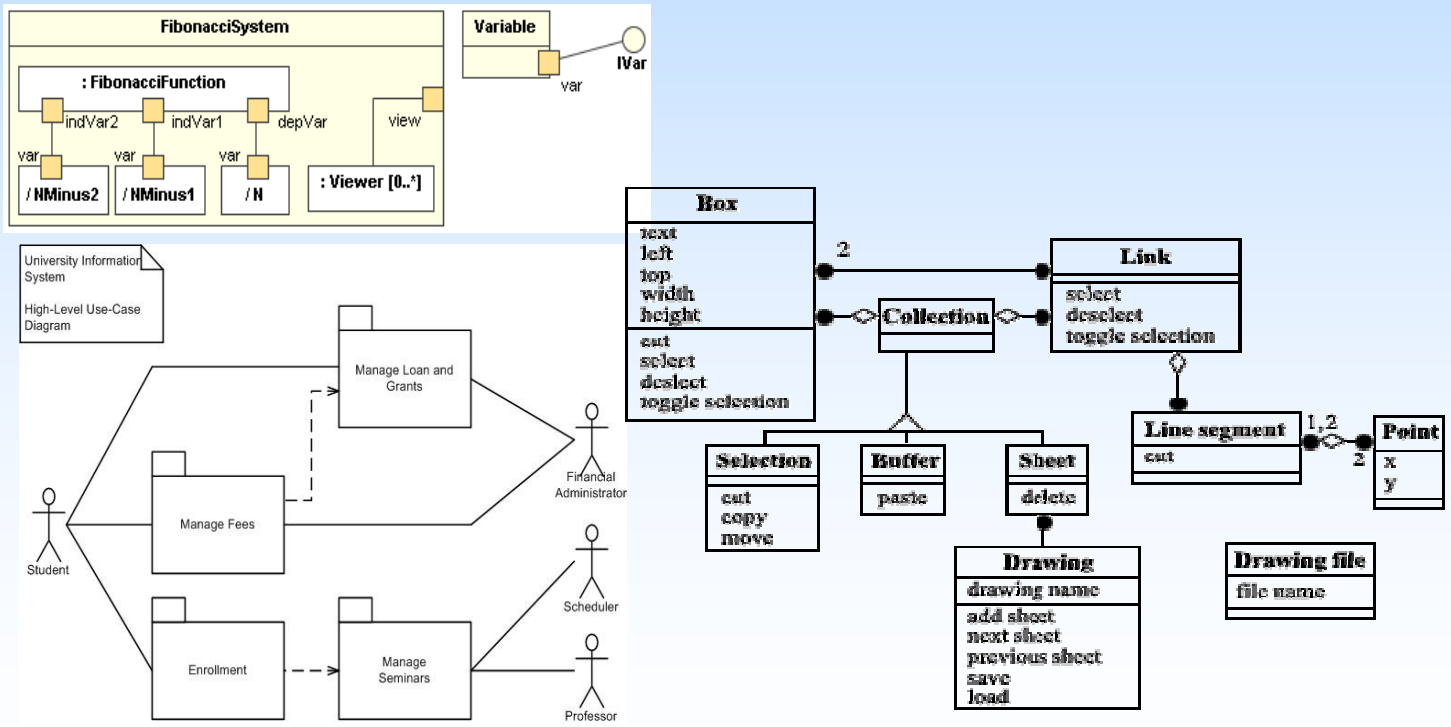
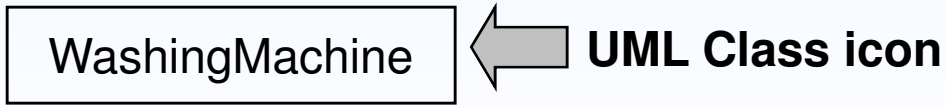
Identifying Classes, Packages and drawing a Class diagrams, Object diagrams



Visualizing a Class

- Represented by a rectangle
- Class naming
  - represented by a word with an initial uppercase letter.
  - Appears near the top of the rectangle
  - If the class has two word name, join the two words together and capitalize the first letter of the second word.

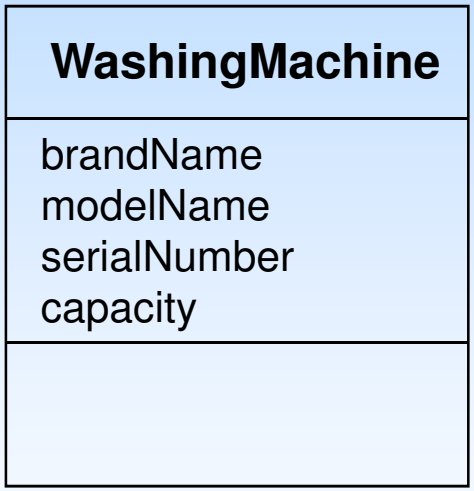


UML Class icon

UCSC

Attributes

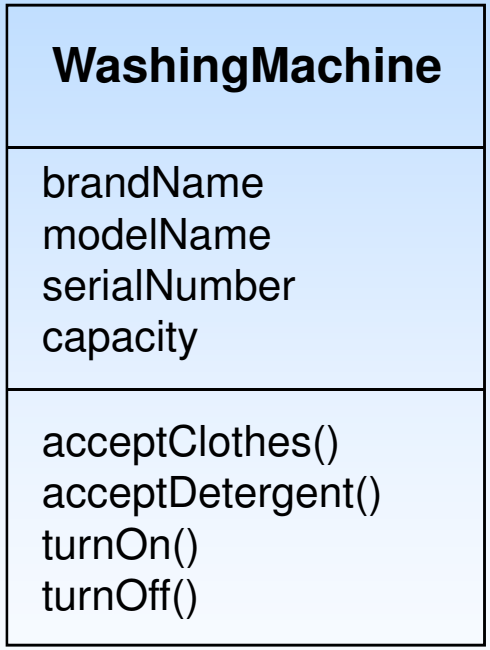
- A property of a class
- Describes a range of values that the property may hold in objects of that class.
- A class may have zero or more attributes.
- If name consists of more than one word the words are jointed and each word begins with a uppercase letter.



UCSC

Operations

- Something a class can do
- Can indicate additional information for operations.



Working with Relationships

The Need for Relationships

- All systems are made up of objects and classes.
- System behavior is achieved through the interactions of the objects in the system.

The Need for Relationships

- For example : When a member wants to borrow a book in a library system (*borrowing* use case), the system has to interact with the following objects:

  book, copy, borrower and borrowed copy

- For the *borrowing* use case following are some of the messages that these objects have to send and receive.

  checkBorrowerId , checkCopyBorrowable, checkOverdue, checkOverlimit etc.

The Need for Relationships

- Relationships provide the conduct for object interactions.
- A relationship is a semantic connection between classes.
- It allows one class to know about the attributes, operations of another class.

The Need for Relationships

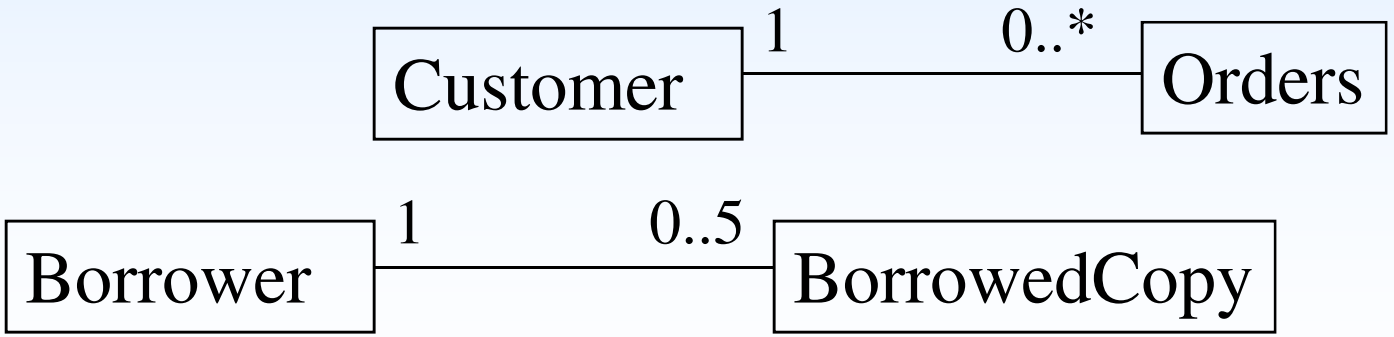
- In order for one class to send a message to another on a sequence diagram or collaboration diagram (see later), there must be a relationship between the two classes.

- There are four types of relationships you can set up between classes.

  Association, Aggregation, Generalization and Dependency.

Associations

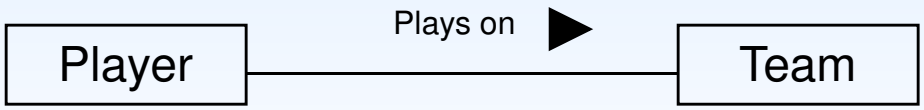
- Indicate a connection (a link) between classes.
- Each class can send messages to the other.
- It can be bi-directional or unidirectional.
- In UML, bi-directional associations are drawn either with arrowheads on both ends or without arrowheads altogether.

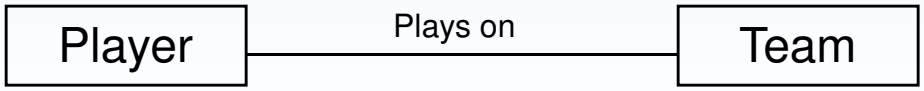


UCSC

Associations

- Name of the association is written just above the line.
- The way to read a relationship can be shown using a filled triangle pointing in the appropriate direction (optional).



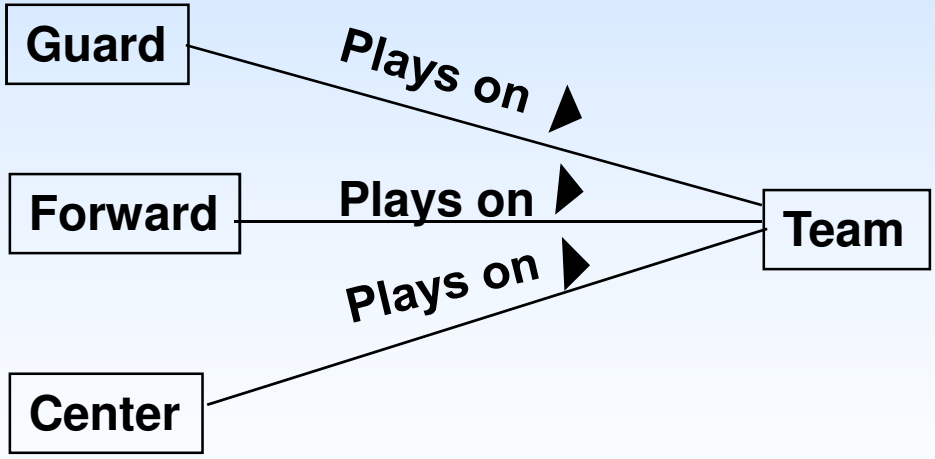


UCSC

Associations

- Several classes can connected to one class.

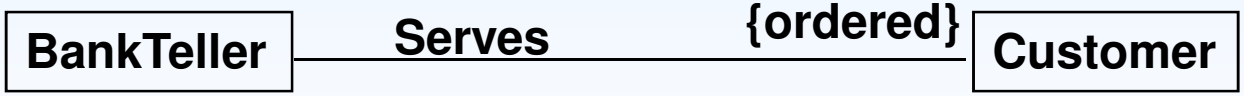
Example:



Constraints on Associations

- An association between two classes has to follow a rule.
- You can indicate that rule by putting a constraint near the association line.

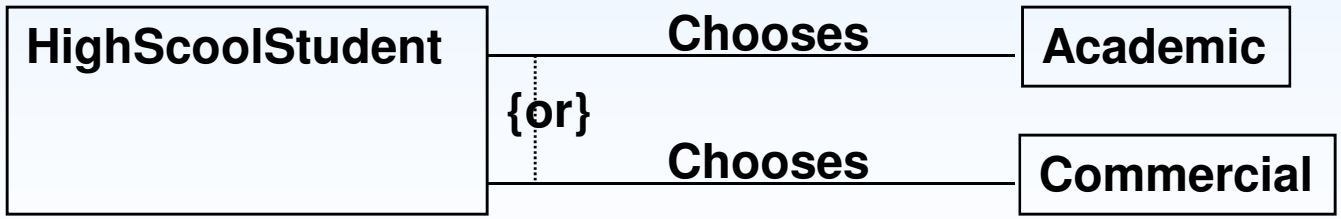
Example:



Constraints on Associations

- Or relationship
  - Signified by {or} on a dashed line that connects two association lines

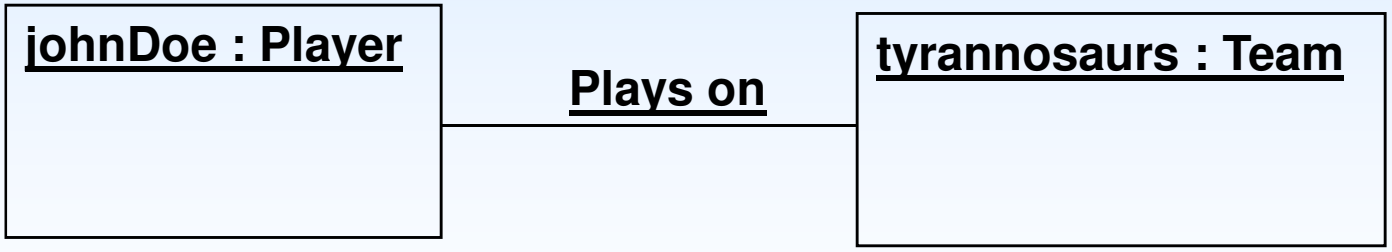
Example: a high school student can choose either an academic course of study or a commercial one.



Associations

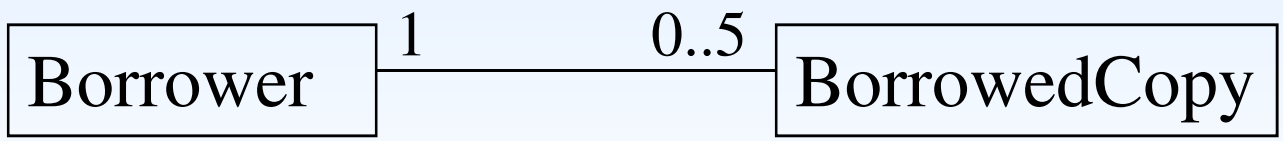
- Links
  - An association may contain instances

  e.g. a specific player who plays for a specific team.



Associations

- Multiplicity
  - The number of objects from one class that relate with a single object in an associated class.
  - Multiplicity can be denoted near the appropriate class



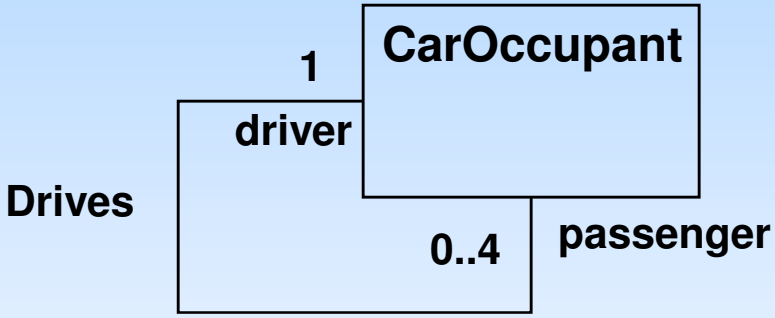
Multiplicity

1	Exactly one
0..*	Zero or more
1..*	One or more
0..1	Zero or one
5..8	Specific Range (5,6,7 or 8)

Reflexive Associations

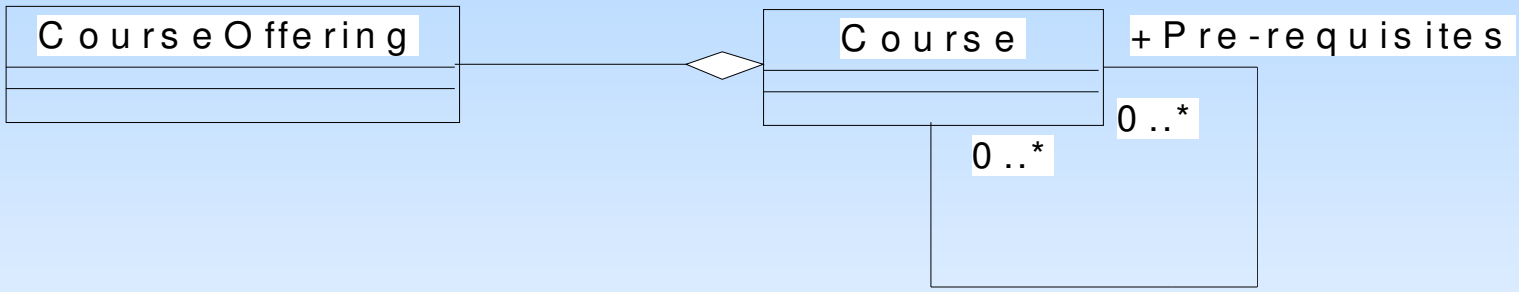
- Some times a class is in an association with itself.
- This can happen when a class has objects that can play a variety of roles.
- Representation:
  - Draw an association line from the class rectangle back to the same class rectangle
  - On the association line indicate the roles, name of the association, direction of the association, and multiplicity.

Reflexive Associations



- A carOccupant can be either a driver or a passenger
- In the role of the driver, one carOccupant drives zero or more additional carOccupants who play the role of passenger

Reflexive Associations



- One Course object playing the role of Prerequisite is related to zero or more course objects.
- One Course object is related to zero or more course objects playing the role of Prerequisite.

UCSC

Inheritance (Generalization)

- Provides the capability to create a hierarchy of classes.
- Common structure and behavior are shared among classes.
- The term super-class or parent class is the name given to the class holding the common information.
- The descendants are called subclasses or child class.
- A subclass (child class) inherits all attributes,operations, and relationships that are defined for all of its super-classes (parent class).

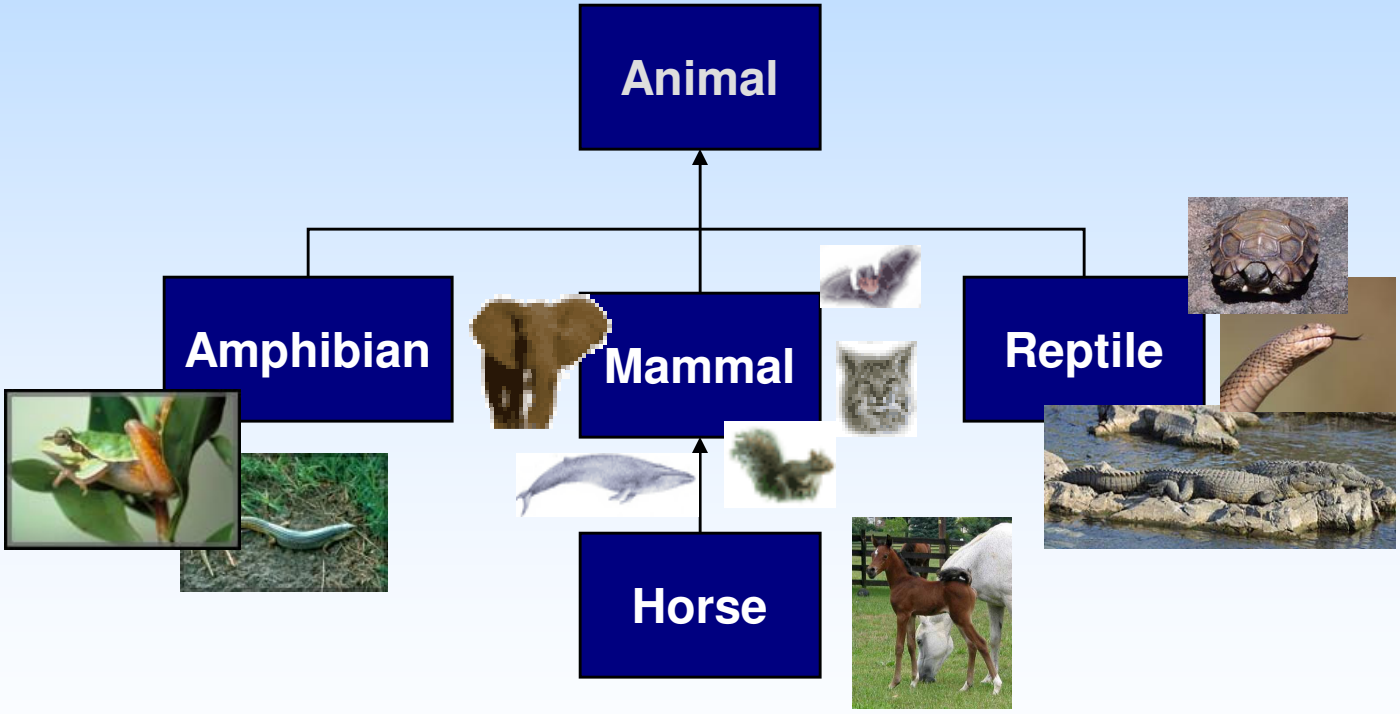
Inheritance (Generalization)

- An inheritance relationship:
  - is not a relationship between different objects.
  - is a relationship between different classes.
  - is never named.
  - *Role* names are not used.
  - Multiplicity does not apply.
- Inheritance is the key to reuse.
  - A class can be created for one application
  - A sub class may be created to add more information needed for a different application.

Inheritance (Generalization)

- There are two ways to find inheritance in any system: *Generalization* and *Specialization*.
- *Generalization* provides the capability to create super-classes that encapsulate structure and behavior common to several classes.
- *Specialization* provides the ability to create subclasses that represent refinement to the super-class. Typically structure and behavior are added to the new subclass.

Inheritance (Generalization)



Inheritance (Generalization)

- A class might have no parents in which case it's a **base class** / **root class.**
- A class might have no children in which case it's a **leaf class**.

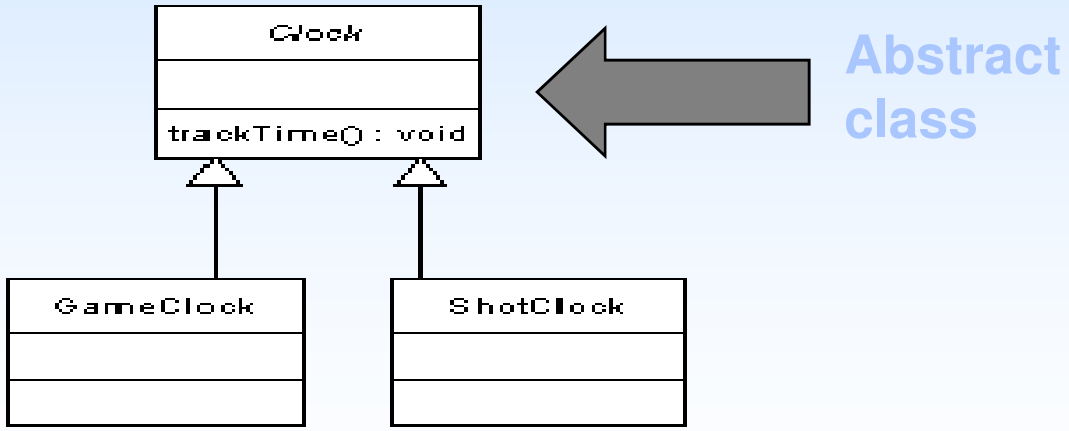
Single inheritance vs multiple inheritance

- **Single inheritance** - a class has exactly one parent.
  - Savings A/C is a kind of Account.
- **Multiple inheritance** – a class has more than one parent.
  - Interest cheque A/C is a kind of Savings A/C and also a kind of Current A/C.

Abstract Class

- Intended only as bases for inheritance
- Provides no objects of their own
- Indicated by writing the name of the abstract class in italics

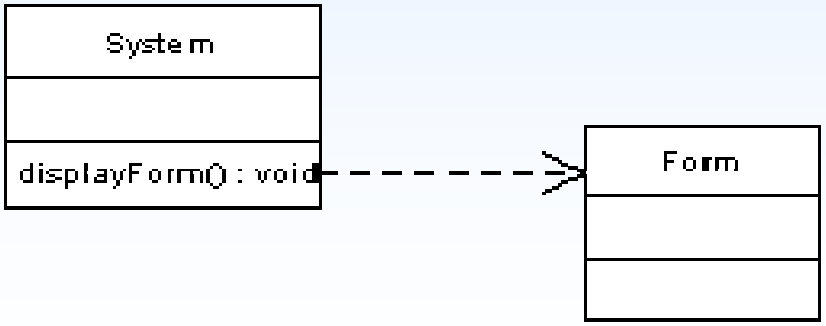
e.g.



UCSC

Dependencies

- In a dependency one class uses another
- Most common usage is to show that a signature in the operation of one class uses another class.
- Depicted as a dashed line joining the two classes in the dependency, with an arrow head adjoining the depended-on class.



UCSC

Class diagrams and Object diagram

- **Class diagram:** gives general, definitional information – the properties of a class and its attributes, and other classes it associates with.
- **Object diagram:** gives information about specific instances of a class and how they link up at specific instants in time.

UCSC

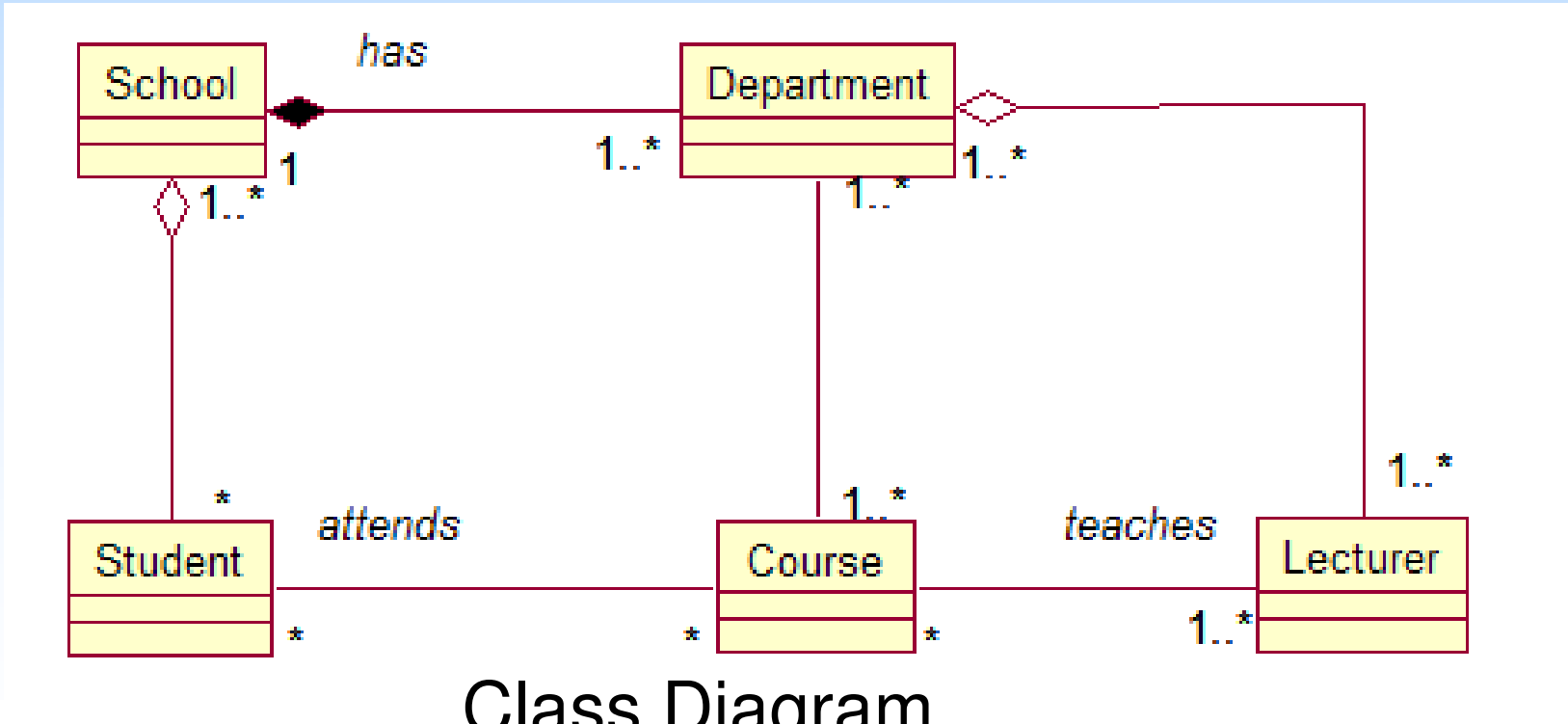
Class diagrams and Object diagram

Class Diagrams :

- Shows set of *classes*, *interfaces*, and *collaborations* and their relationships.
- Most common diagram found in modelling object-oriented system.
- Address the static view of a system.

class: a category or group of things that have the same attributes and the same behaviours.

Class diagrams and Object diagram



Class Diagram

e.g. School of Computing

Class diagrams and Object diagram

Object Diagrams :

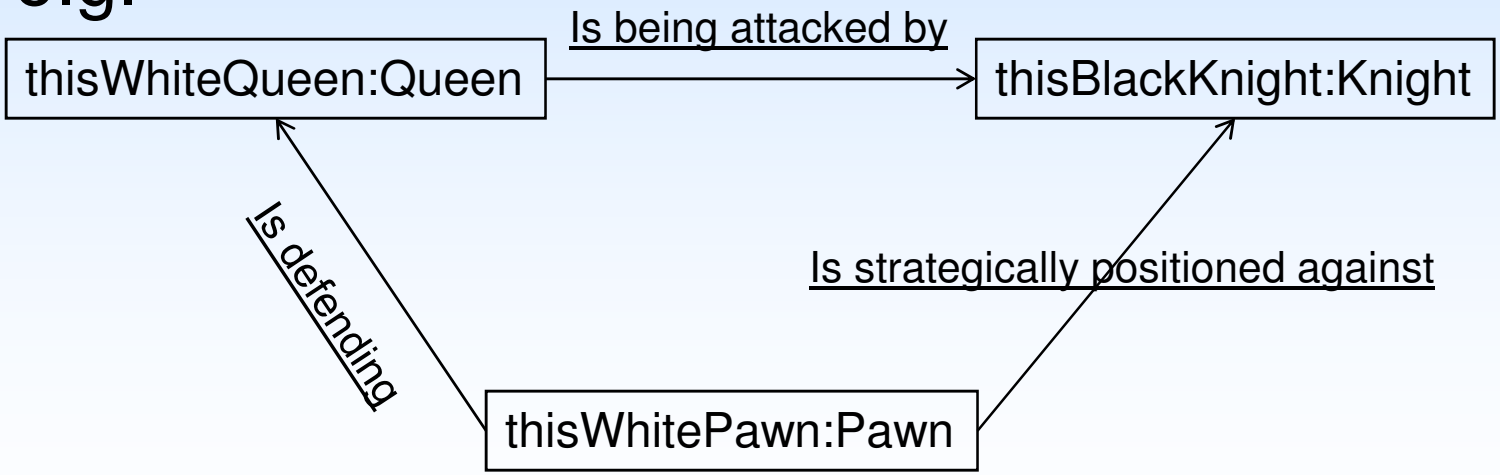
- Similar to a class diagram
- Models actual object instances with current attribute values.
- Shows a set of *objects* and their relationships.
- Provides a snap shot of the system's object at one point in time.

UCSC

Class diagrams and Object diagram

Object Diagram

e.g.



What is an object?

- An Object is a representation of an entity.
- It can represent something concrete, such as Harsha's truck or concept such as a bank transaction or a purchase order.
- Each object in a system has three characteristics: *state, behavior,* and *identity.*

State, Behavior and identity

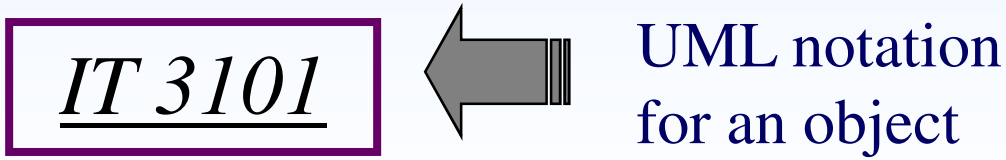
- The **State** of an object is one of the possible conditions in which it may exist.

  e.g. States of a Hotel Room are *occupied*, *available,* and *reserved*.

- The state of an object typically changes over time.

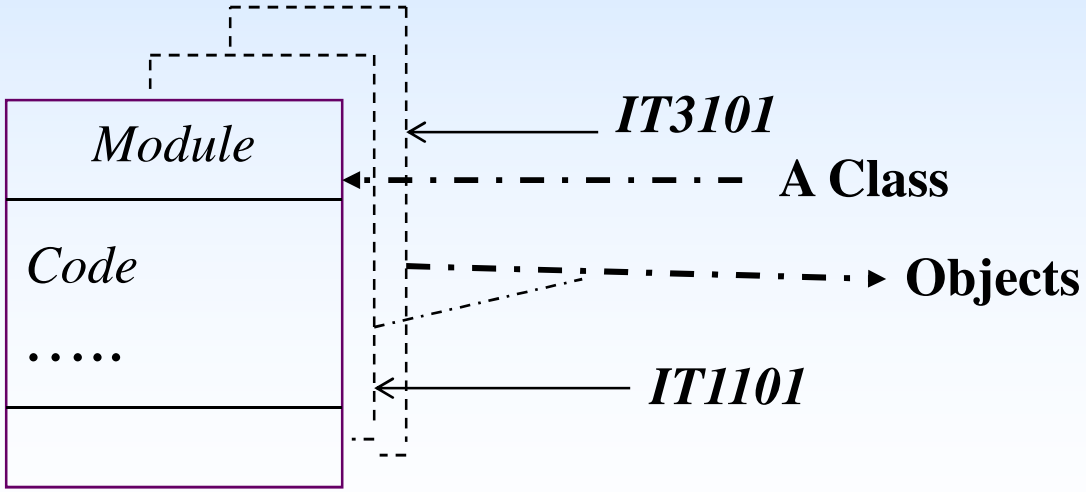
State, Behavior and identity

- **Behavior** determines how an object responds to requests from other objects.
  - Behavior is implemented by the set of operations for the object.
- **Identity** means that each object is unique, even if its state is identical to that of another object.



What is a class?

- A Class is a description of a group of objects with common properties (attributes), common behavior (operations), common relationships to other objects, and common semantics.

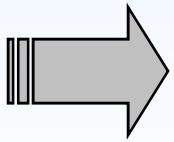


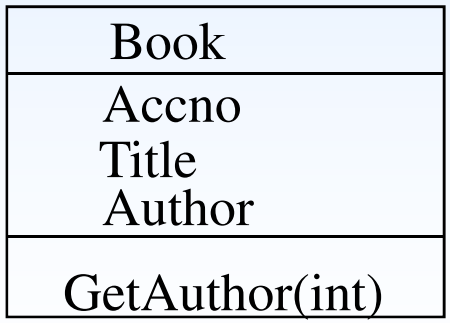
UCSC

Class Diagrams

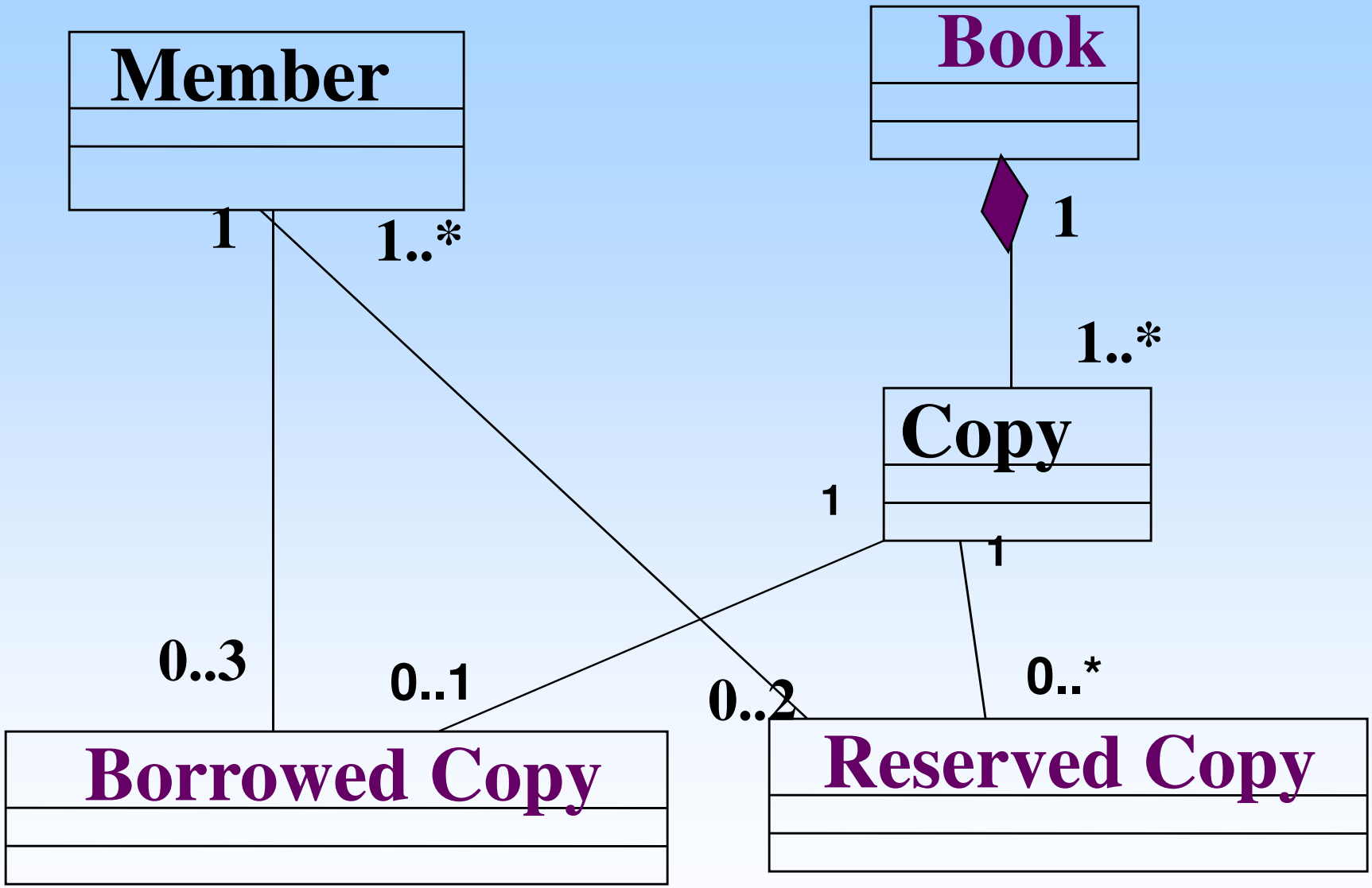
- Backbone of nearly all OO Methods.
- A class diagram describes the types of objects in the system and the various kinds of static relationships that exist among them.
- It also shows the attributes and services of a class and the constraints that apply to the way objects are connected.

UML Notation for a Class





UCSC



Class Diagram for a library system

Stereotypes and Classes

A Stereotype is a mechanism you can use to categorize your classes.

- Say you want to quickly find all of the forms in the model,
- You could create a stereotype called *form*, and assign all of your windows this stereotype.
- To find your forms later, you would just need to look for the classes with that stereotype.

Stereotypes and Classes

- There are *three* primary class stereotypes in UML.

Boundary

Entity

Control

UCSC

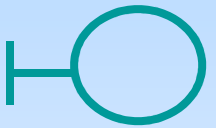
Stereotypes and Classes

***Boundary** Class:*

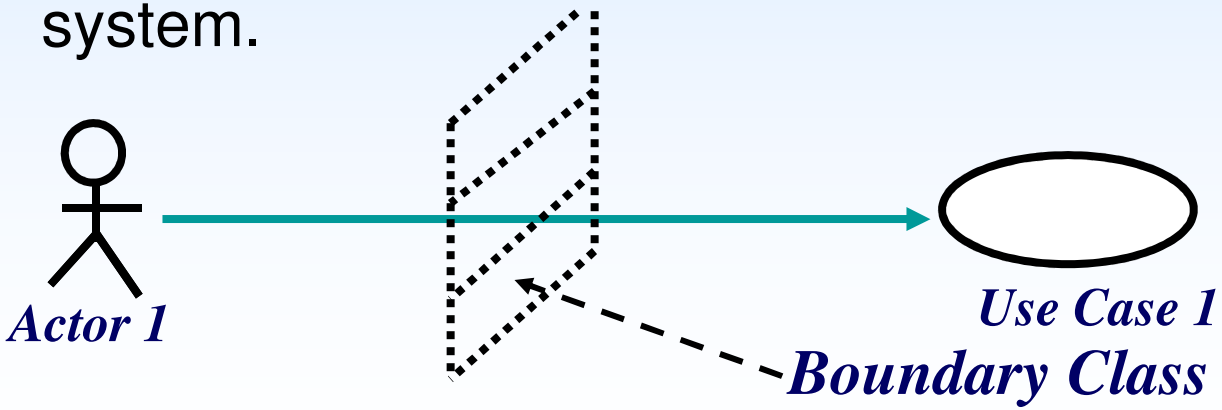
- They provide the interface to a user or another system. (ie. Interface to an actor).
- Handles communication between system surroundings and the inside of the system.
- To find the *Boundary* classes, you can examine your Use Case diagram,

Stereotypes and Classes

***Boundary** Class:*



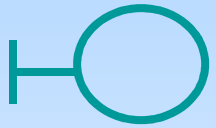
- At a minimum there must be, one *Boundary* class for every actor-use case interaction.
- Boundary class allows actor to interact with the system.



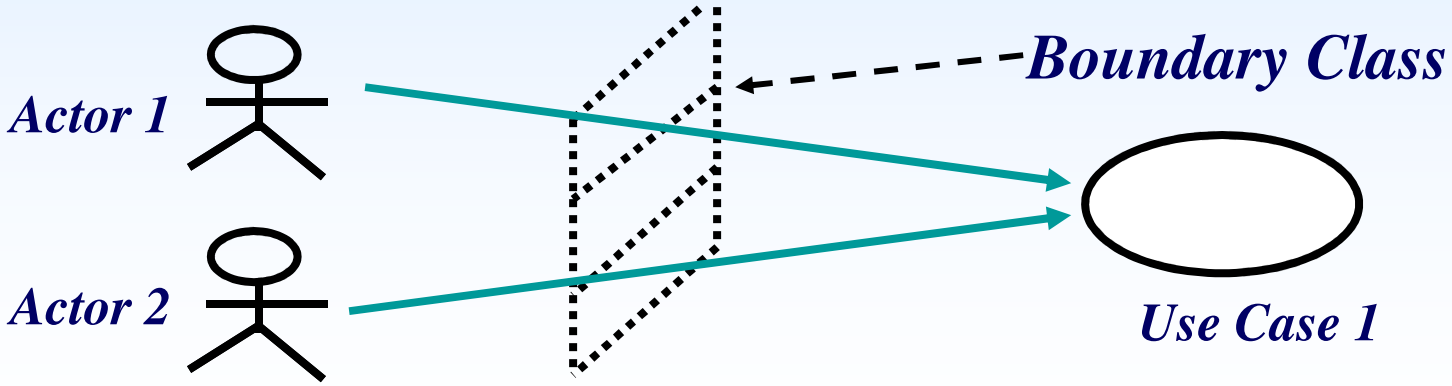
UCSC

Stereotypes and Classes

***Boundary** Class:*



- You do not necessarily have to create a unique *Boundary* class for every actor-use case pair.
- Two actors may initiate the same use case.
- They might both use the same Boundary class to communicate with the system.



UCSC

Finding Boundary Classes

- These are classes that mediate between the subject (System boundary) and its environment.
  - User Interface class – classes that interface between the system and humans;
  - System Interface class – classes that interface with other systems;
  - Device Interface class – classes that interface with external devices such as sensors;

Stereotypes and Classes

Entity Class

- They are needed to perform task internal to the system. Reflect a real world entity.

Identifying Entity Classes

- Identify the nouns and noun phrases used to describe the responsibilities.

Stereotypes and Classes

Entity Class

- The initial list of nouns must be filtered because,
  - it could contain nouns that are outside the problem domain.
  - nouns that are just language expressions.
  - nouns that are redundant.
  - nouns that are attributes.

Stereotypes and Classes cont..

Control Class:

- Sequencing behaviour specific to one or more use cases.
- There is typically one *control* class per use case.
- Co-ordinates the events needed to realise the behaviour specified in the use case.

  Eg. Running or *executing* the use case.

Finding Controller Classes

- Simple behavior can often be distributed between Boundary or Entity classes
- Consider more complex behavior of the system as described by the use cases.
- Work out how these behavior should be partitioned among the analysis classes.
- Control classes process messages from an interface class and respond to them by sending and receiving messages from the entity classes.

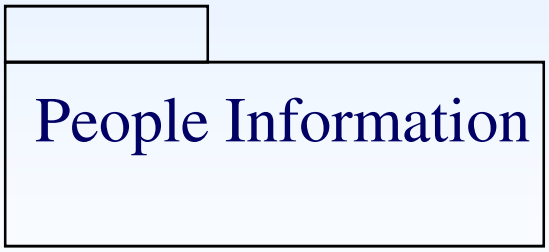
Package Diagrams

Purpose of a Package

- If a system contained only a few classes, you could manage them easily.
- Most systems are composed of many classes.
- Packages are used to group them together for ease of use, maintainability, and reusability.

Package Diagrams

- By grouping classes to *packages* we can look at the higher level view of the model.
- Package diagrams help you to maintain control over a system's overall structure

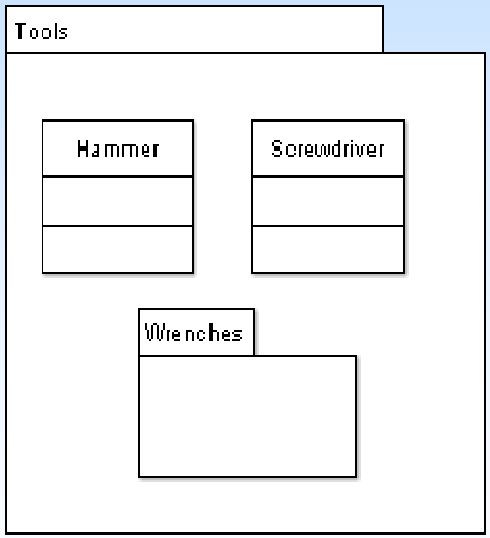


UML Notation

Package Diagrams

- Surround the grouped elements with a tabbed-folder icon.
- To reference an element in a package the notation is PackageName::PackageElement

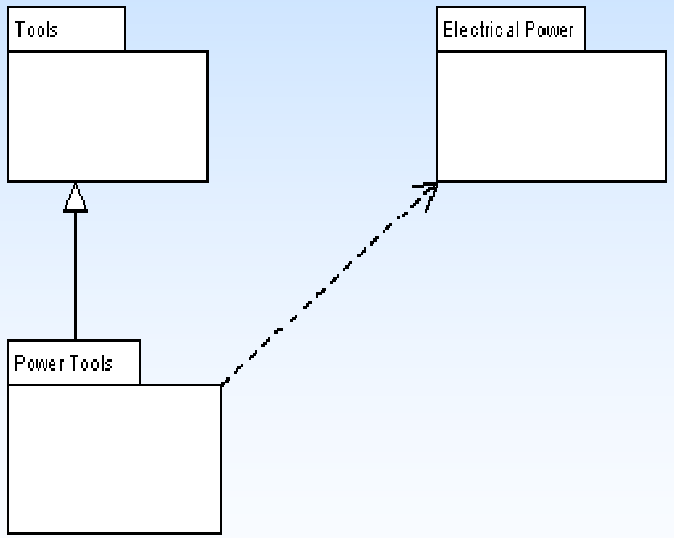
  e.g. Tools::Hammer



UCSC

Inter-package Relationships

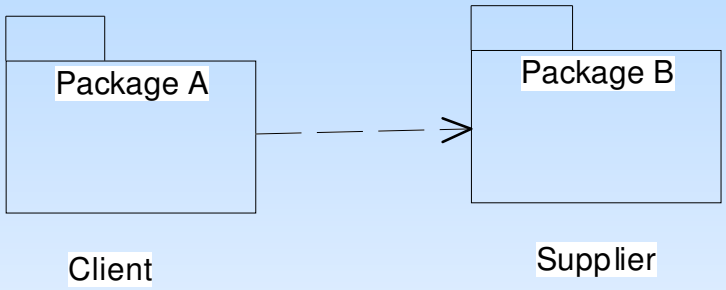
- Packages can generalize another, depend on another, or refine another
- Relationship type is a dependency relationship.
- It is shown as a dashed arrow to the dependent package.



UCSC

Inter-package Relationships

• If package A depend on package B:



- **One or more classes in package A initiates communication with one or more public classes in package B.**
- **Package A is referred to as the Client package, whereas package B is referred to as the Supplier package.**

Discovered by examining the scenarios and class relationships.

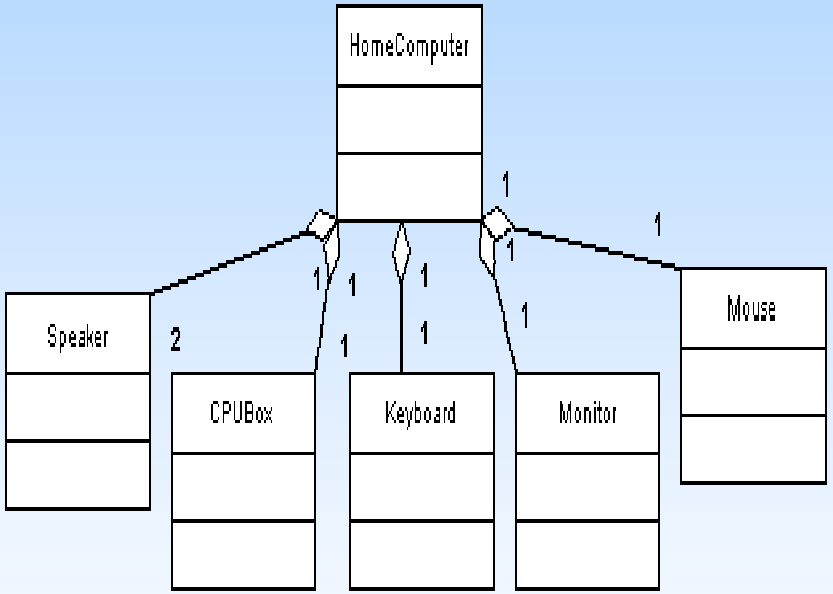
UCSC

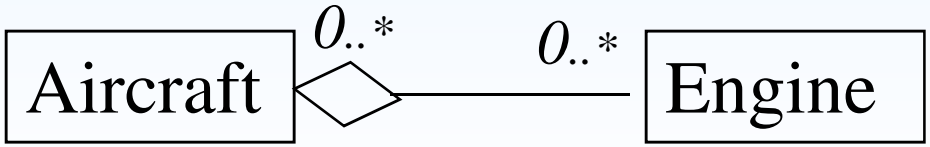
Merging Packages

- A package can be merged with another.
- The merge relationship is a kind of dependency between the package that does the merging (the source) and the package that gets merge.
- The result of a merge is that the source package is transformed.

Aggregation

- An aggregation is a stronger form of association.
- It is a relationship between a whole and its parts or composition.



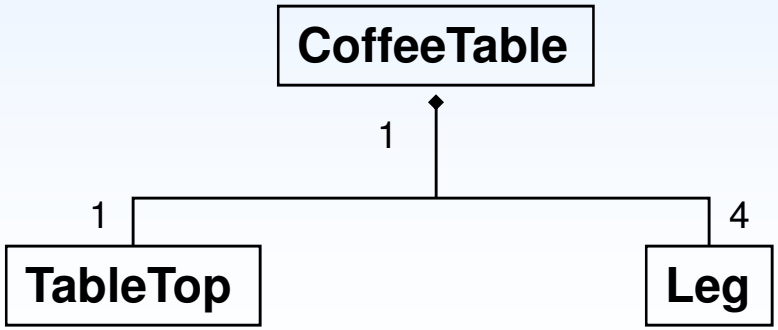


*** Removed in UML 2.x

UCSC

Composites

- A strong type of aggregation
- Each component in composite can belong to just one whole
- Can be denoted using a filled diamond.



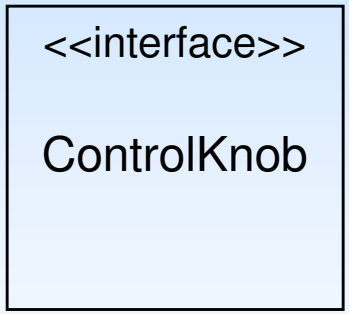
UCSC

Interfaces and Realizations

- Interface: a set of operations that specifies some aspect of a class's behavior, and it's operations a class presents to other classes.
- Realization: the relationship between a class and its interface

Interfaces and Realizations

- Modeling an interface is similar to modeling a class
- Interface can be modeled using a rectangle icon.
  - This icon has no attributes.
  - Add the keyword <<interface>>above the name of the interface in the rectangle.



UCSC

Interfaces and Realizations

- The symbol for the realization relationship between a class and its interface looks like the symbol for inheritance, except the line to the open triangle is dashed instead of solid.