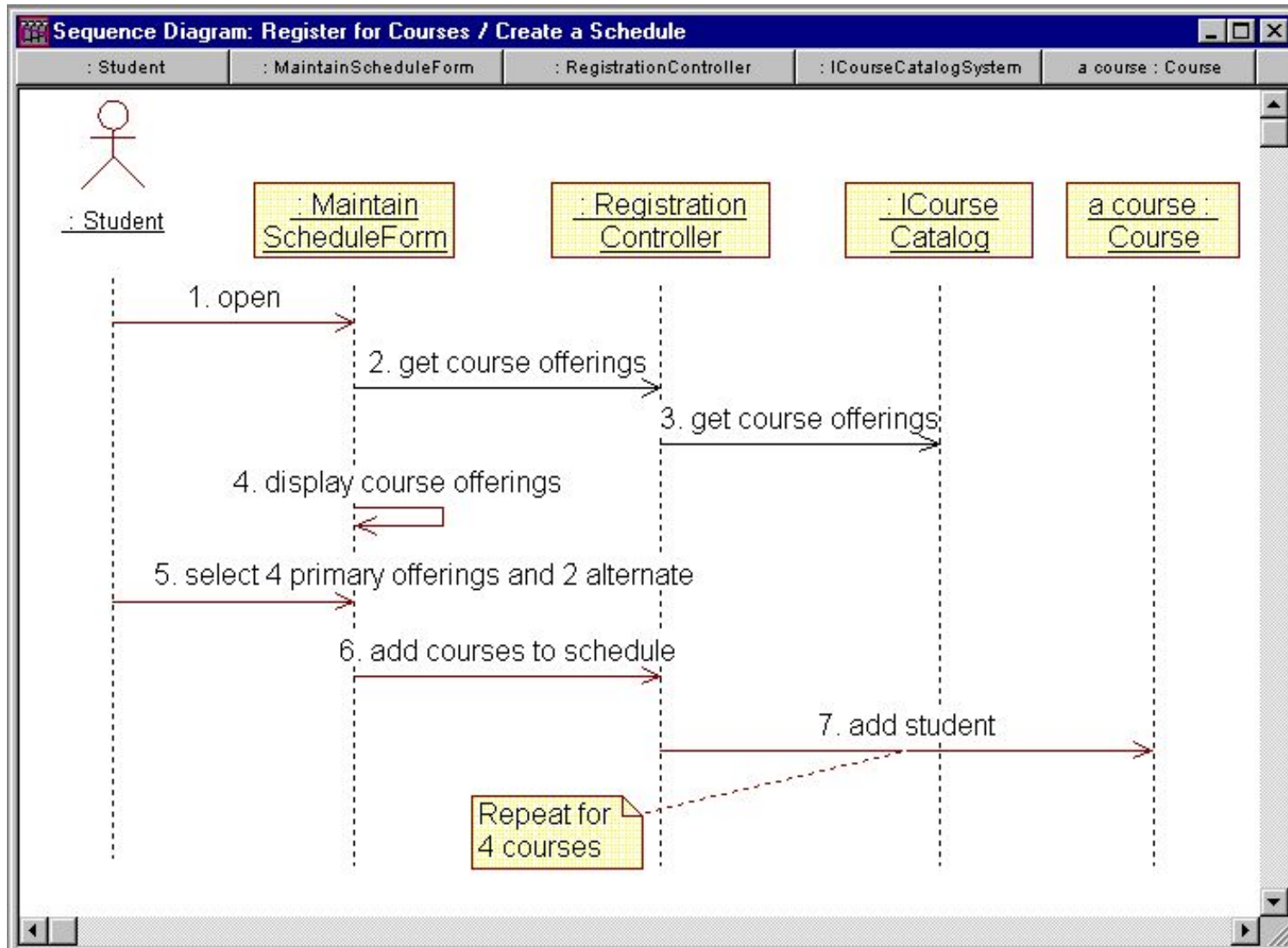- The third slide shows a full sequence diagram example.

69

Messages

Rati**o**nal
unifying **software** teams

Focus of Control

71

Exercise: Sequence Diagram

Sequence Diagram: Register for Courses / Create a Schedule

: Student | : MaintainScheduleForm | : RegistrationController | : ICourseCatalogSystem | a course : Course

- 1. open
- 2. get course offerings
- 3. get course offerings
- 4. display course offerings
- 5. select 4 primary offerings and 2 alternate
- 6. add courses to schedule
- 7. add student

Repeat for 4 courses

72

Other Diagrams

UML diagrams: Package diagram

- A type of class diagram, package diagrams show dependencies between high-level system component.

- A "package" is usually a collection of related classes, and will usually be specified by it's own class diagram.

- The software in two distinct packages is separate; packages only interact through well-defined interfaces, there is no direct sharing of data or code.

- Not all packages in a system's package diagram are new software; many packages (components) in a complex system are often already available as existing or off-the-shelf software.

Package diagram example
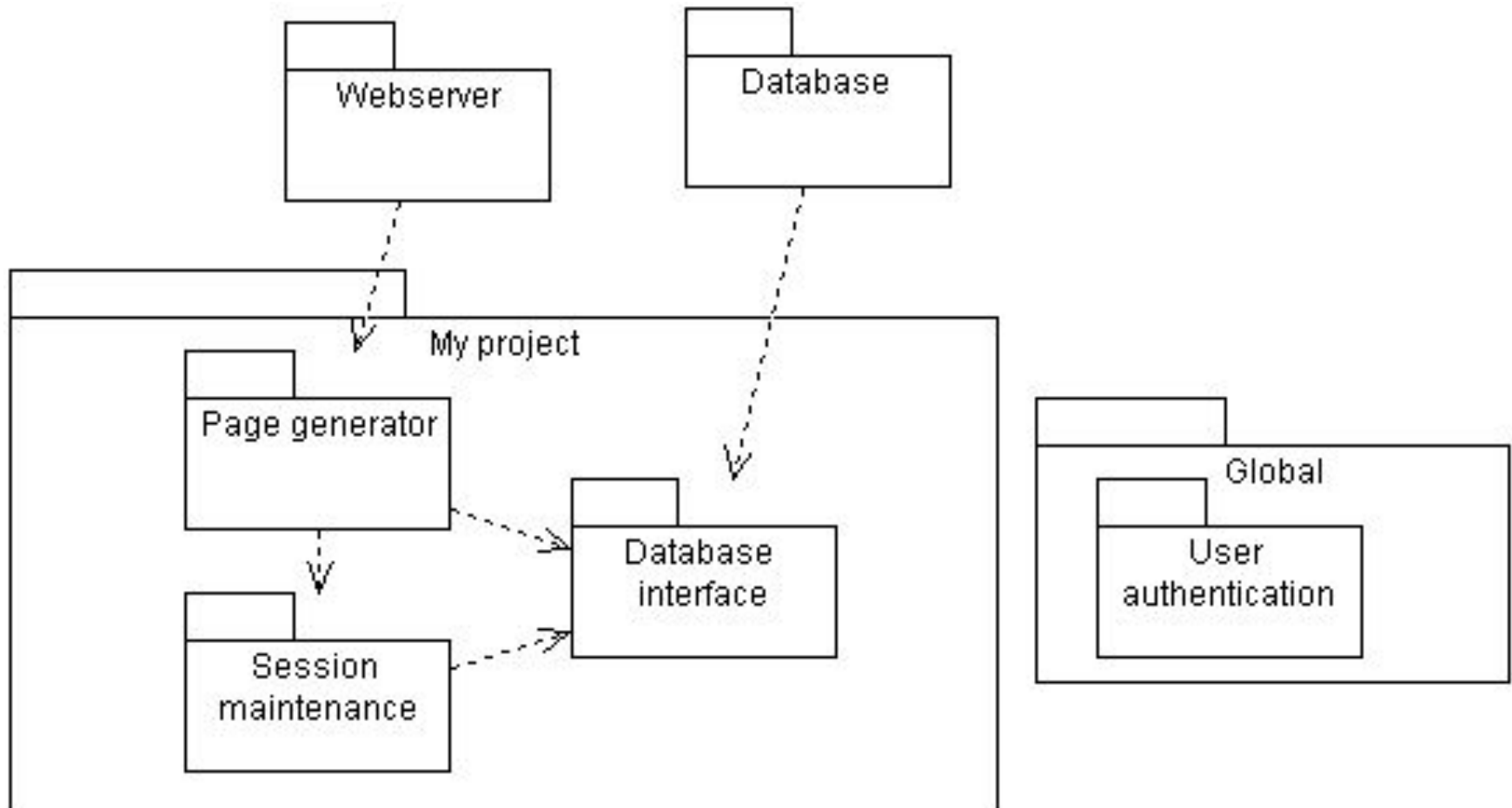
About the last example...

- This package diagram indicates that:
  - there are three dependent but decoupled software components that will be developed in "My Project", which is itself a package or component.
  - Parts of my software depend on some existing software packages, which I won't be developing, but just using ("Webserver" and "Database").
  - There is a globally available package "User authentication" which all the other packages depend on.

About the next slide…

- The next slide shows a complete *deployment diagram*.

- A deployment diagram is useful for showing how your software will be deployed on hardware. It may show how your system will integrate with existing systems in the domain.

Exercise: Deployment Diagram
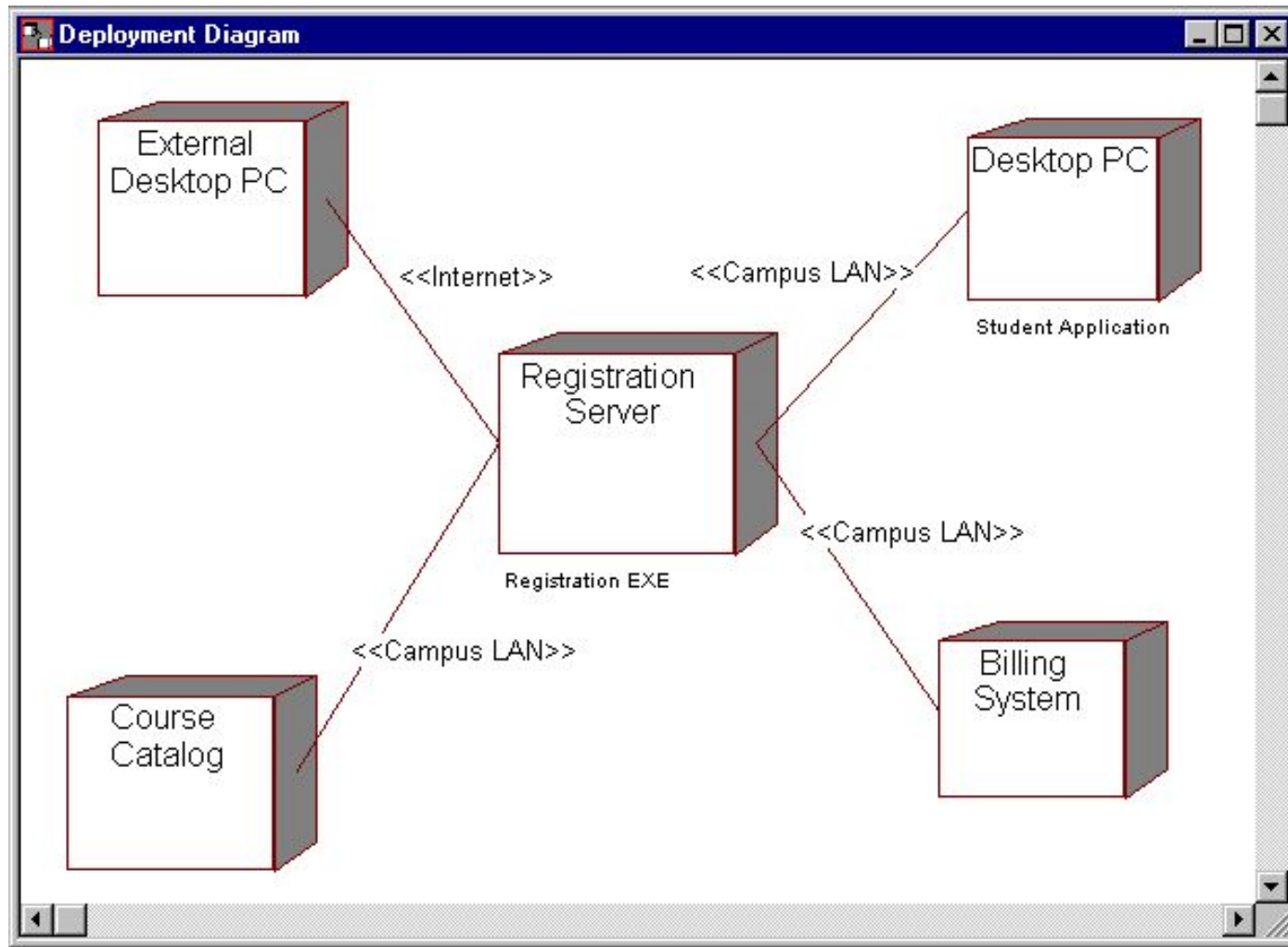
UML diagrams: other diagrams

- State diagrams: similar in function to sequence diagrams, but with focus on the prerequisites for an operation, rather than the exact sequence of actions.

- Deployment diagrams: shows the installation of software on hardware platforms.

- Others: activity diagrams, collaboration diagrams.

- Look in *UML Distilled* for examples.

UML diagrams: other diagrams

- UML Distilled: A Brief Guide to the Standard Object Modeling Language Martin Fowler, Kendall Scott

- IBM Rational

  http://www-306.ibm.com/software/rational/uml/