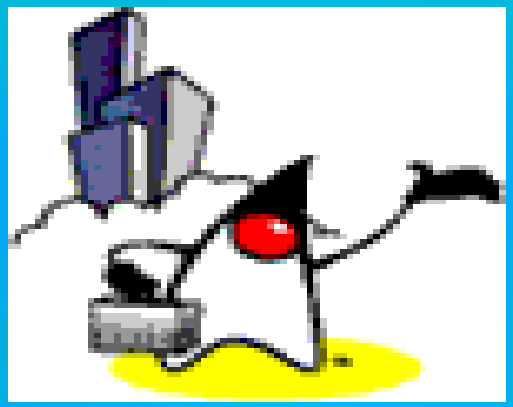
United Modeling Language (UML) Basics

Topics

- What is UML?
- What is Modeling?
- UML Diagrams
  - Use Case Diagram
  - Class Diagram
  - Activity Diagram
  - Package Diagram
  - State-Transition Diagram
  - Sequence Diagram
  - Collaboration Diagram
  - Component Diagram



What is UML?

Unified Modeling Language

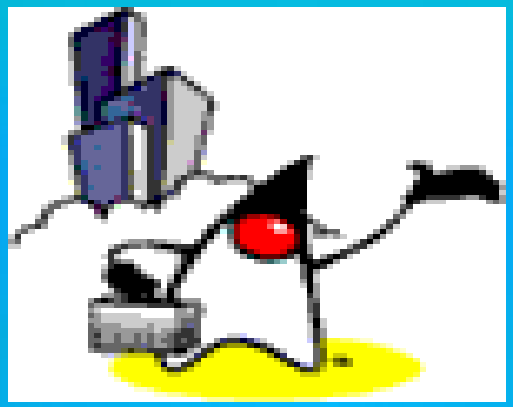
- The Unified Modeling Language (UML) is the standard language for specifying, visualizing, constructing, and documenting all the work products or artifacts of a software system.
- It unifies the notation of Booch, Rambaugh, and Jacobson, and augmented with other contributors once submitted to OMG.
- It proposes a standard for technical exchange of models and designs.

UML is NOT

- It is not a method or methodology.
- It does not indicate a particular process.
- It is not a programming language.

Terminology

UML	*Class*	*Association*	*Generalization*	*Aggregation*
Booch	Class	Uses	Inherits	Containing
Coad	Class & Object	Instance Connnection	Gen-Spec	Part-Whole
Jacobson	Object	Acquaintance Association	Inherits	Consists of
Odell	Object Type	Relationship	Subtype	Composition
Rambaugh	Class	Association	Generalizationn	Aggregation
Shlaer/Mellor	Object	Relationship	Subtype	n/a



What is Modeling?

Model

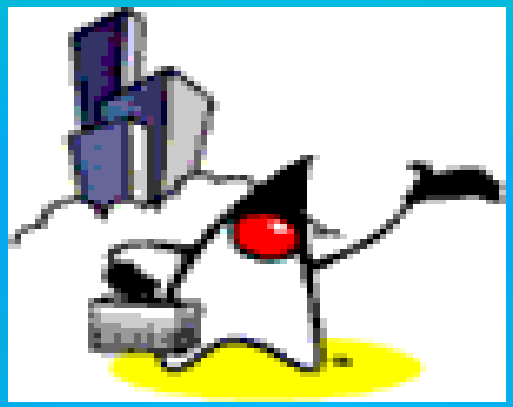
- A model is a pattern of something to be made.
- It is a representation of something in the real world.
  - They are built quicker and easier than the objects they represent.
  - They are used to simulate to better understand the objects they represent.
  - They are modified to evolve as one learns about a task or problem.
  - They are used to represent details of the models that one chooses to see, and others ignored.
  - They are representation of real or imaginary objects in any domain

Four General Elements

- Icons
- Two-dimensional Symbols
- Paths
- Strings

Changes in the Models

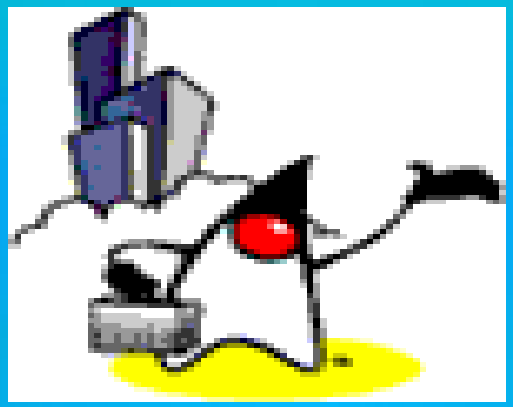
- Level of Abstraction
- Degree of Formality
- Level of Detail



UML Base Diagrams

UML Baseline Diagrams

- Use Case Diagrams*
- Class Diagrams*
- Package Diagrams
- Activity Diagrams
- State-Transition Diagrams
- Sequence Diagrams
- Collaboration Diagrams
- Deployment Diagrams

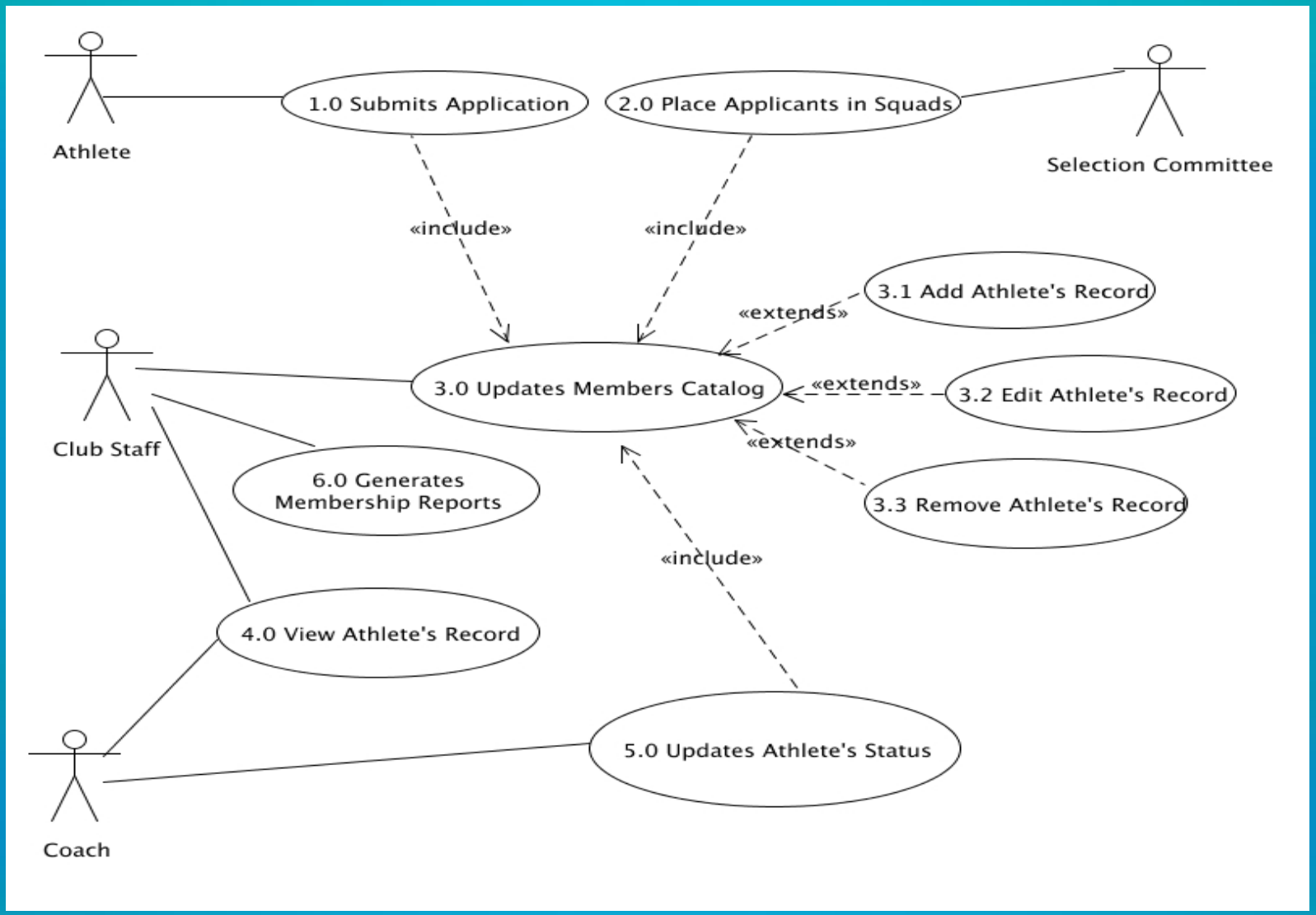


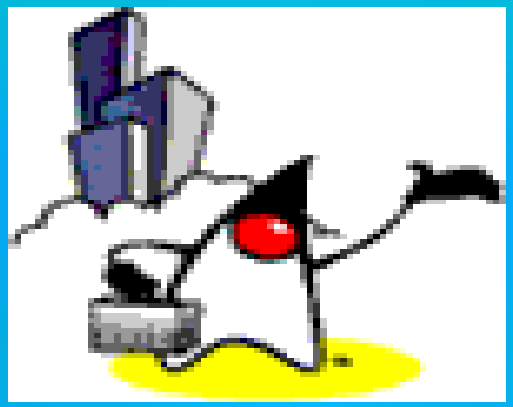
Use Case Diagram

Use Case Diagram

- Provides a basis of communication between end-users, stakeholders and developers in the planning of the software project.
- Attempts to model the system environment by showing the external actors and their connection to the functionality of the system.

Sample Use Case Diagram



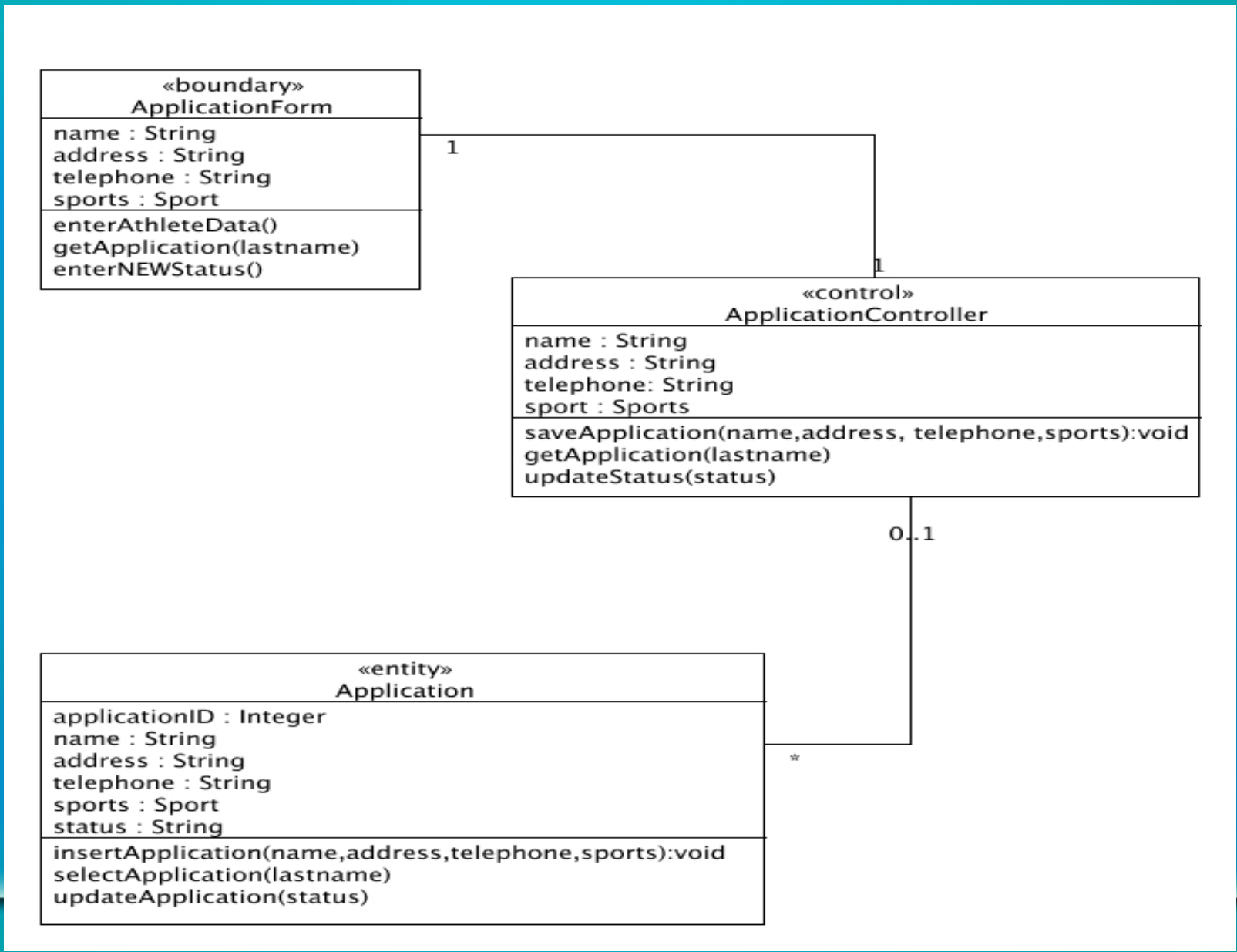


Class Diagram

Class Diagrams

- Shows the static structure of the domain abstractions of the system.
- Describes the types of objects in the system and the various kinds of static relationships that exists among them.
- Show the attributes and operations of a class and constraints for the way objects collaborate

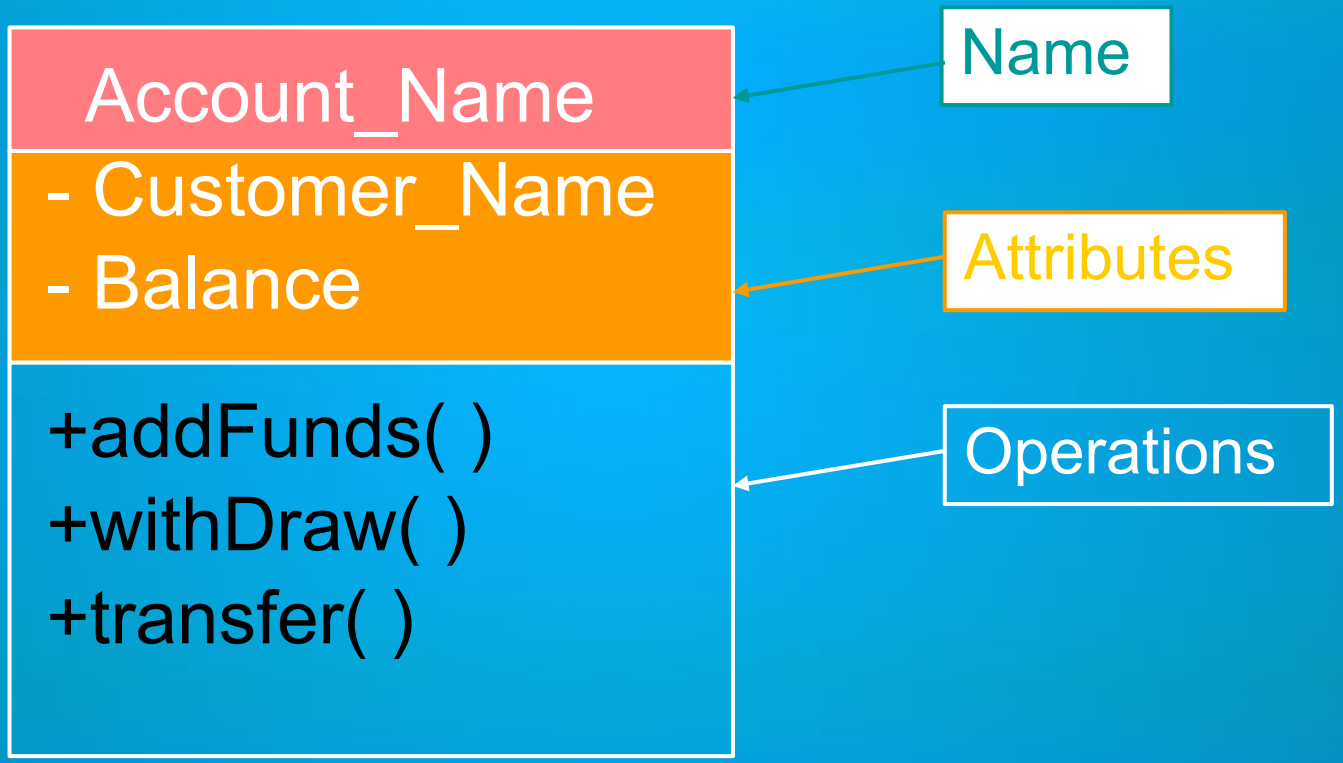
Sample Class Diagrams



Class representation

- Each class is represented by a rectangle subdivided into three compartments
  - Name
  - Attributes
  - Operations
- Modifiers are used to indicate visibility of attributes and operations.
  - '+' is used to denote *Public* visibility (everyone)
  - '#' is used to denote *Protected* visibility (friends and derived)
  - '-' is used to denote *Private* visibility (no one)
- By default, attributes are hidden and operations are visible.

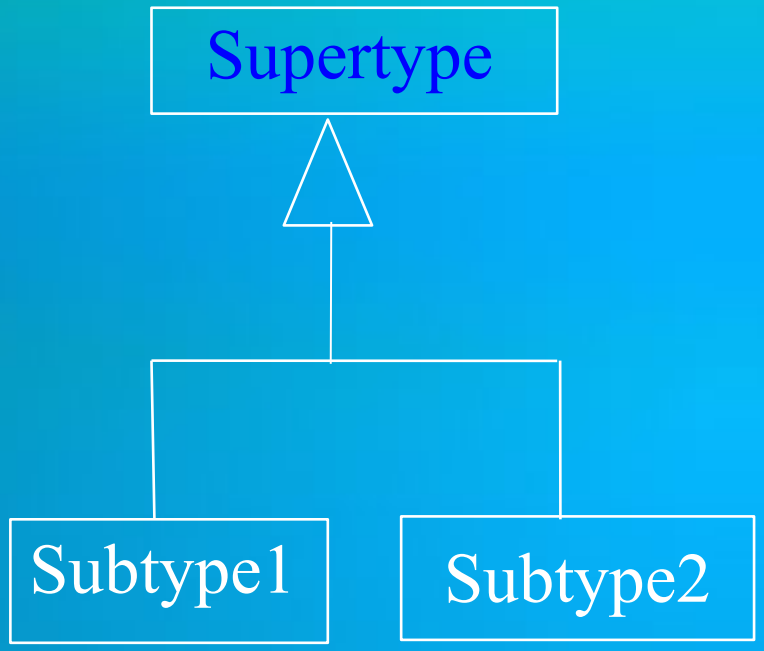
An example of Class



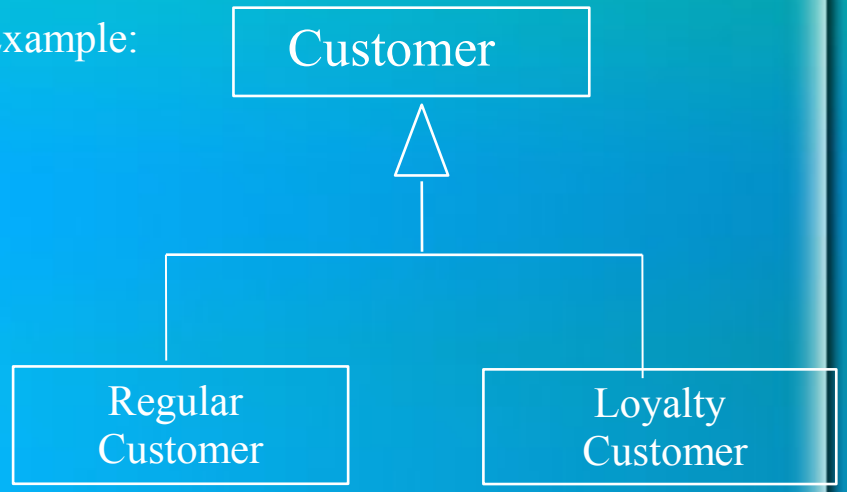
OO Relationships

- There are two kinds of Relationships
  - Generalization (parent-child relationship)
  - Association (student enrolls in course)
- Associations can be further classified as
  - Aggregation
  - Composition

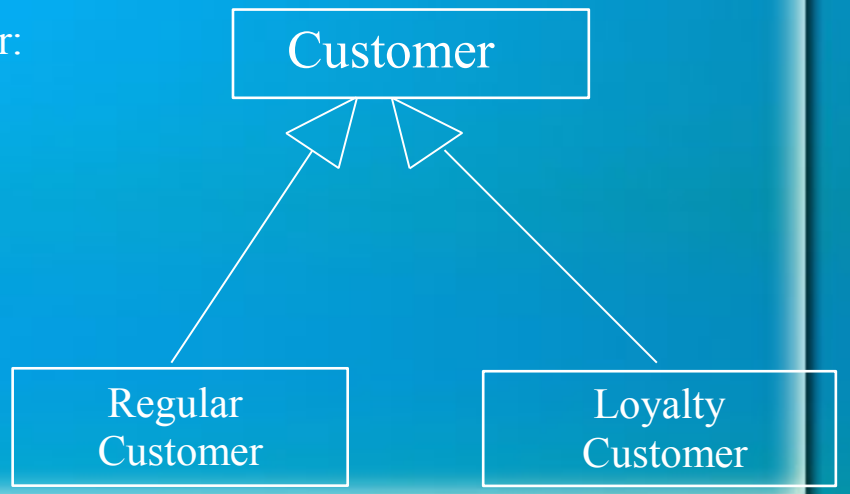
OO Relationships: **Generalization**



Example:



or:



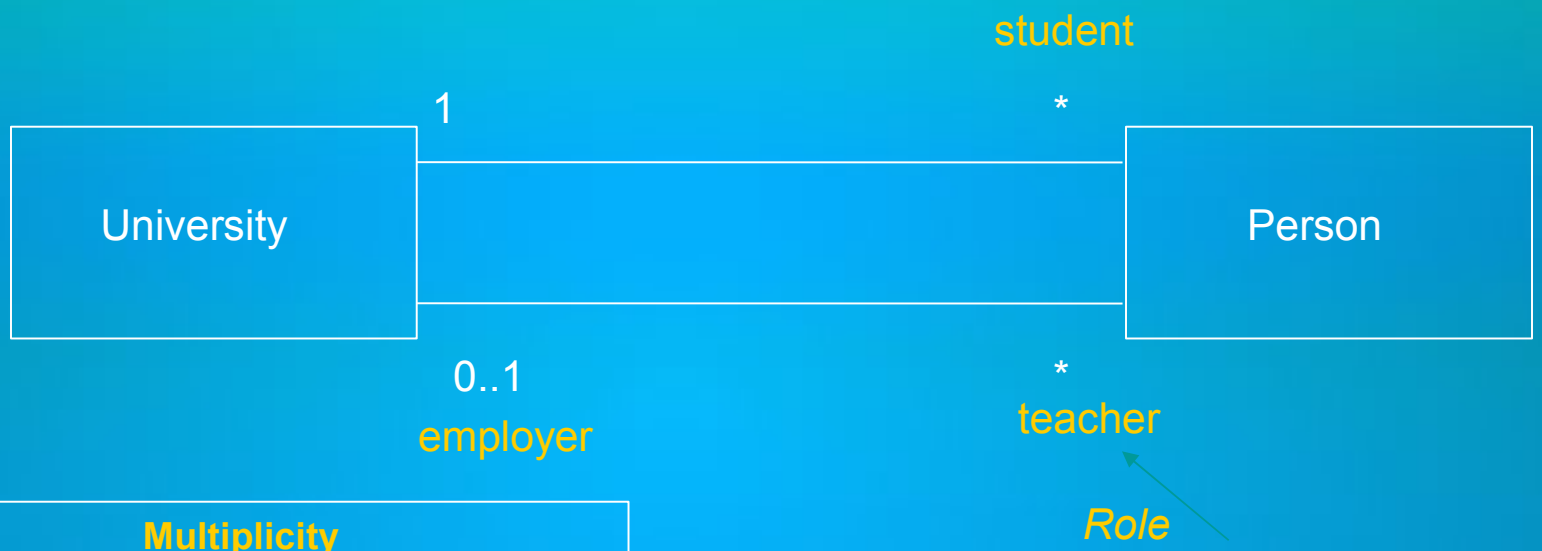
- Generalization expresses a parent/child relationship among related classes.

- Used for abstracting details in several layers

OO Relationships: Association

- Represent relationship between instances of classes
  - Student enrolls in a course
  - Courses have students
  - Courses have exams
  - Etc.
- Association has two ends
  - Role names (e.g. enrolls)
  - Multiplicity (e.g. One course can have many students)
  - Navigability (unidirectional, bidirectional)

Association: Multiplicity and Roles



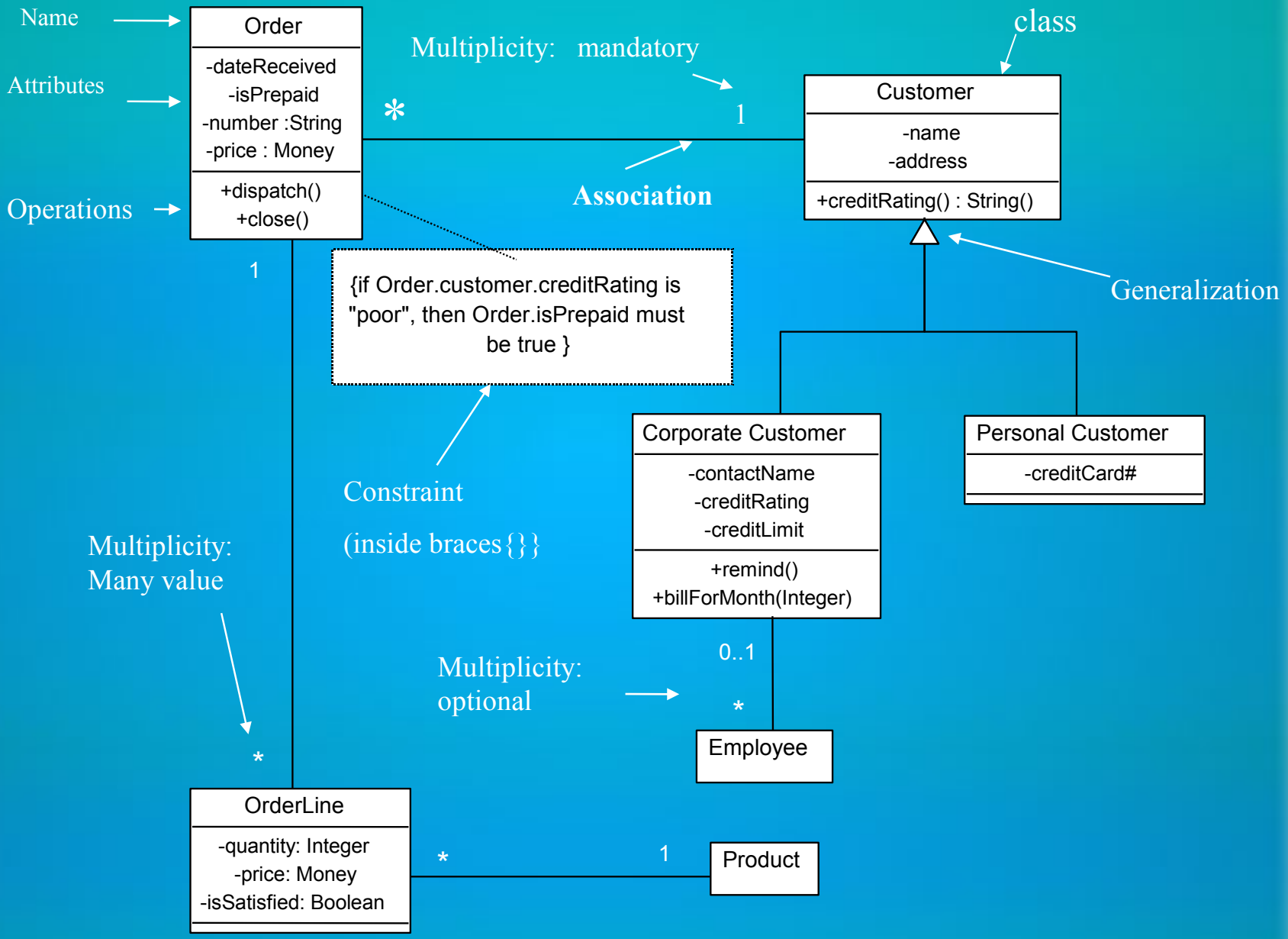
Multiplicity

Symbol	Meaning
1	One and only one
0..1	Zero or one
M..N	From M to N (natural language)
* integer	From zero to any positive
0..* integer	From zero to any positive
1..* integer	From one to any positive

Role

"A given university groups many people; some act as students, others as teachers. A given student belongs to a single university; a given teacher may or may not be working for the university at a particular time."

Class Diagram



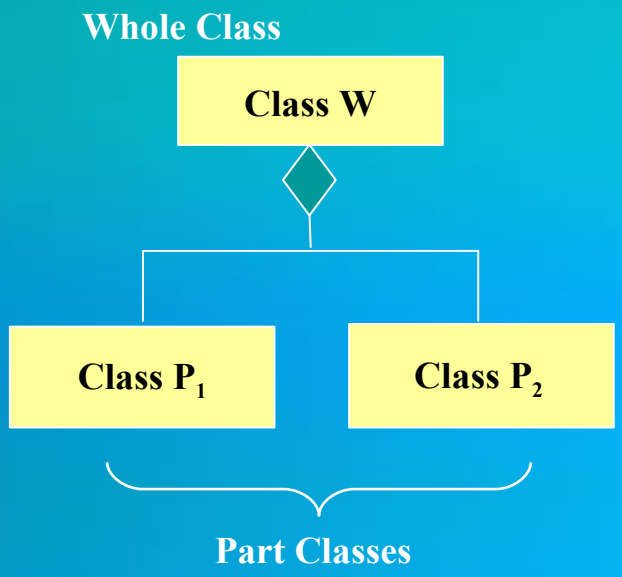
[from *UML Distilled Third Edition*]

Association: Model to Implementation


```
Class Student {
    Course enrolls[4];
}

Class Course {
    Student have[];
}
```

OO Relationships: **Composition**



Composition: expresses a relationship among instances of related classes. It is a specific kind of Whole-Part relationship.

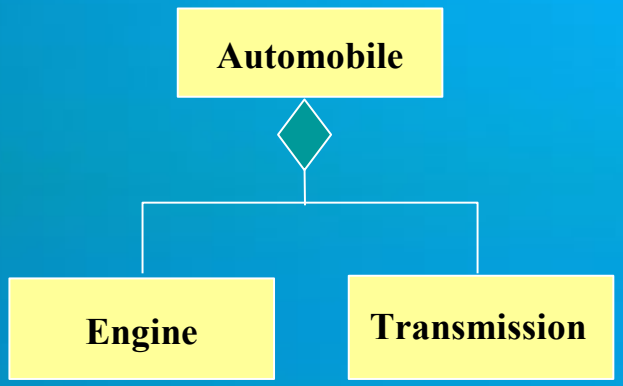
It expresses a relationship where an instance of the Whole-class has the responsibility to create and initialize instances of each Part-class.

It may also be used to express a relationship where instances of the Part-classes have privileged access or visibility to certain attributes and/or behaviors defined by the Whole-class.

Composition should also be used to express relationship where instances of the Whole-class have exclusive access to and control of instances of the Part-classes.

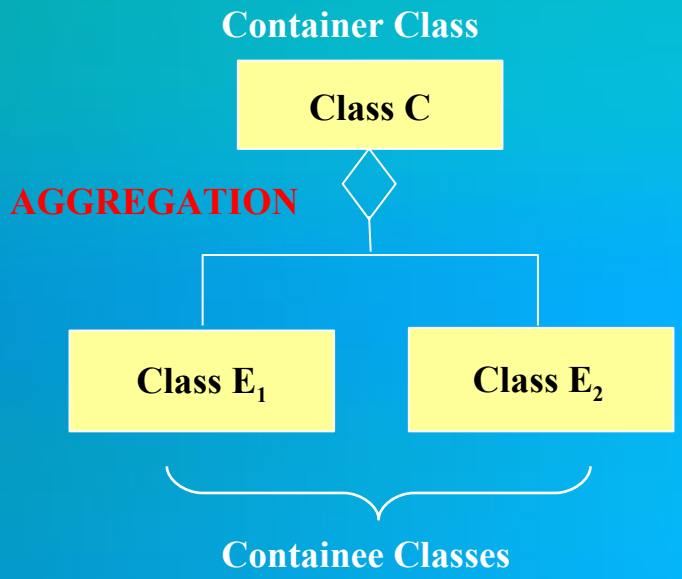
Composition should be used to express a relationship where the behavior of Part instances is undefined without being related to an instance of the Whole. And, conversely, the behavior of the Whole is ill-defined or incomplete if one or more of the Part instances are undefined.

Example



[From Dr.David A. Workman]

OO Relationships: Aggregation

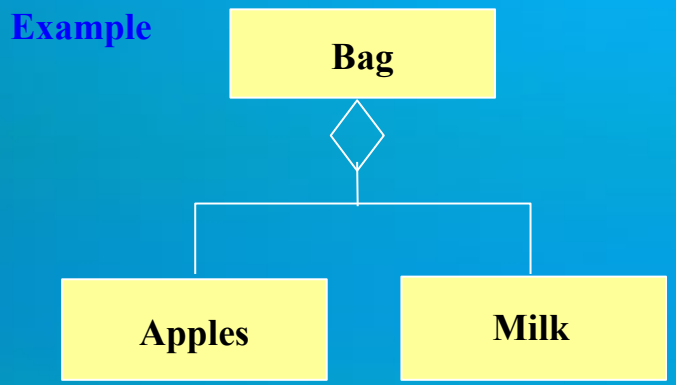


Aggregation: expresses a relationship among instances of related classes. It is a specific kind of Container-Containee relationship.

It expresses a relationship where an instance of the Container-class has the responsibility to hold and maintain instances of each Containee-class that have been created outside the auspices of the Container-class.

Aggregation should be used to express a more informal relationship than composition expresses. That is, it is an appropriate relationship where the Container and its Containees can be manipulated independently.

Aggregation is appropriate when Container and Containees have no special access privileges to each other.



[From Dr.David A. Workman]

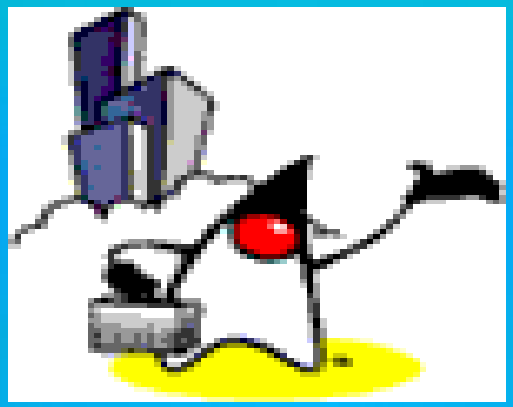
Aggregation vs. Composition

- **Composition** is really a strong form of **aggregation**
  - components have only one owner
  - components cannot exist independent of their owner
  - components live or die with their owner

  e.g. Each car has an engine that can not be shared with other cars.

- **Aggregations** may form "part of" the aggregate, but may not be essential to it. They may also exist independent of the aggregate.

  e.g. Apples may exist independent of the bag.

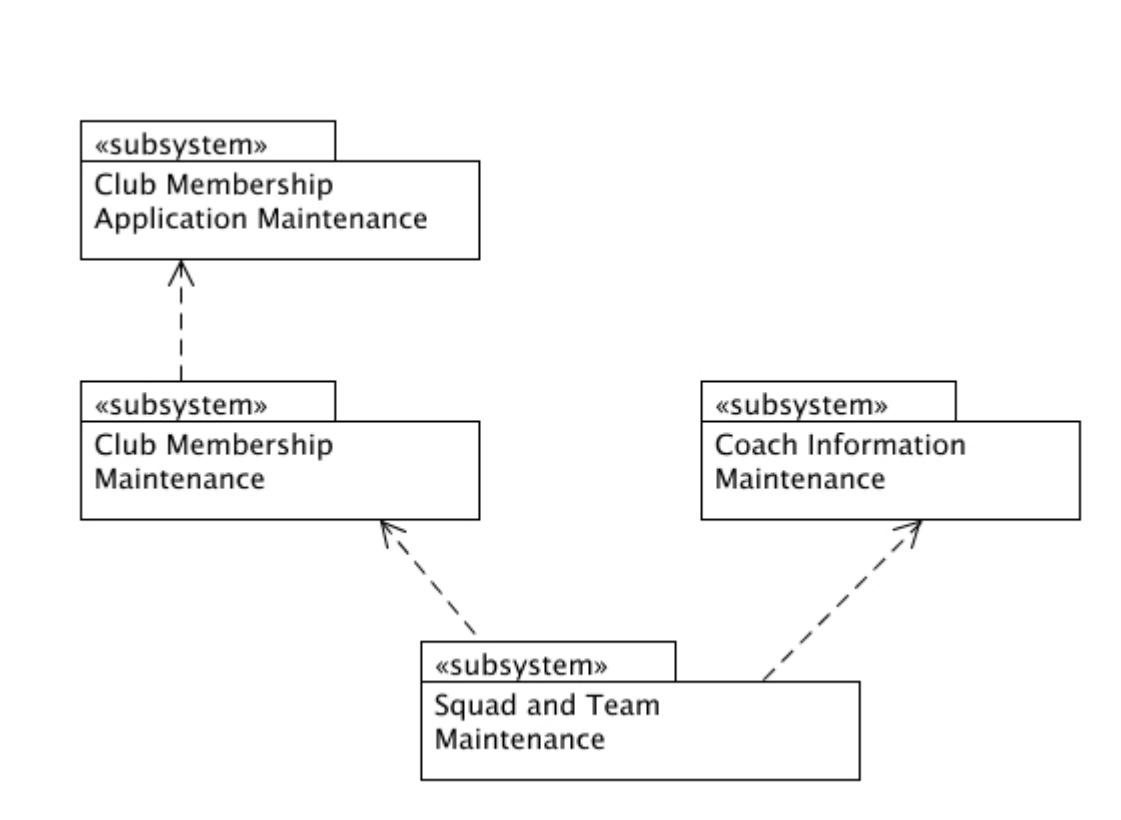


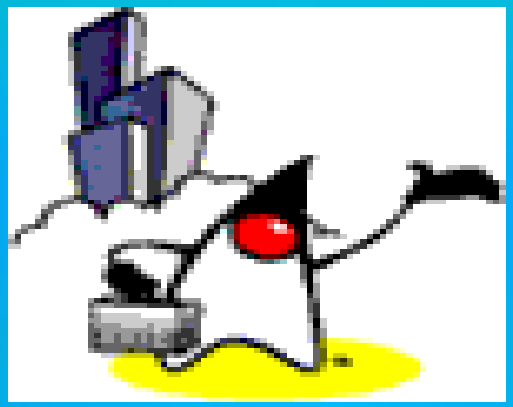
Package Diagram

Package Diagrams

- Shows the breakdown of larger systems into a logical grouping of smaller subsystems.
- Shows groupings of classes and dependencies among them
- A dependency exists between two elements if changes to the definition of one element may cause changes to the other.

Sample Package Diagram



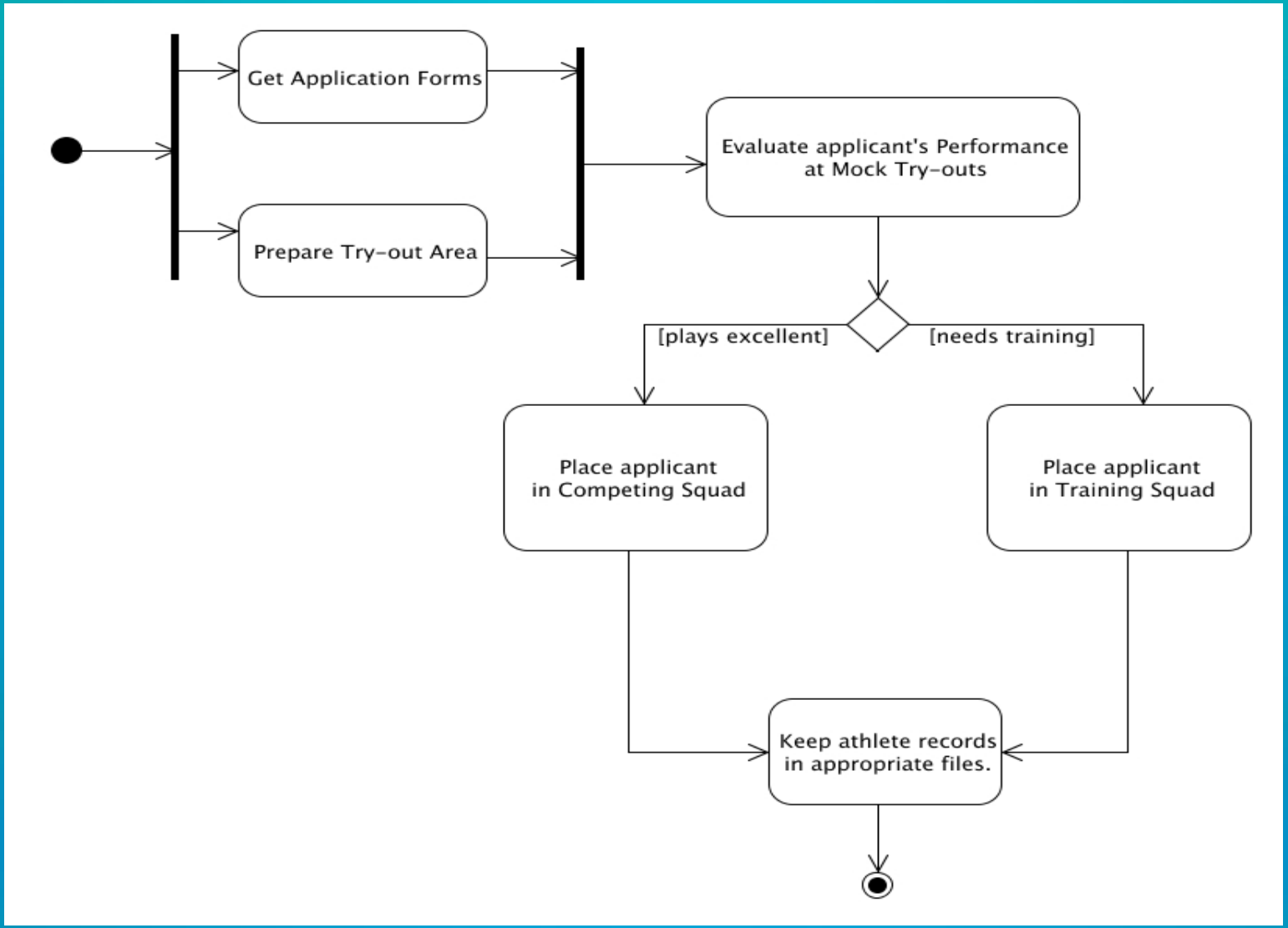


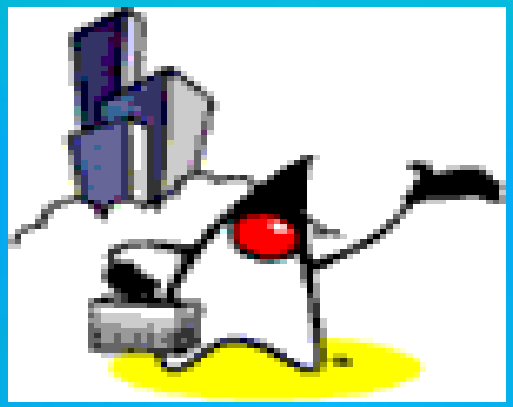
Activity Diagram

Activity Diagram

- Show the sequential flow of activities
  - Typically, in an operation
  - Also in a use case or event trace
- Complement the class diagram by showing the workflow of the business (a.k.a Flowchart)
- Encourage discovery of parallel processes which helps eliminate unnecessary sequences in business processes

Sample Activity Diagram



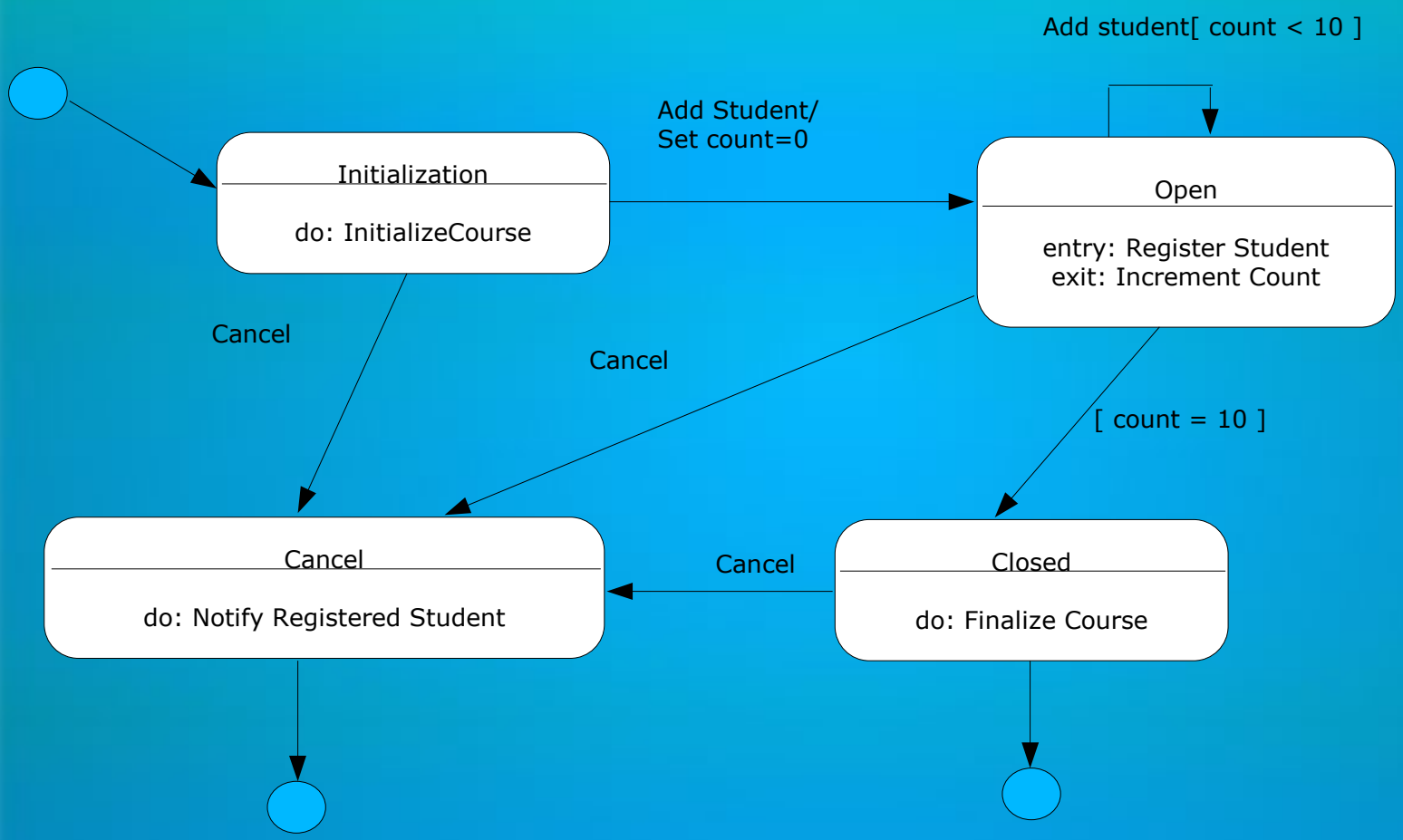


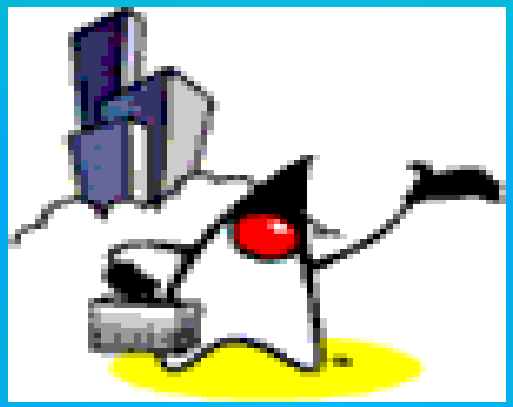
State Transition Diagram

State-Transition Diagrams

- Show all the possible states that objects of the class can have and which events cause them to change
- Show how the object's state changes as a result of events that are handled by the object
- Good to use when a class has complex lifecycle behavior

Sample State-Transition Diagram



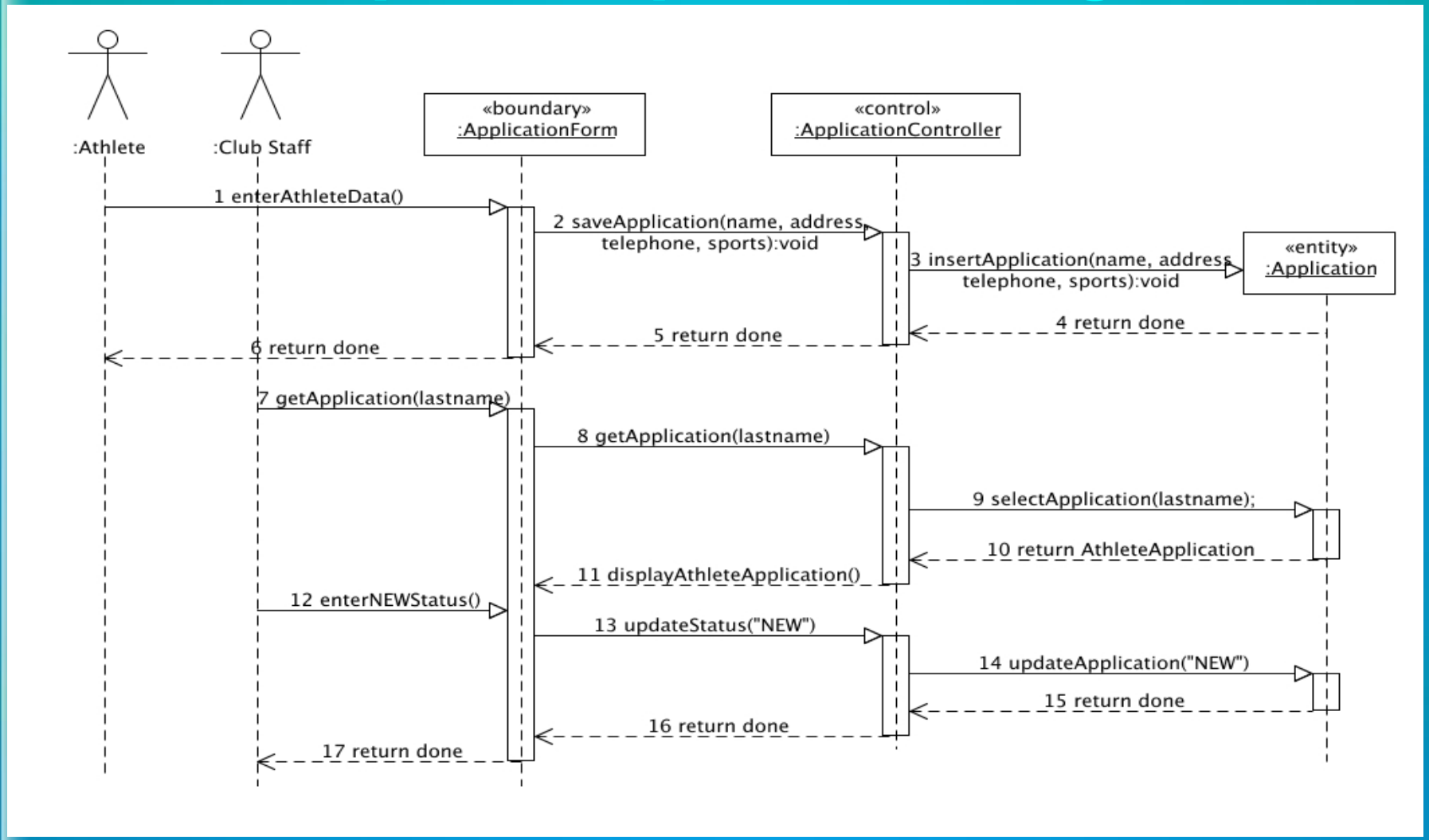


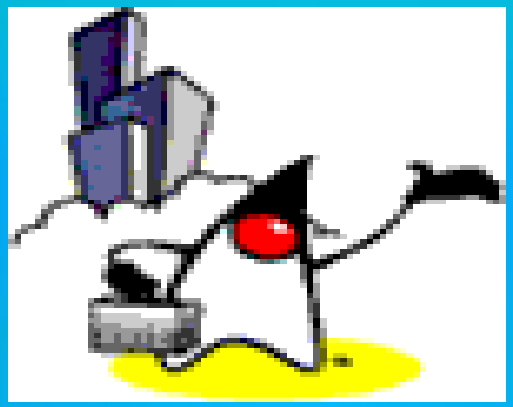
Sequence Diagram

Sequence Diagrams

- Show the dynamic collaboration between objects for a sequence of messages send between them in a sequence of time
- Time sequence is easier to see in the sequence diagram read from top to bottom
- Choose sequence diagram when only the sequence of operations needs to be shown

Sample Sequence Diagram



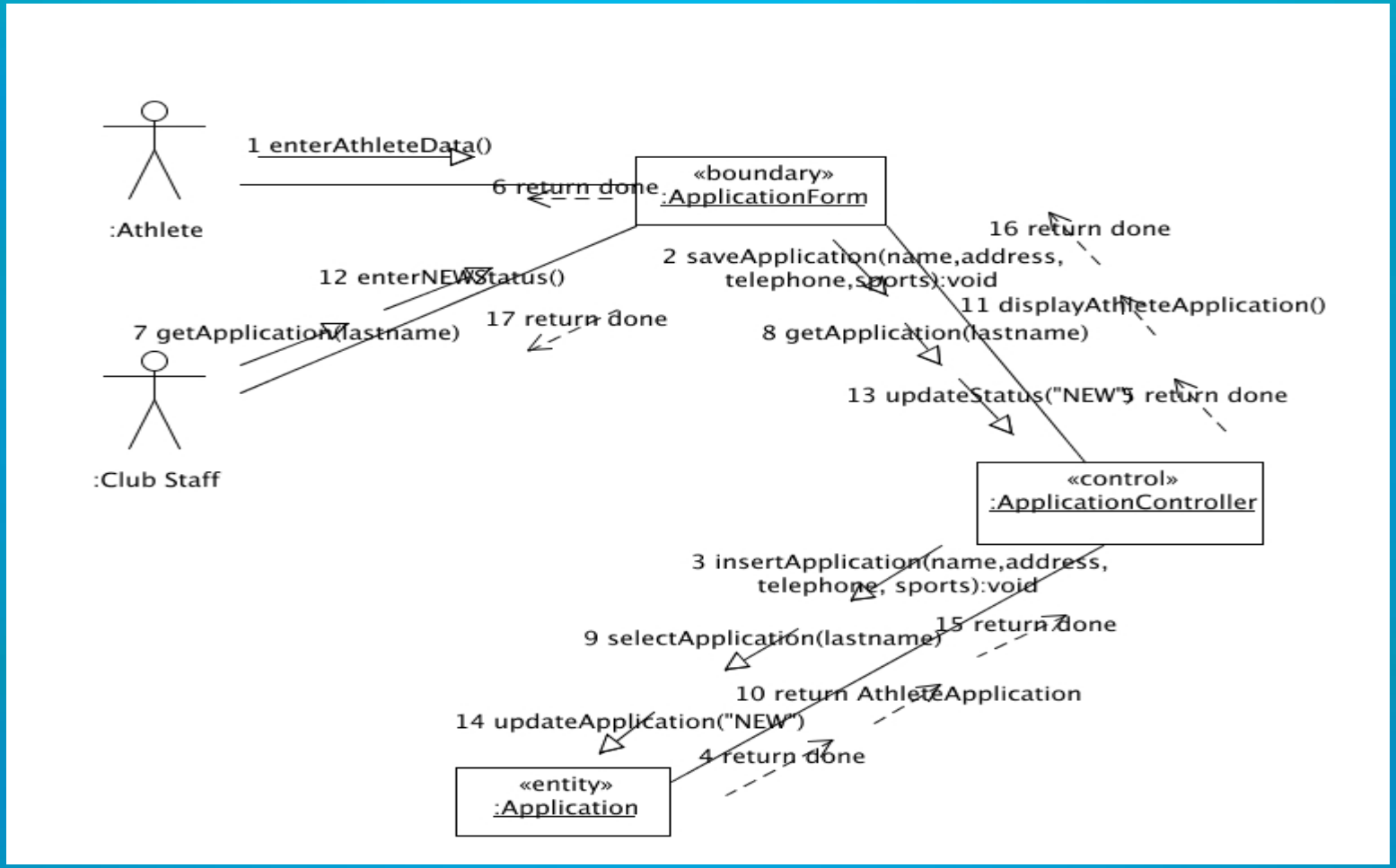


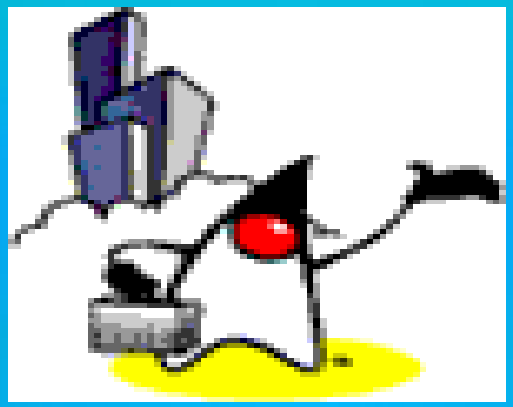
Collaboration Diagram

Collaboration Diagram

- Show the actual objects and their links, the "network of objects" that are collaborating
- Time sequence is shown by numbering the message label of the links between objects
- Choose collaboration diagram when the objects and their links facilitate understanding the interaction, and sequence of time is not as important

Sample Collaboration Diagram



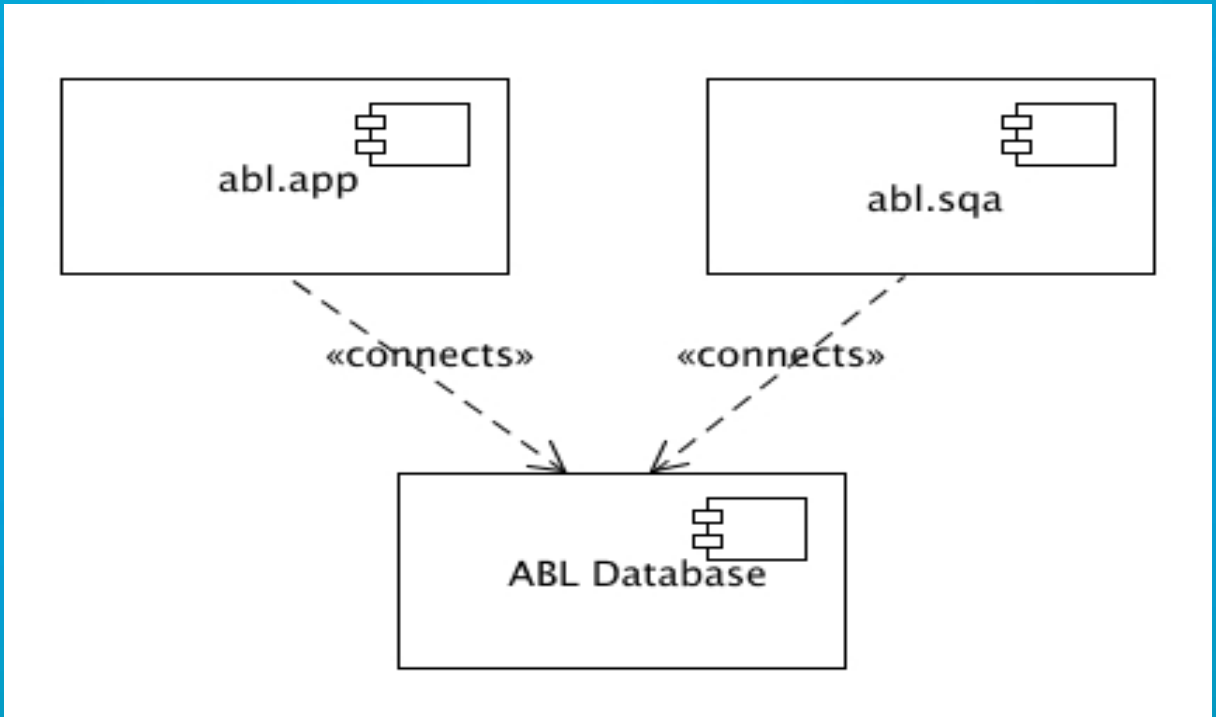


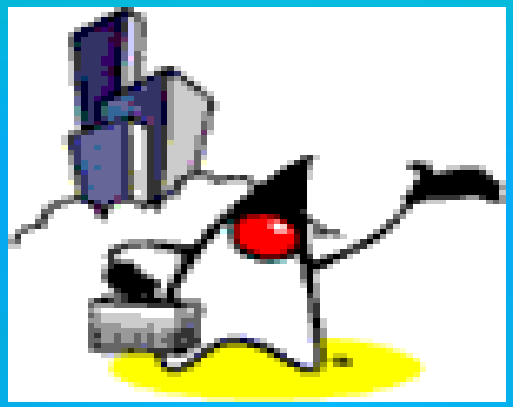
Component Diagram

Component Diagram

- It is used to model the components of the software.
- Components may be a source code, binary code, executable code or dynamically linked libraries.
- Components encapsulates implementation and shows a set of interfaces.

Sample Component Diagram



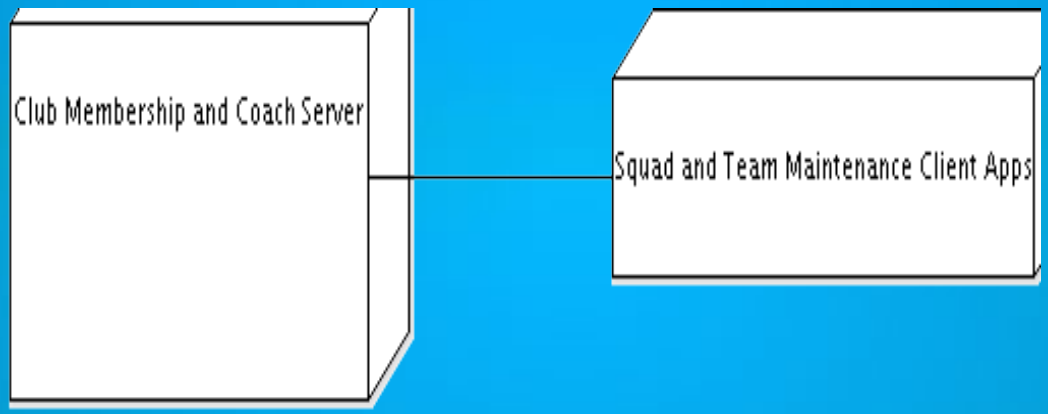


Deployment Diagram

Deployment Diagram

- Show the physical architecture of the hardware and software of the system
- Highlight the physical relationship among software and hardware components in the delivered system
- Components on the diagram typically represent physical modules of code and correspond exactly to the package diagram

Sample Deployment



United Modeling Language (UML)