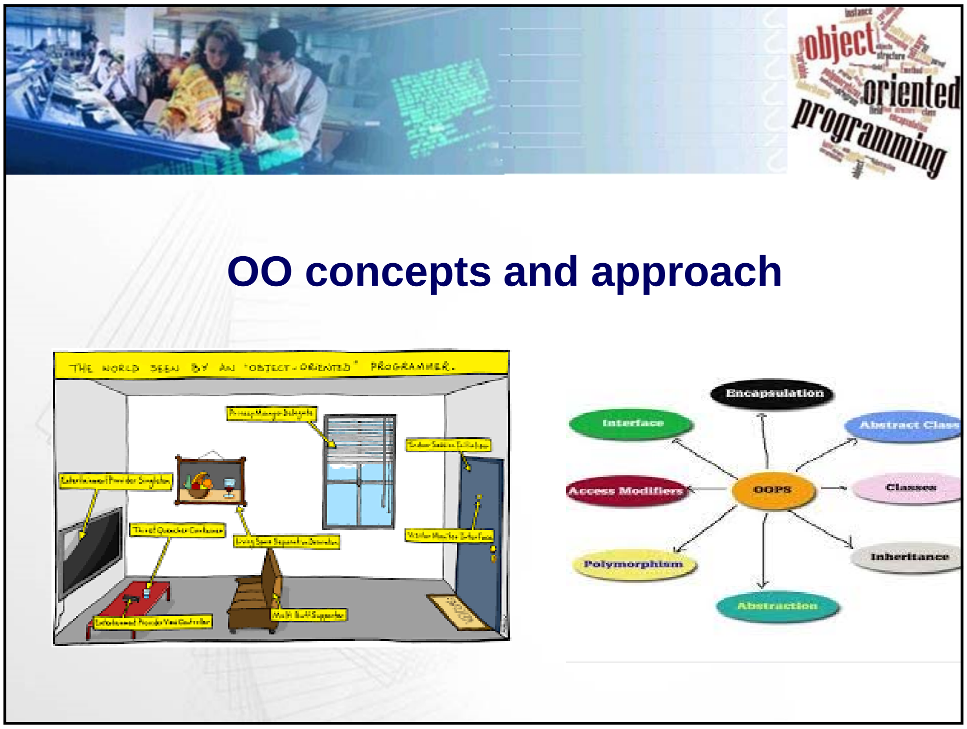
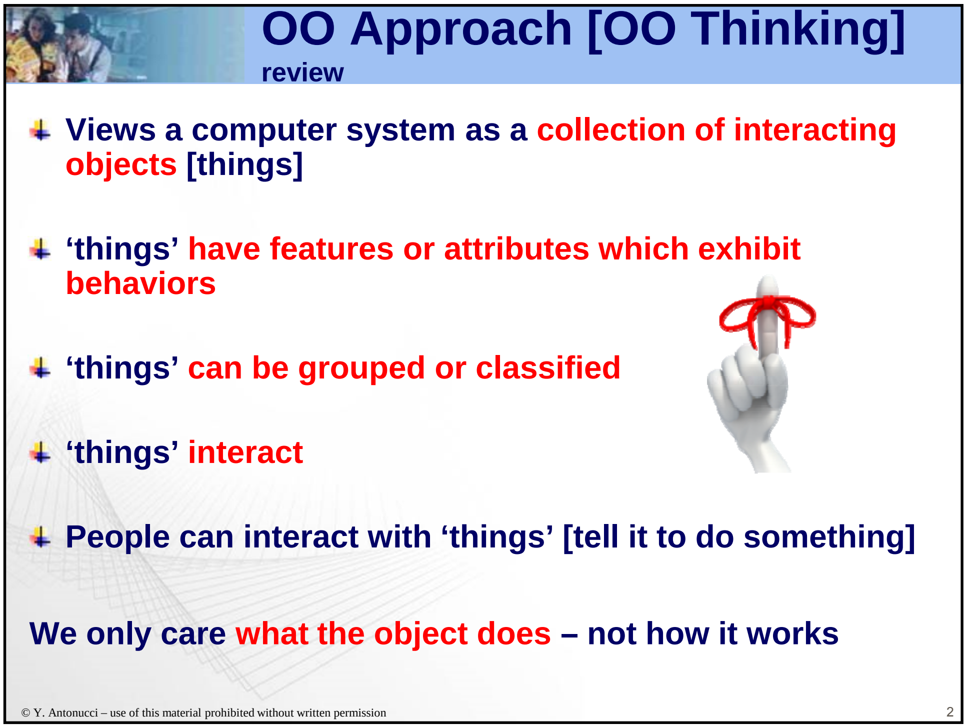


object oriented programming
THE WORLD SEEN BY AN "OBJECT-ORIENTED" PROGRAMMER.
Privacy Manager Delegate
Indoor Sunshine Facilitator
Entertainment Provider Singleton
Thirst Quencher Container
Living Space Separation Decorator
Visitor Monitor Interface
Entertainment Provider View Controller
Multi Butt Supporter
Encapsulation
Interface
Abstract Class
Access Modifiers
OOPS
Classes
Polymorphism
Inheritance
Abstraction

## OO concepts and approach

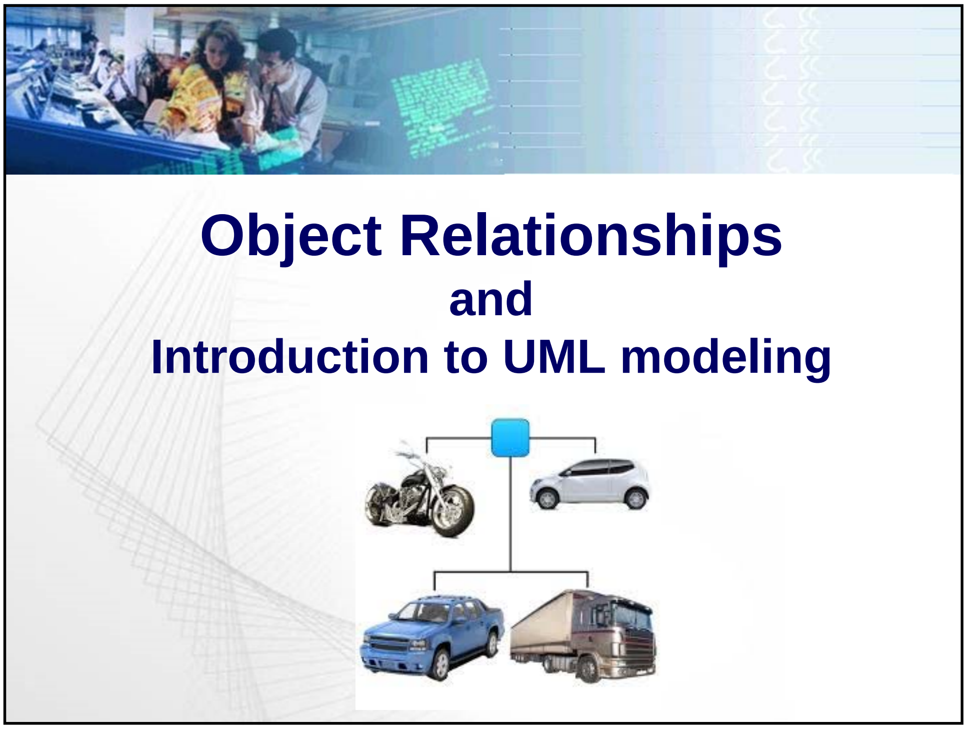


## OO Approach [OO Thinking]

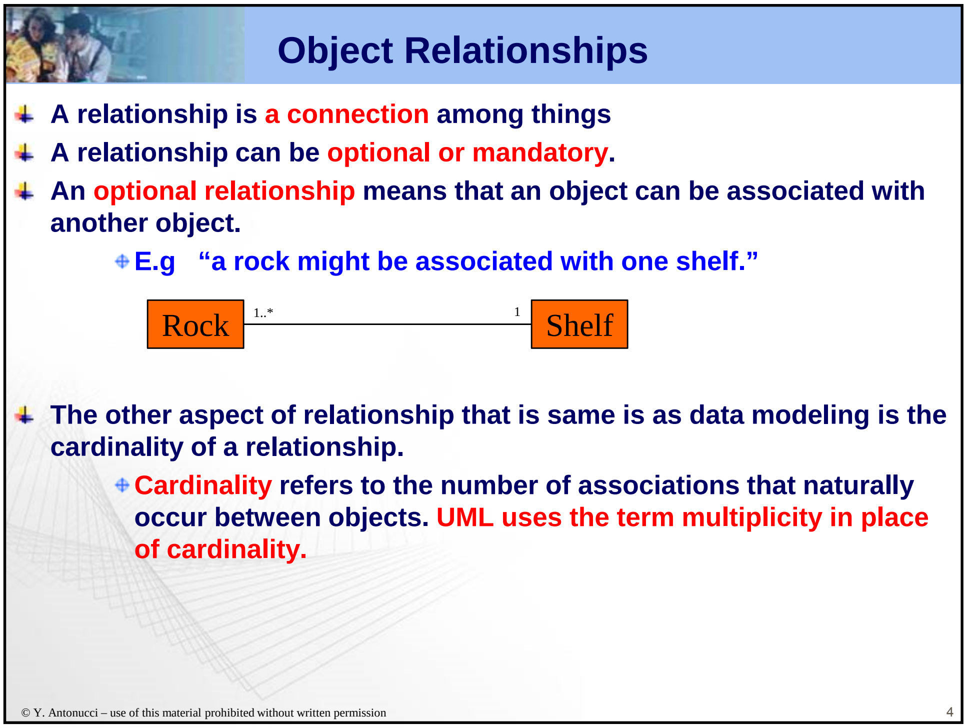
review

- Views a computer system as a collection of interacting objects [things]
- 'things' have features or attributes which exhibit behaviors
- 'things' can be grouped or classified
- 'things' interact
- People can interact with 'things' [tell it to do something]

We only care what the object does – not how it works



# Object Relationships
## and
## Introduction to UML modeling

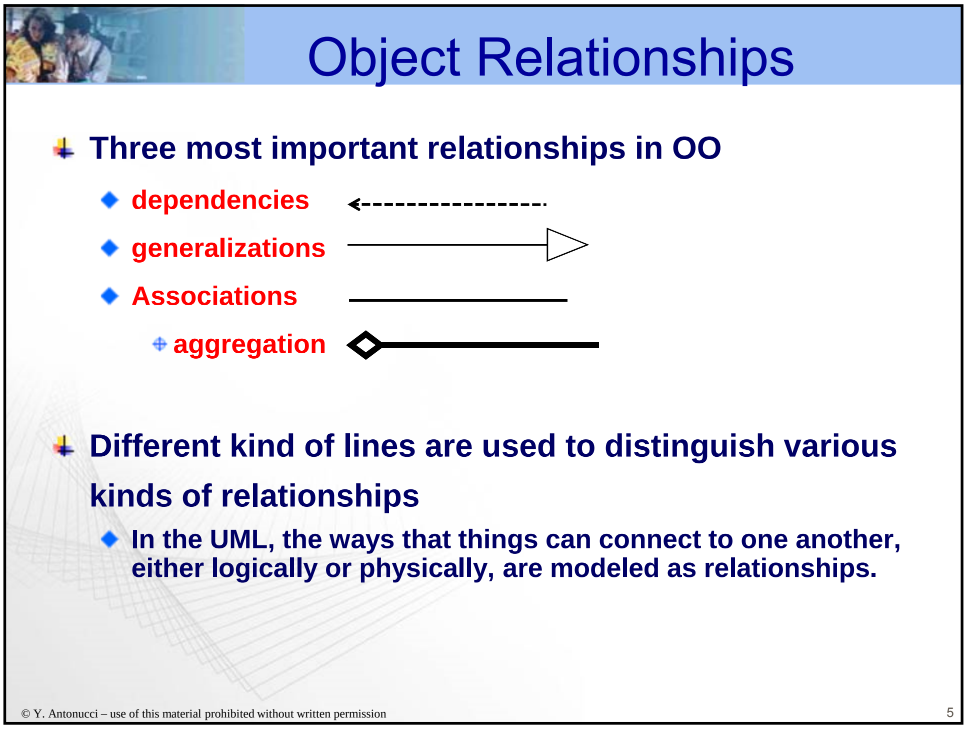


## Object Relationships

- A relationship is a connection among things
- A relationship can be optional or mandatory.
- An optional relationship means that an object can be associated with another object.
  - E.g "a rock might be associated with one shelf."

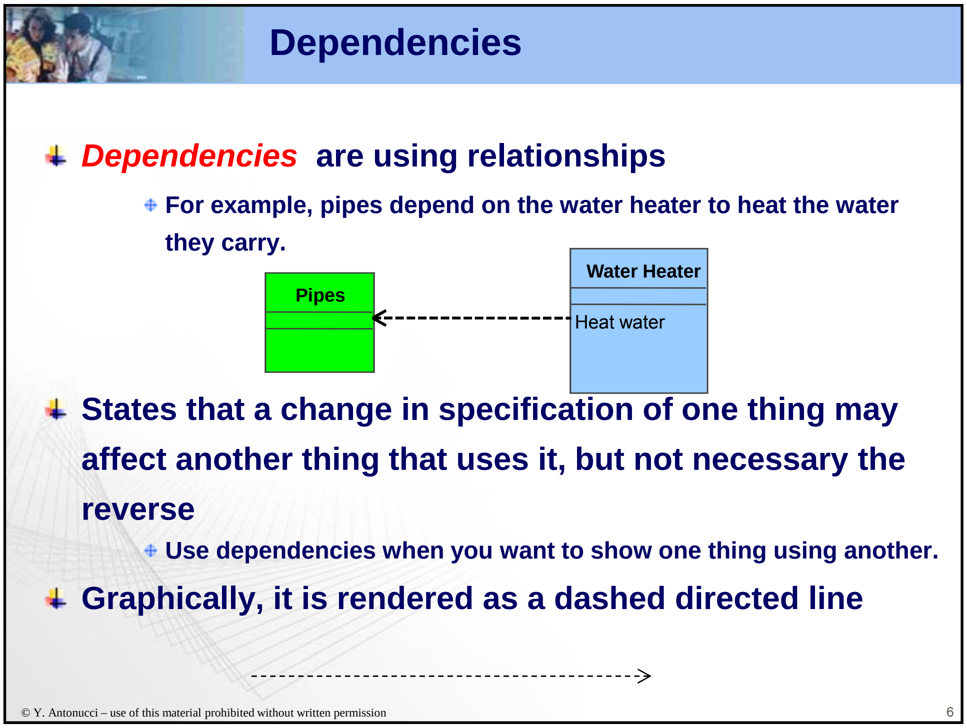
Rock 1..* — 1 Shelf

- The other aspect of relationship that is same is as data modeling is the cardinality of a relationship.
  - Cardinality refers to the number of associations that naturally occur between objects. UML uses the term multiplicity in place of cardinality.



## Object Relationships

- Three most important relationships in OO
  - dependencies
  - generalizations
  - Associations
    - aggregation
- Different kind of lines are used to distinguish various kinds of relationships
  - In the UML, the ways that things can connect to one another, either logically or physically, are modeled as relationships.

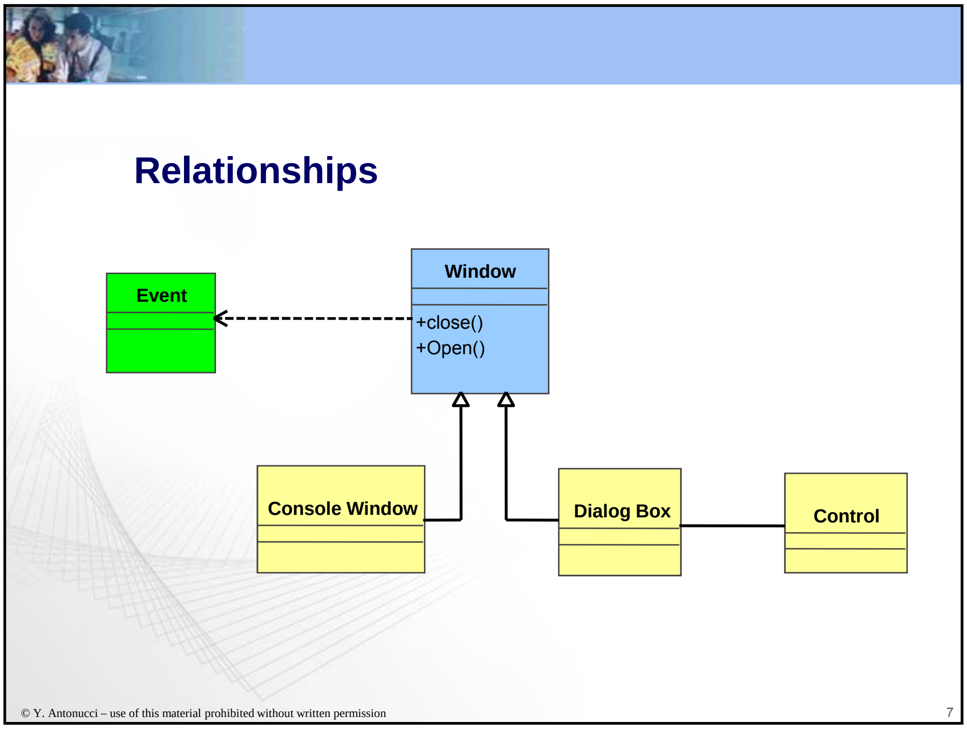


## Dependencies

- *Dependencies* are using relationships
  - For example, pipes depend on the water heater to heat the water they carry.

Pipes
Water Heater
Heat water

- States that a change in specification of one thing may affect another thing that uses it, but not necessary the reverse
  - Use dependencies when you want to show one thing using another.
- Graphically, it is rendered as a dashed directed line



## Relationships

Window

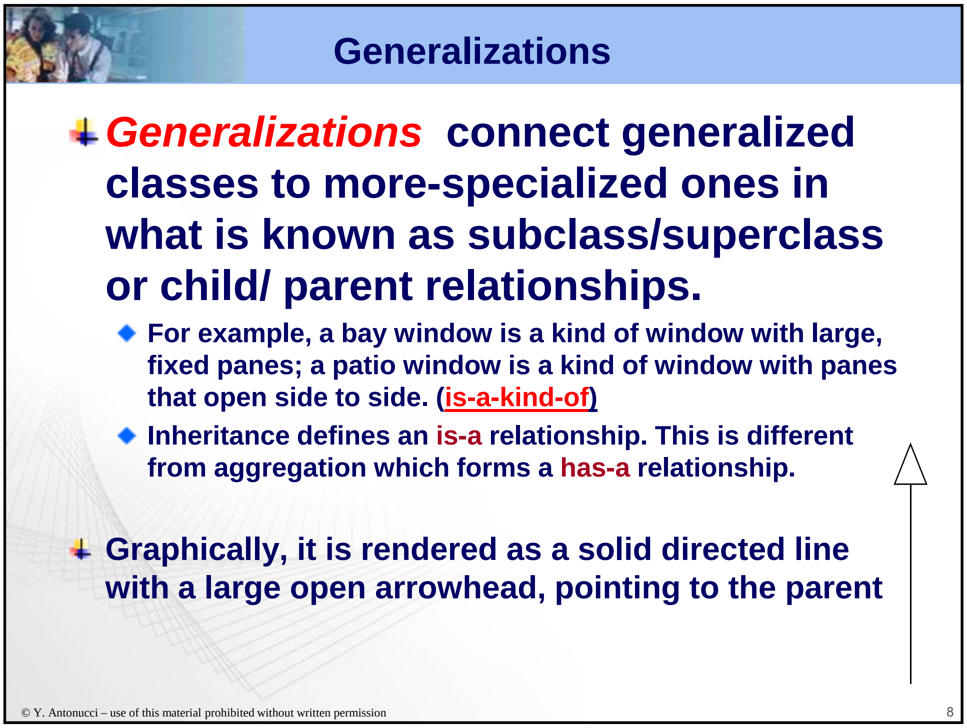
+close()
+Open()

Event

Console Window

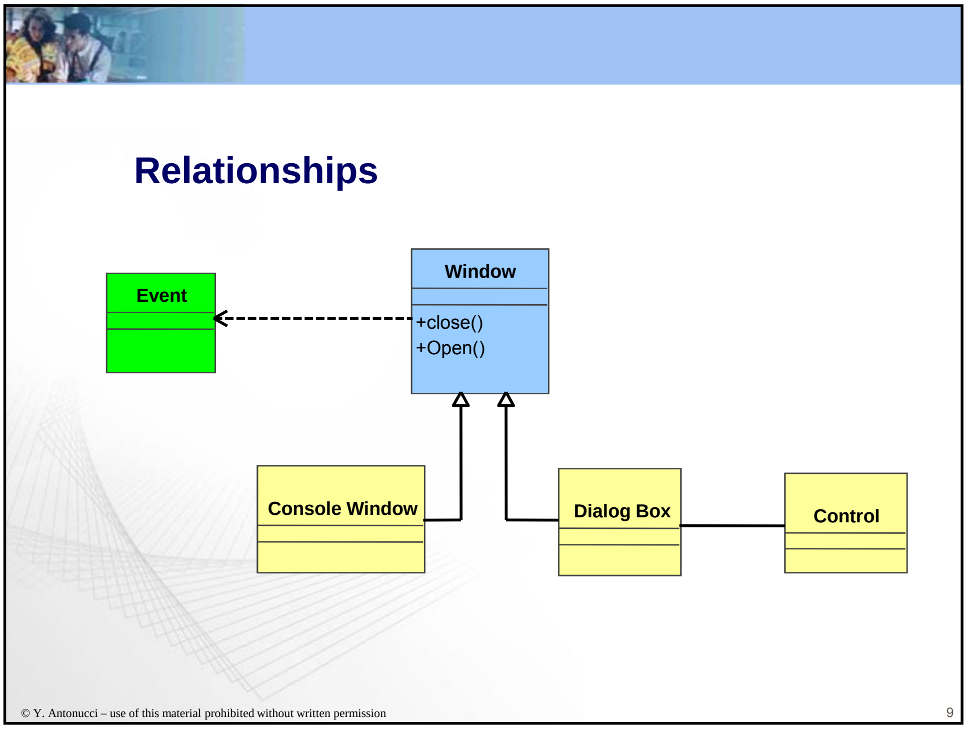
Dialog Box

Control

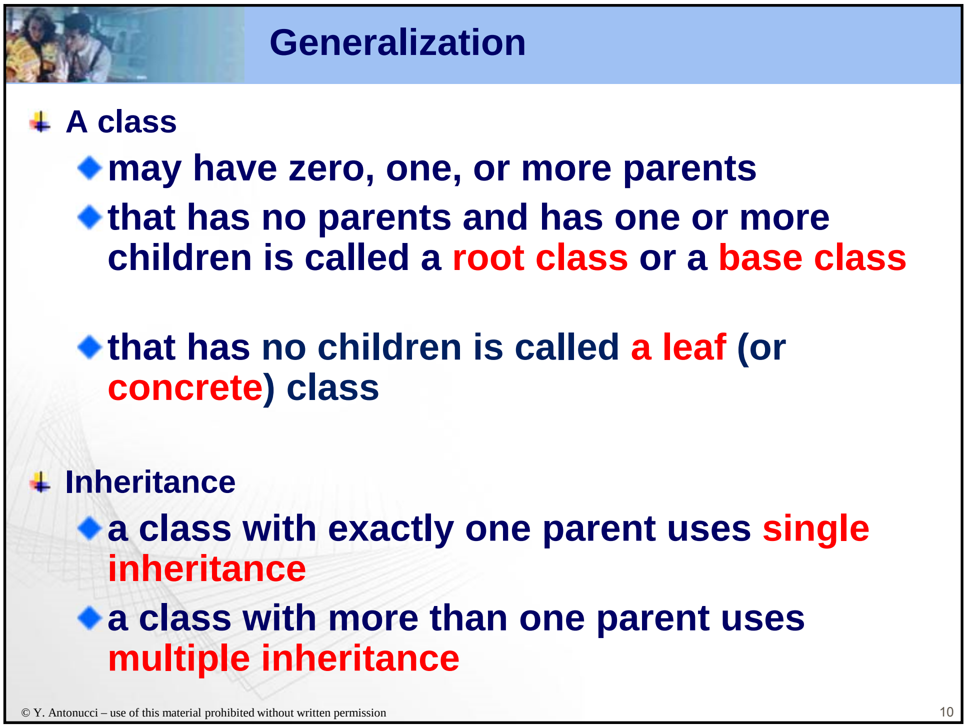


## Generalizations

- *Generalizations* connect generalized classes to more-specialized ones in what is known as subclass/superclass or child/ parent relationships.
  - For example, a bay window is a kind of window with large, fixed panes; a patio window is a kind of window with panes that open side to side. (is-a-kind-of)
  - Inheritance defines an is-a relationship. This is different from aggregation which forms a has-a relationship.
- Graphically, it is rendered as a solid directed line with a large open arrowhead, pointing to the parent

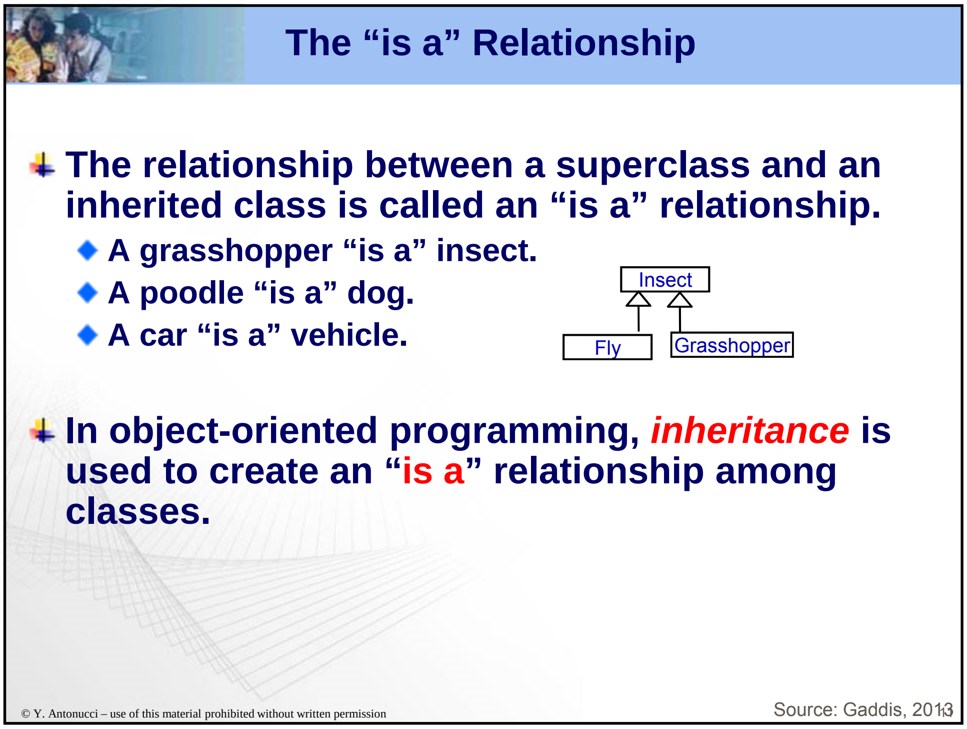


## Relationships



## Generalization

- A class
  - may have zero, one, or more parents
  - that has no parents and has one or more children is called a root class or a base class
  - that has no children is called a leaf (or concrete) class
- Inheritance
  - a class with exactly one parent uses single inheritance
  - a class with more than one parent uses multiple inheritance

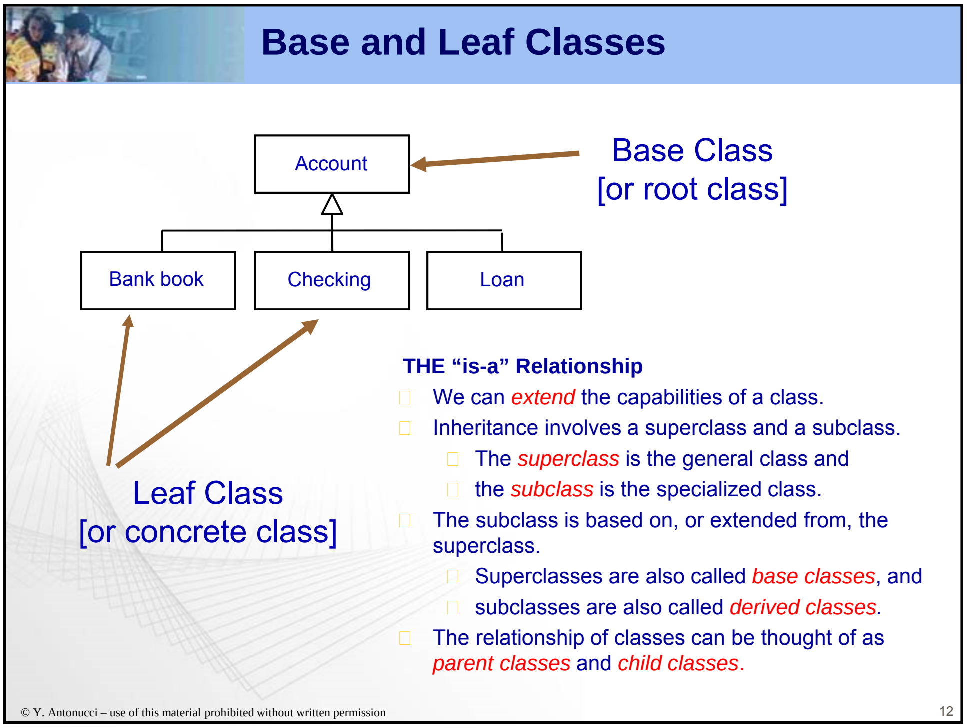


## The “is a” Relationship

- The relationship between a superclass and an inherited class is called an “is a” relationship.
  - A grasshopper “is a” insect.
  - A poodle “is a” dog.
  - A car “is a” vehicle.

- In object-oriented programming, *inheritance* is used to create an “is a” relationship among classes.

Source: Gaddis, 2013

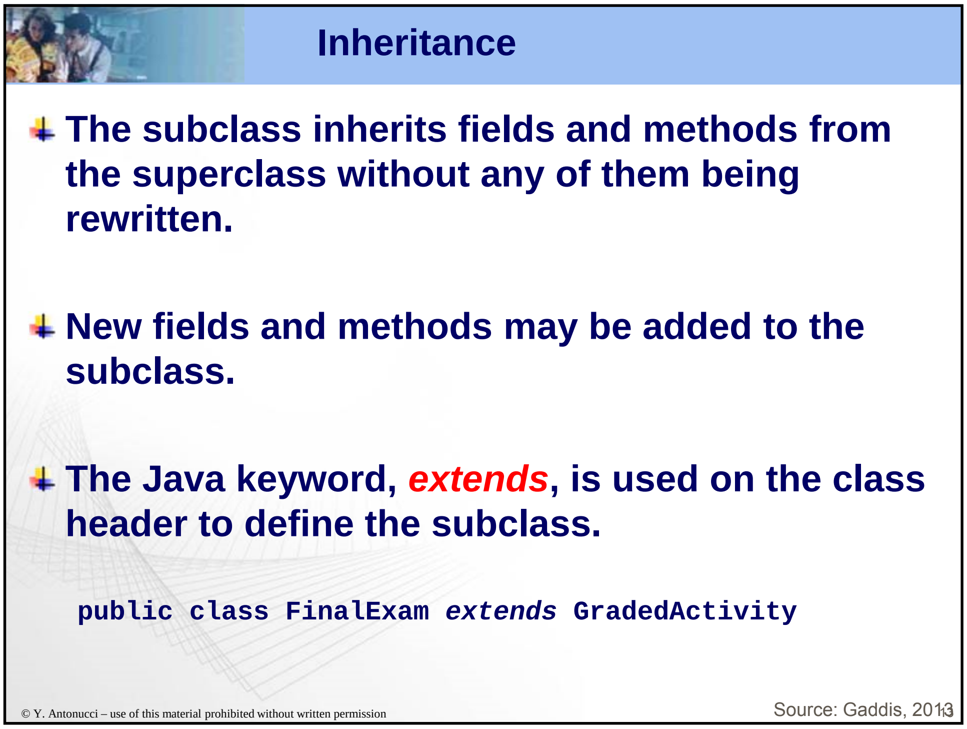


## Base and Leaf Classes

**THE “is-a” Relationship**

- We can *extend* the capabilities of a class.
- Inheritance involves a superclass and a subclass.
  - The *superclass* is the general class and
  - the *subclass* is the specialized class.
- The subclass is based on, or extended from, the superclass.
  - Superclasses are also called *base classes*, and
  - subclasses are also called *derived classes*.
- The relationship of classes can be thought of as *parent classes* and *child classes*.

12



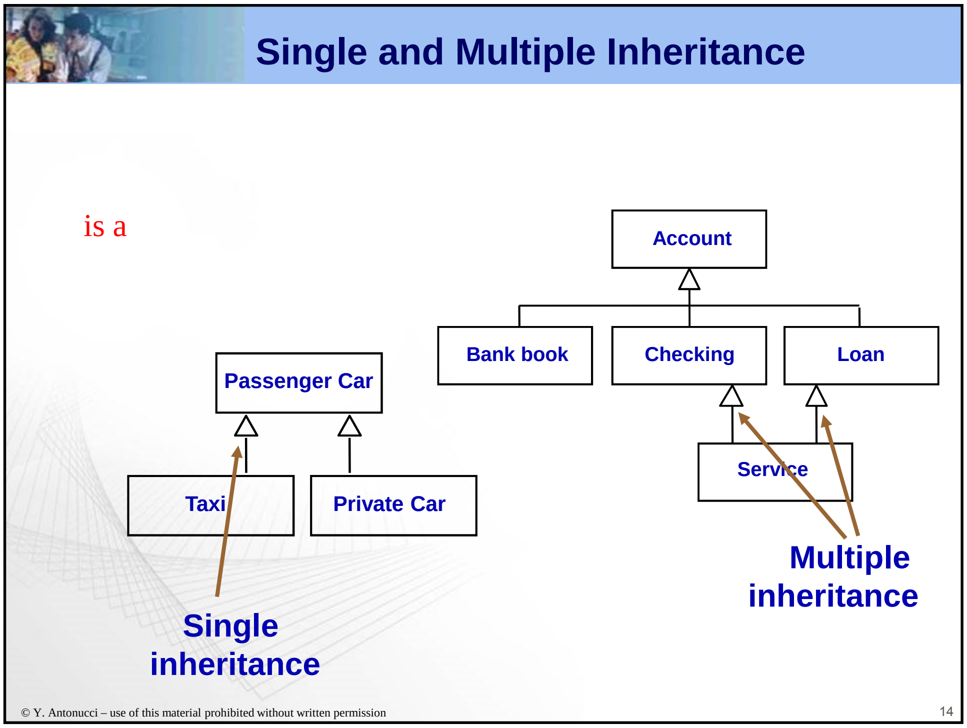
## Inheritance

- The subclass inherits fields and methods from the superclass without any of them being rewritten.
- New fields and methods may be added to the subclass.
- The Java keyword, *extends*, is used on the class header to define the subclass.

```
public class FinalExam extends GradedActivity
```

© Y. Antonucci – use of this material prohibited without written permission

Source: Gaddis, 2013



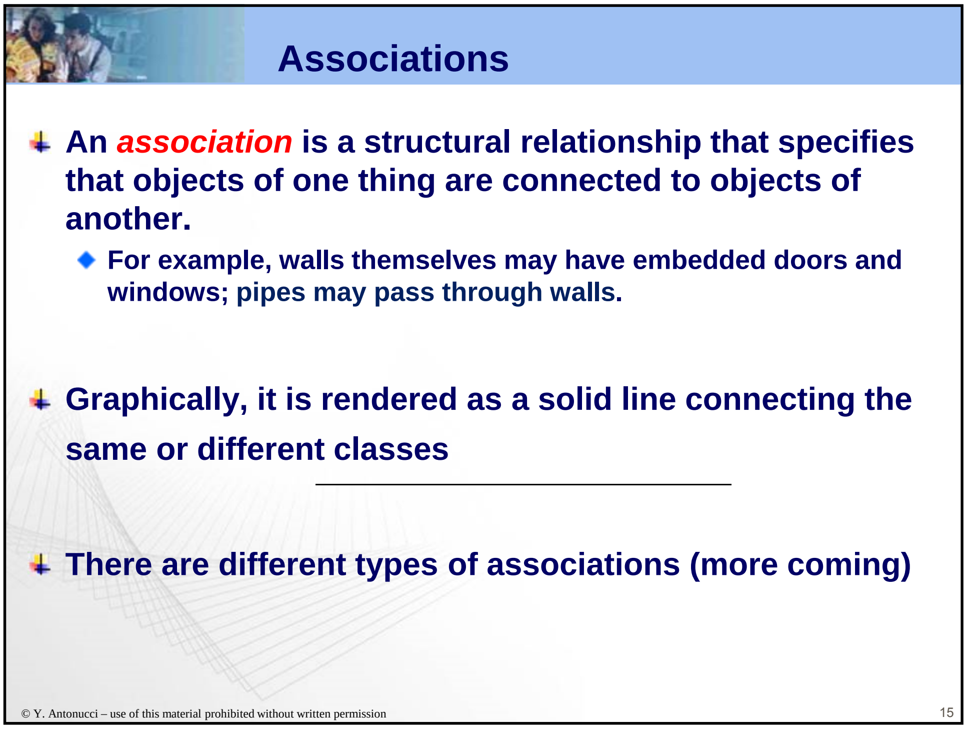
## Single and Multiple Inheritance

is a

Multiple inheritance

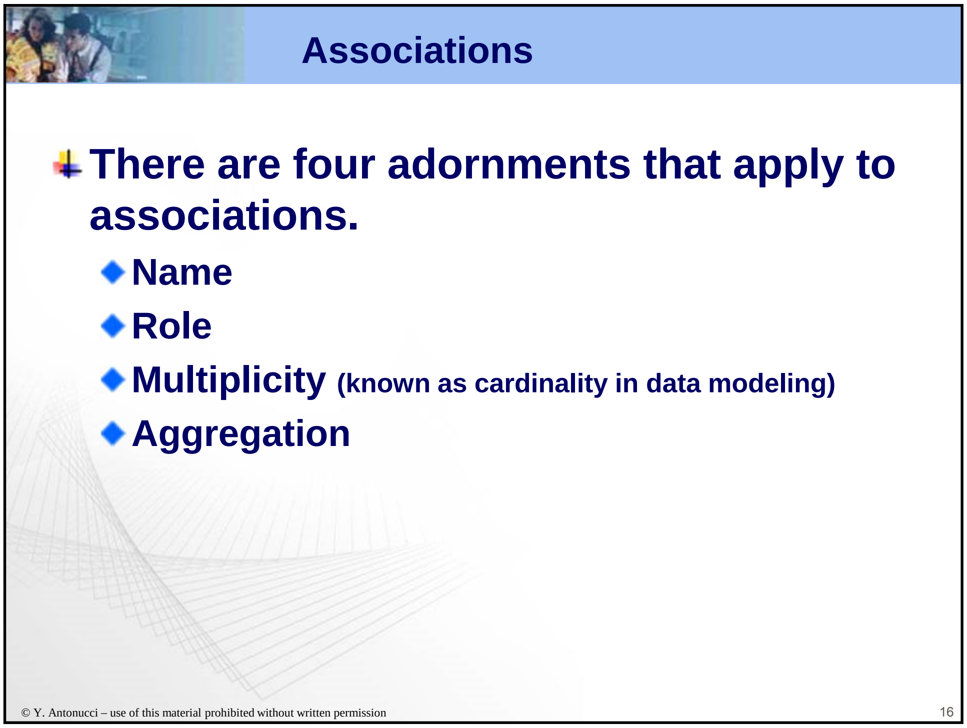
Single inheritance

© Y. Antonucci – use of this material prohibited without written permission



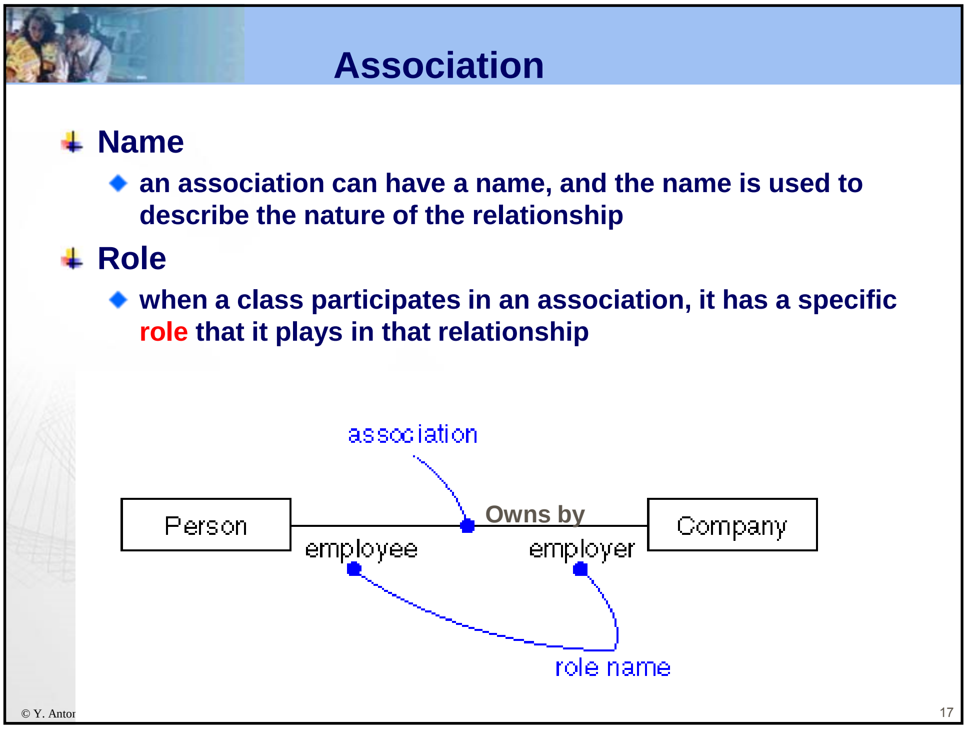
## Associations

- An *association* is a structural relationship that specifies that objects of one thing are connected to objects of another.
  - For example, walls themselves may have embedded doors and windows; pipes may pass through walls.
- Graphically, it is rendered as a solid line connecting the same or different classes
- There are different types of associations (more coming)



## Associations

- There are four adornments that apply to associations.
  - Name
  - Role
  - Multiplicity (known as cardinality in data modeling)
  - Aggregation



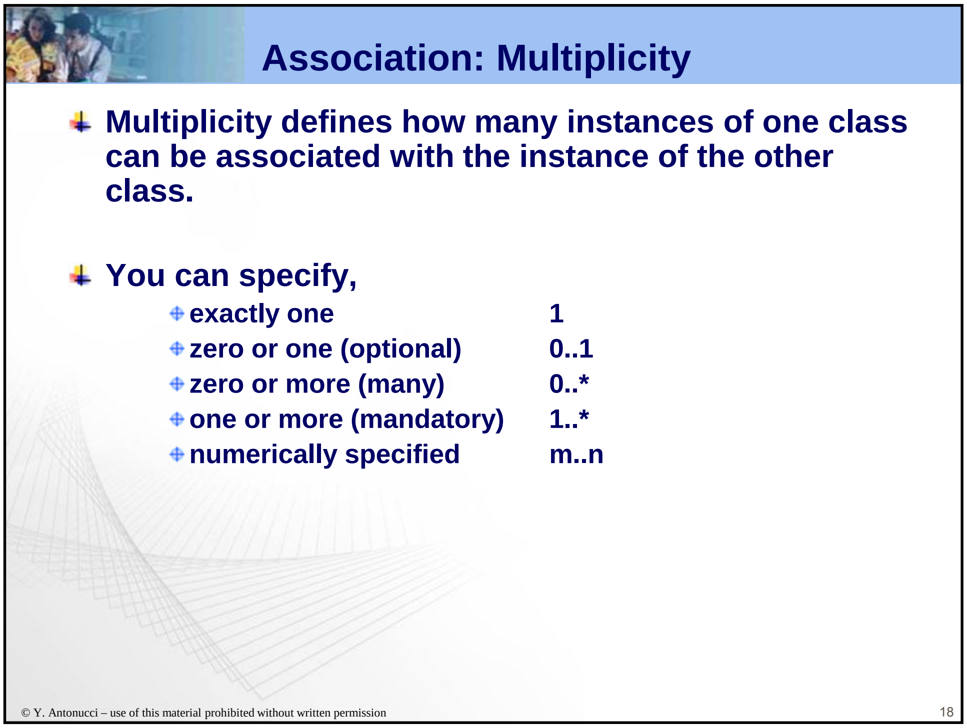
## Association

- **Name**
  - **an association can have a name, and the name is used to describe the nature of the relationship**
- **Role**
  - **when a class participates in an association, it has a specific role that it plays in that relationship**

association
Person
Owns by
Company
employee
employer
role name

© Y. Antor

17



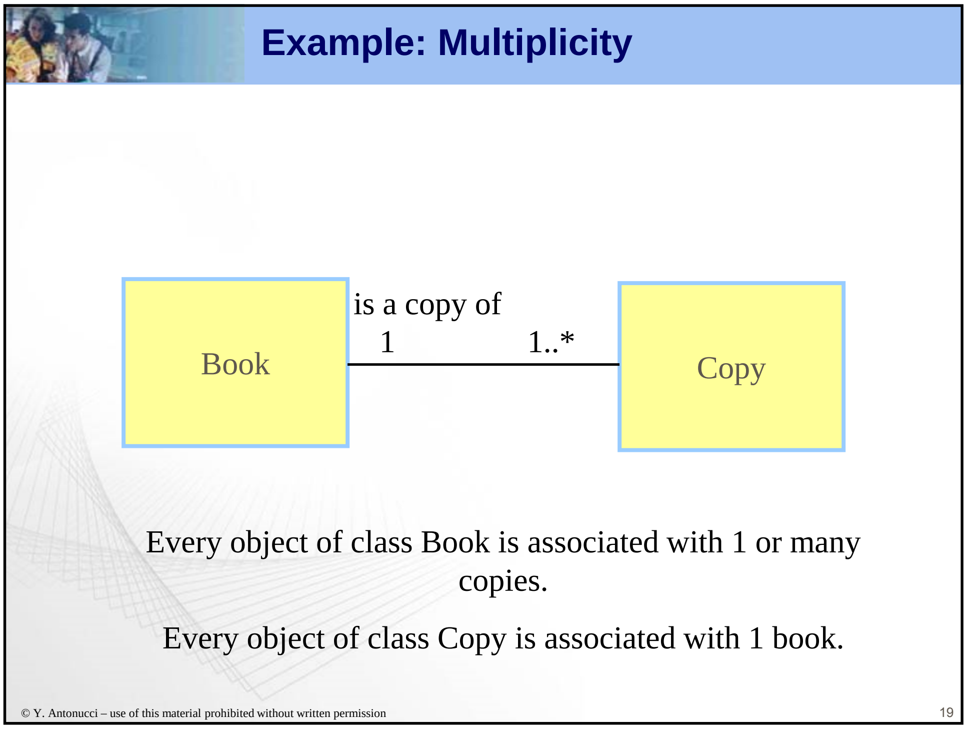
## Association: Multiplicity

- **Multiplicity defines how many instances of one class can be associated with the instance of the other class.**
- **You can specify,**

| | |
|---|---|
| **exactly one** | **1** |
| **zero or one (optional)** | **0..1** |
| **zero or more (many)** | **0..*** |
| **one or more (mandatory)** | **1..*** |
| **numerically specified** | **m..n** |

© Y. Antonucci – use of this material prohibited without written permission

18



## Example: Multiplicity

Book

is a copy of

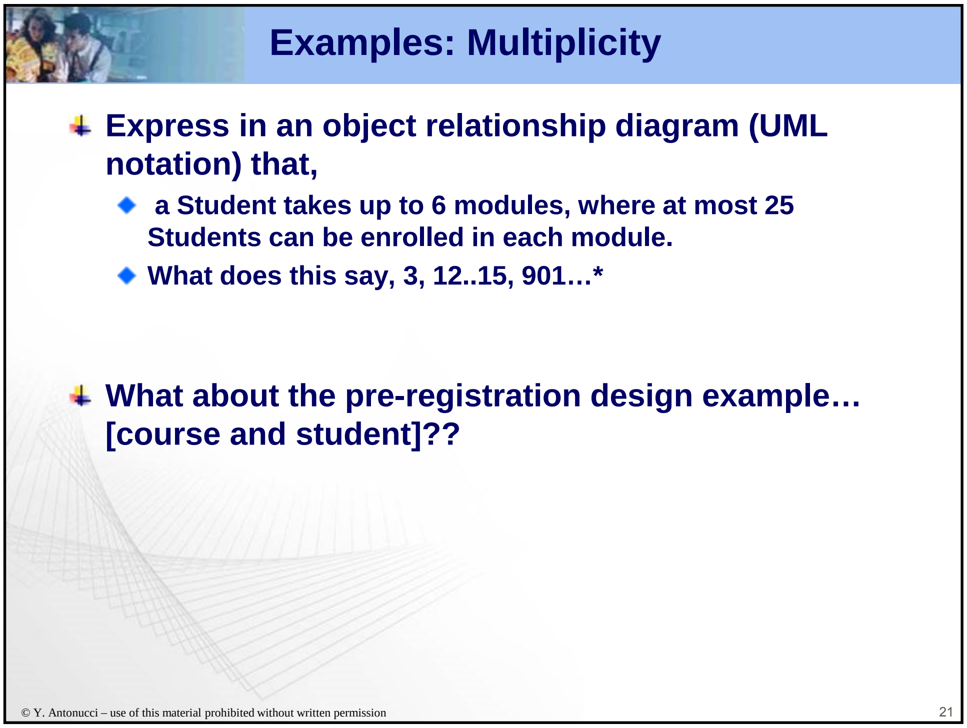
1

1..*

Copy

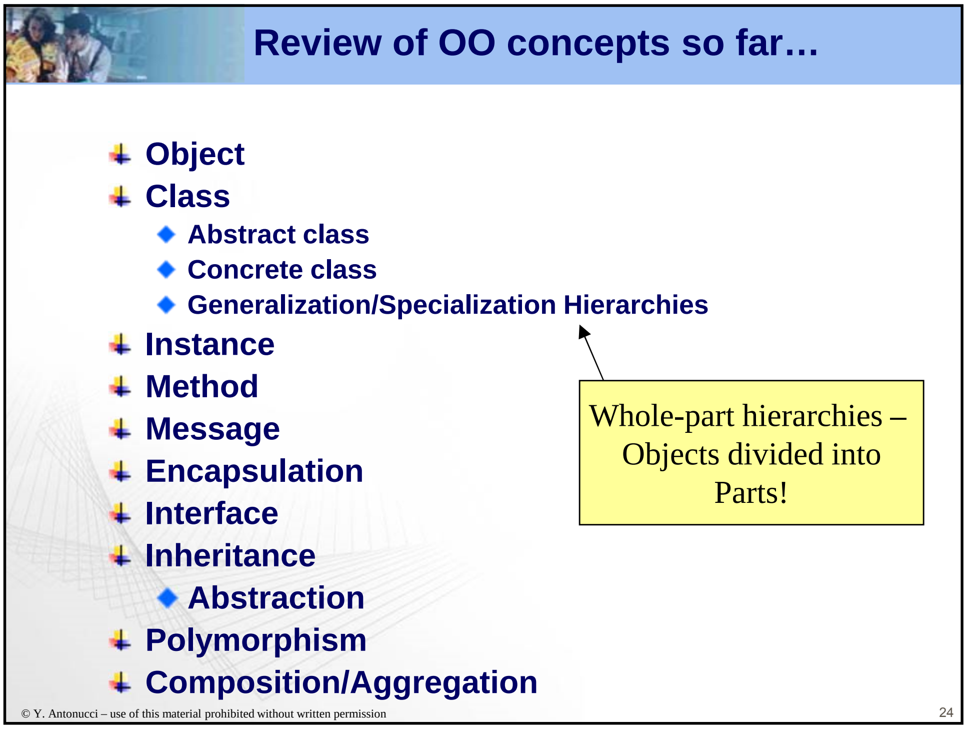
Every object of class Book is associated with 1 or many copies.

Every object of class Copy is associated with 1 book.



## Examples: Multiplicity

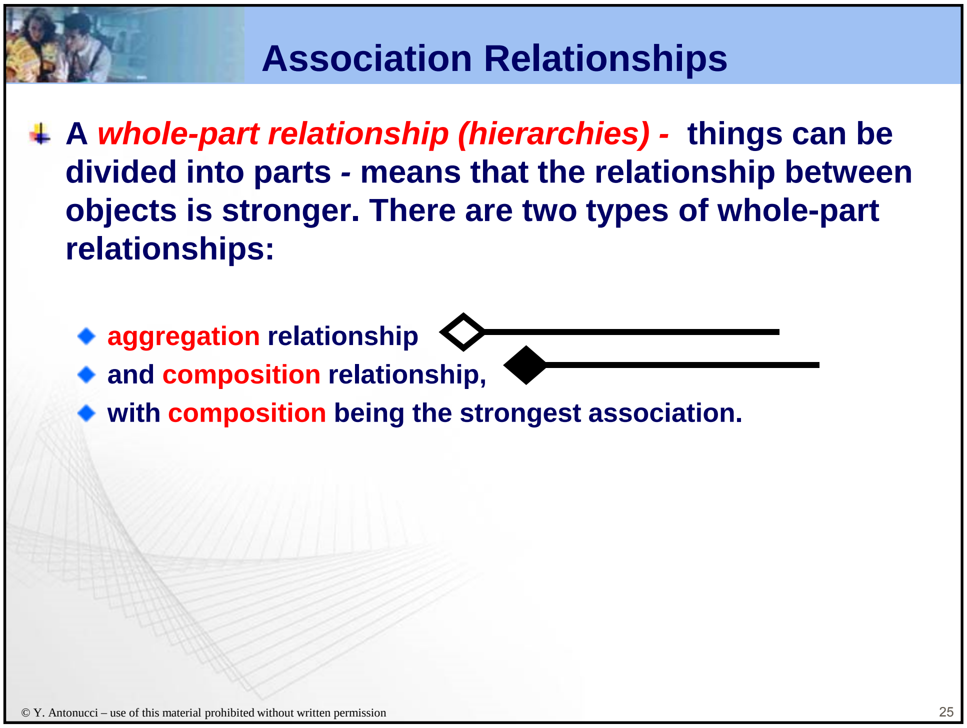
- Express in an object relationship diagram (UML notation) that,
  - a Student takes up to 6 modules, where at most 25 Students can be enrolled in each module.
  - What does this say, 3, 12..15, 901…*
- What about the pre-registration design example… [course and student]??



## Review of OO concepts so far…

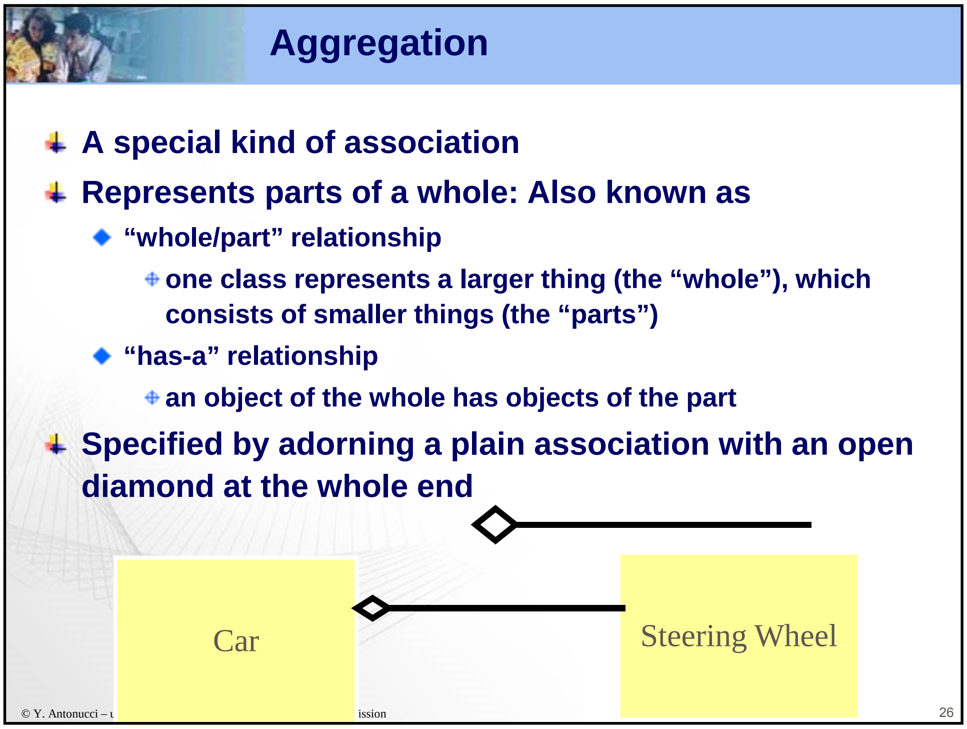
- Object
- Class
  - Abstract class
  - Concrete class
  - Generalization/Specialization Hierarchies
- Instance
- Method
- Message
- Encapsulation
- Interface
- Inheritance
  - Abstraction
- Polymorphism
- Composition/Aggregation

Whole-part hierarchies – Objects divided into Parts!



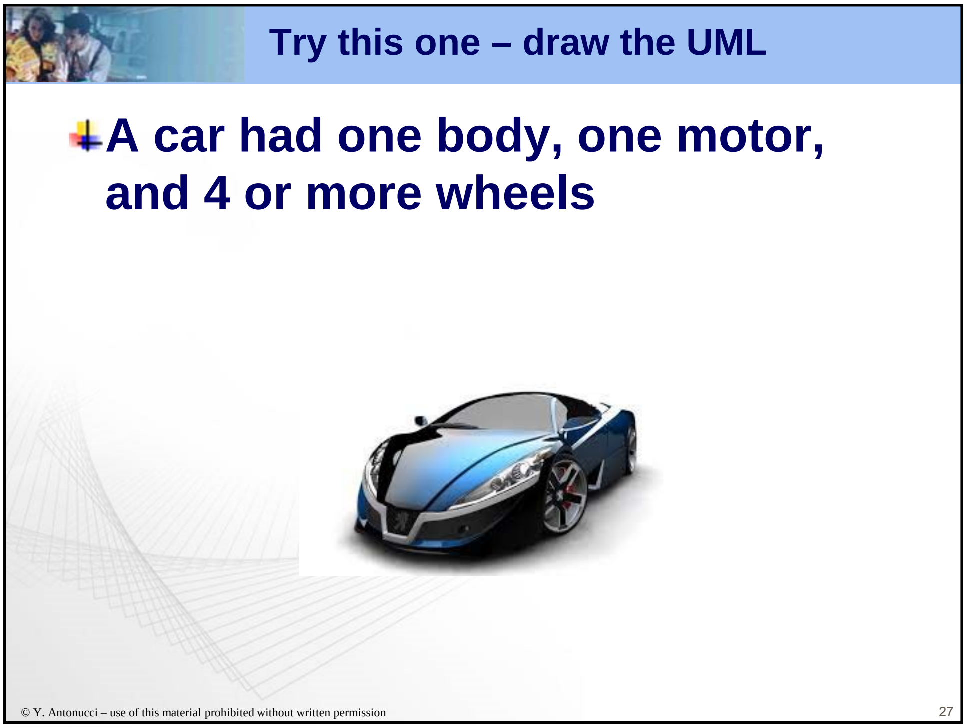
## Association Relationships

- A ***whole-part relationship (hierarchies) -*** things can be divided into parts - means that the relationship between objects is stronger. There are two types of whole-part relationships:
  - aggregation relationship ◇——
  - and composition relationship, ◆——
  - with composition being the strongest association.



## Aggregation

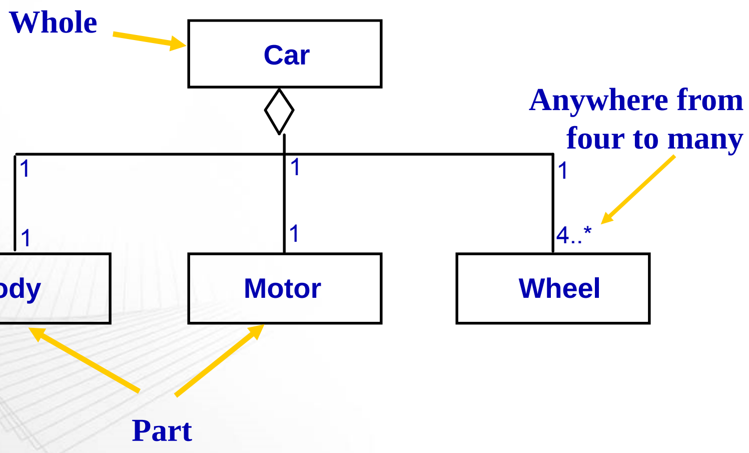
- A special kind of association
- Represents parts of a whole: Also known as
  - “whole/part” relationship
    - one class represents a larger thing (the “whole”), which consists of smaller things (the “parts”)
  - “has-a” relationship
    - an object of the whole has objects of the part
- Specified by adorning a plain association with an open diamond at the whole end



## Try this one – draw the UML

- A car had one body, one motor, and 4 or more wheels

## Aggregation



Whole

Car

Anywhere from four to many

1

1

1

1

1

4..*

Body

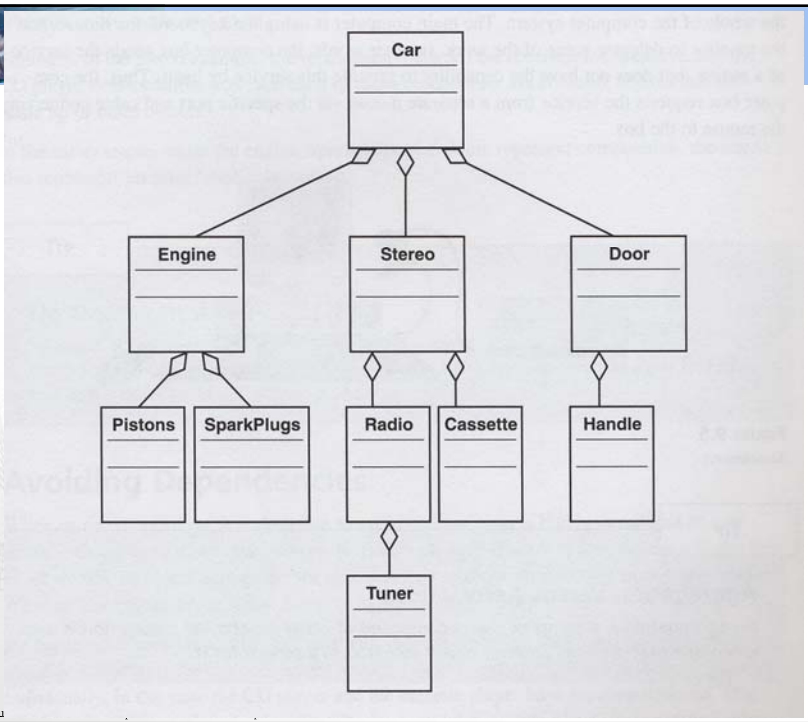
Motor

Wheel

Part

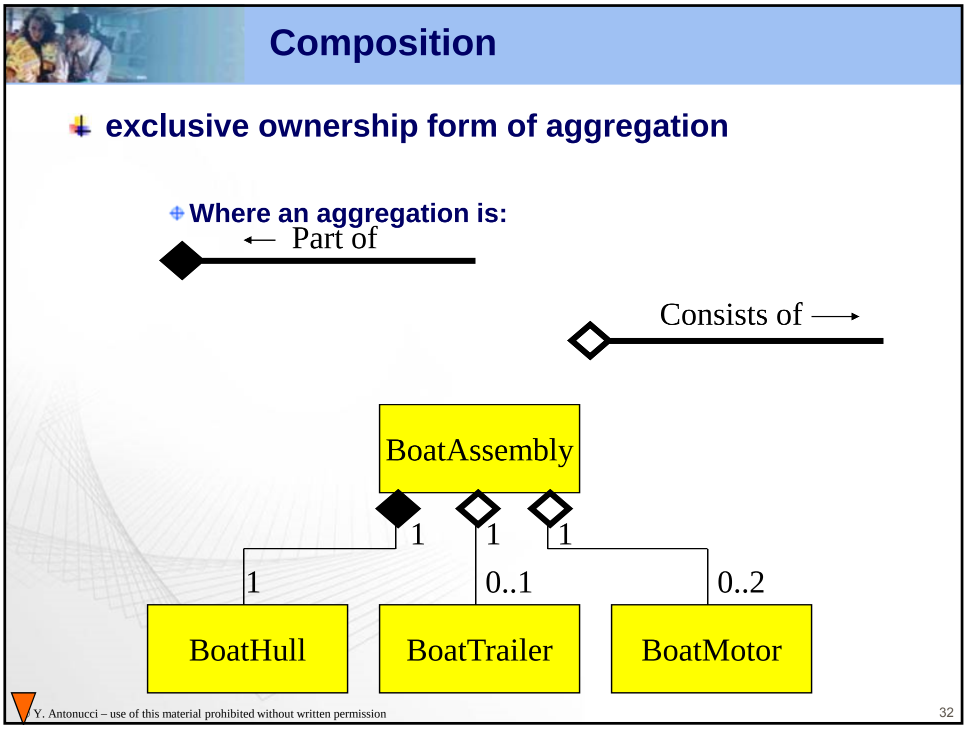
## Try this one….

- A car has an engine, a stereo, 4 doors.  The engine has many pistons and many sparkplugs.  The stereo includes a radio and a cassette, where the radio has a tuner.  Each door has a handle.



## Try this one – can you draw the UML?

- At a dive shop,
  - a boat assembly, not a boat, is rented
  - 1 boat assembly contains a boat hull, a motor, and a trailer
  - another boat assembly contains a boat hull and 2 motors, but no trailer
  - a 3rd boat assembly contains a boat hull and a trailer, but no motor



## Composition

- exclusive ownership form of aggregation
  - Where an aggregation is:

← Part of

Consists of →

BoatAssembly

1

1

1

1

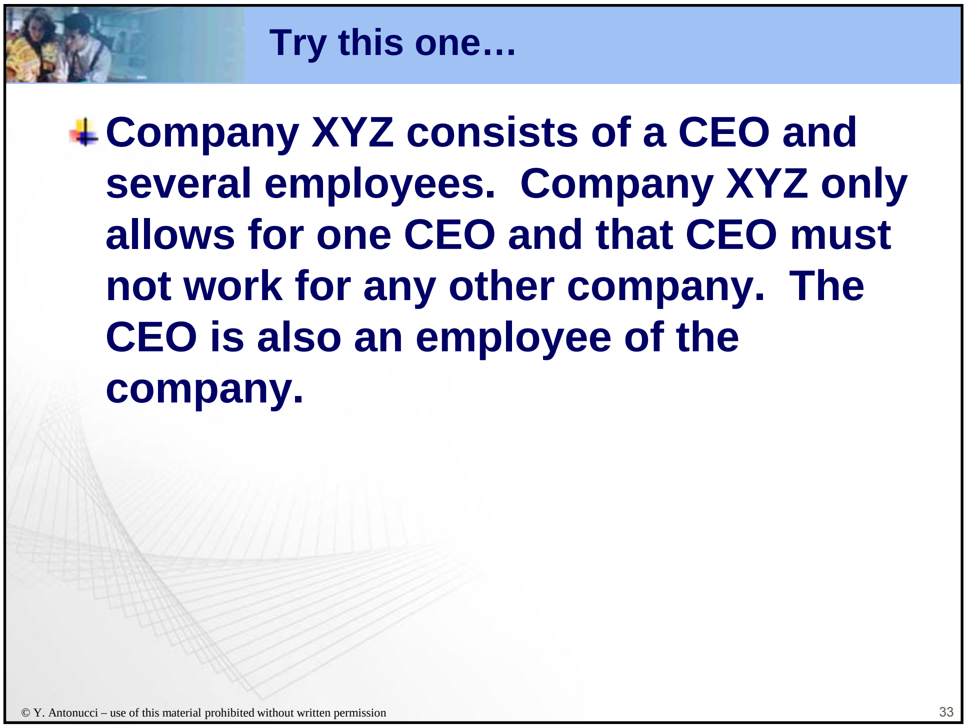
0..1

0..2

BoatHull

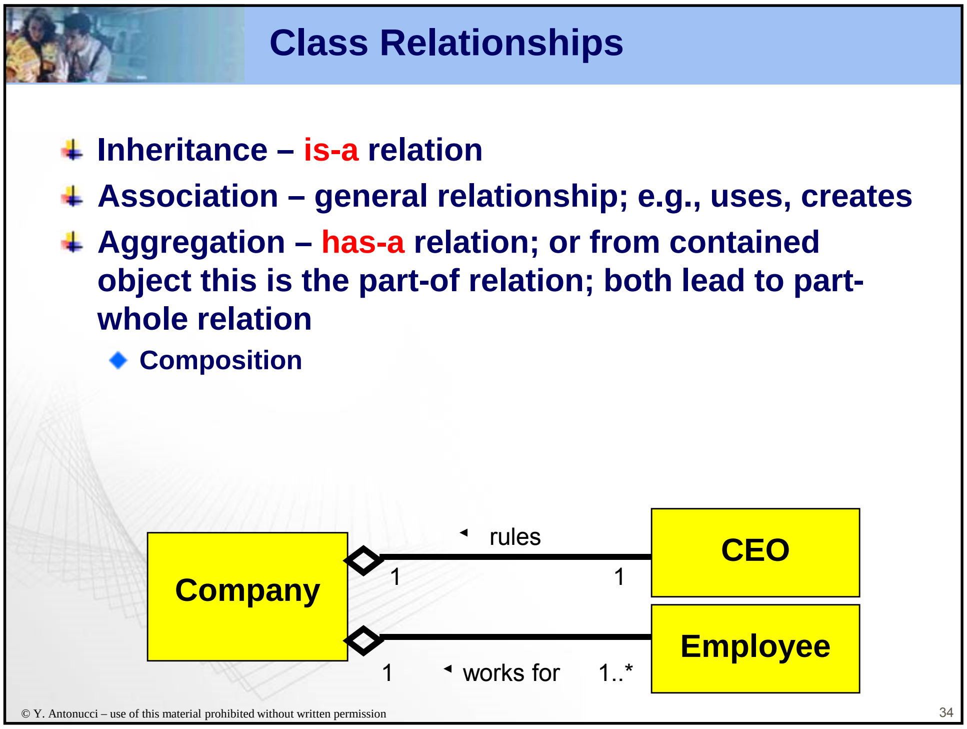
BoatTrailer

BoatMotor



## Try this one…

- Company XYZ consists of a CEO and several employees.  Company XYZ only allows for one CEO and that CEO must not work for any other company.  The CEO is also an employee of the company.



## Class Relationships

- Inheritance – is-a relation
- Association – general relationship; e.g., uses, creates
- Aggregation – has-a relation; or from contained object this is the part-of relation; both lead to part-whole relation
  - Composition

Company

◂ rules

1

1

CEO

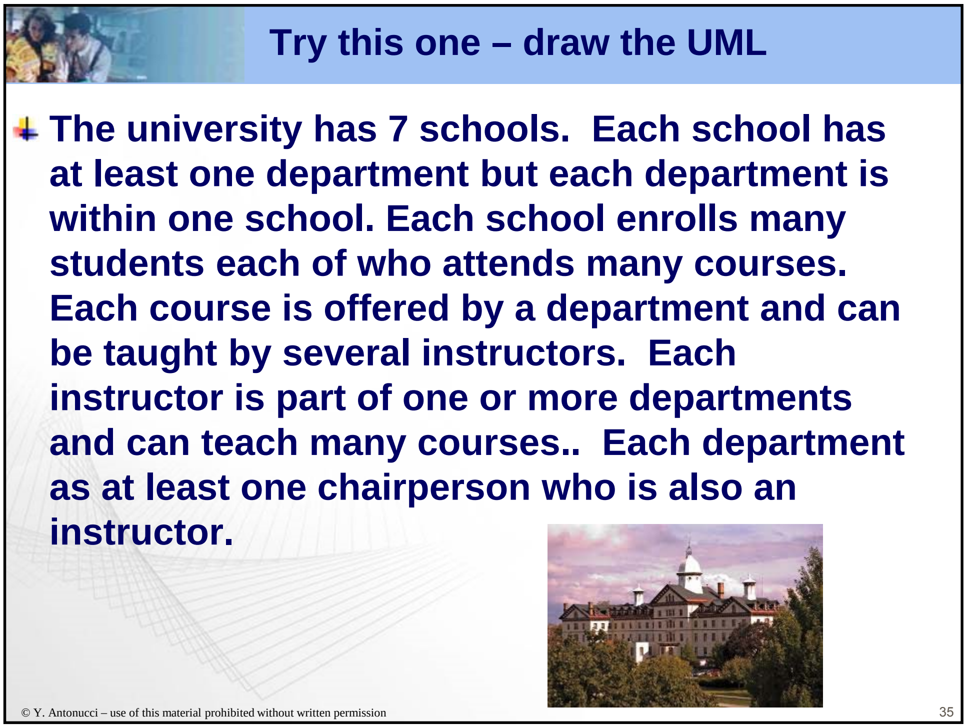
Employee

1

◂ works for

1..*

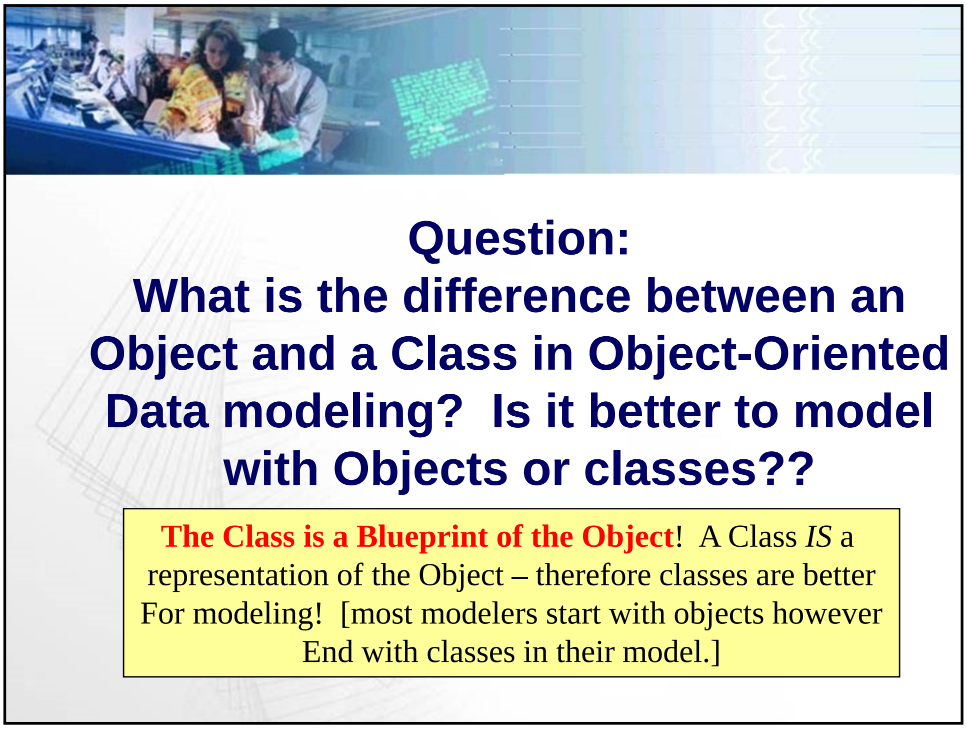
34



## Try this one – draw the UML

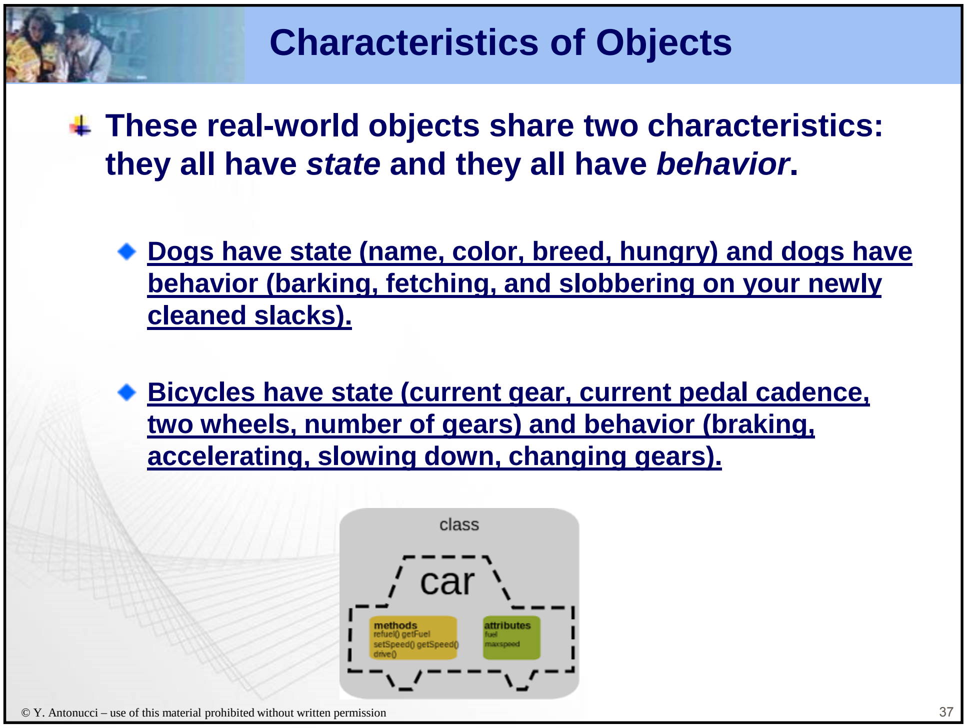
- The university has 7 schools. Each school has at least one department but each department is within one school. Each school enrolls many students each of who attends many courses. Each course is offered by a department and can be taught by several instructors. Each instructor is part of one or more departments and can teach many courses.. Each department as at least one chairperson who is also an instructor.

35



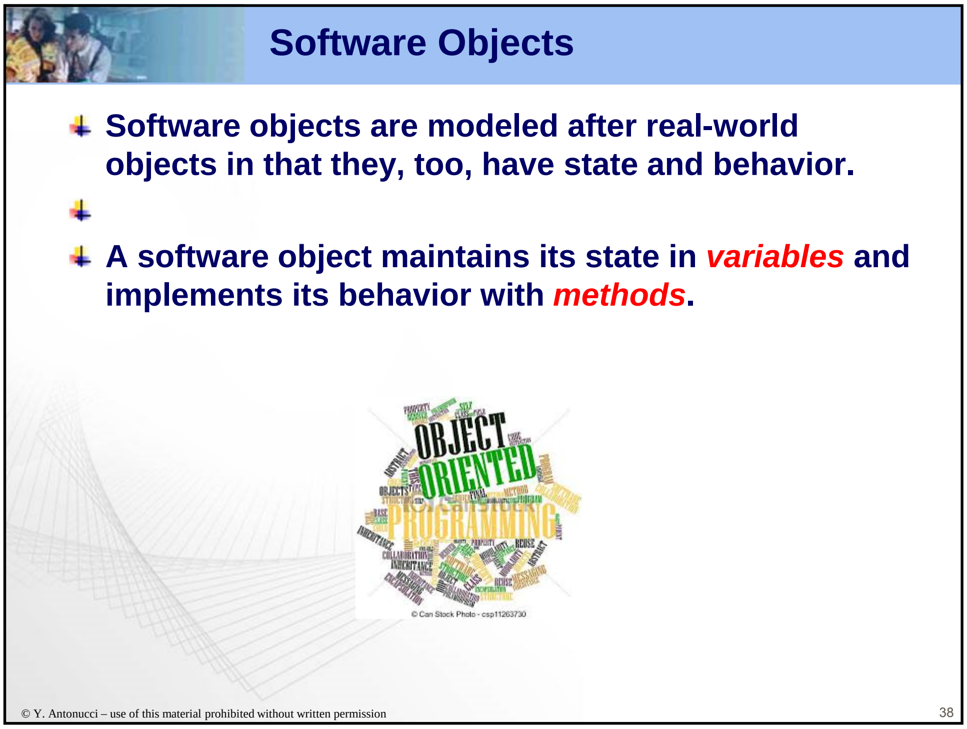
## Question:
## What is the difference between an Object and a Class in Object-Oriented Data modeling? Is it better to model with Objects or classes??

**The Class is a Blueprint of the Object**! A Class *IS* a representation of the Object – therefore classes are better For modeling! [most modelers start with objects however End with classes in their model.]



## Characteristics of Objects

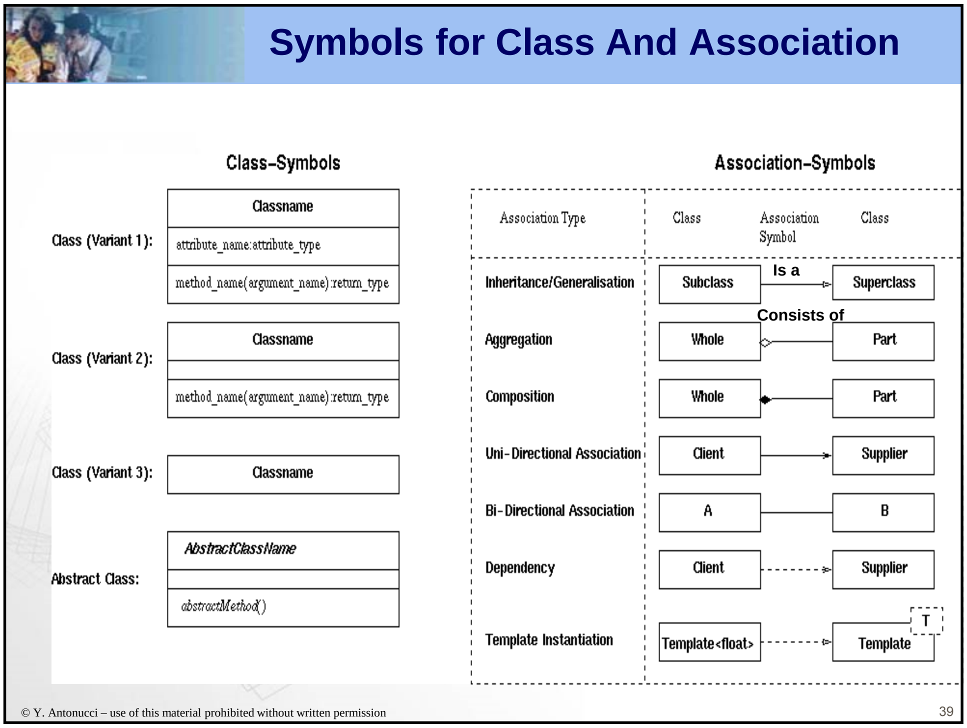
- **These real-world objects share two characteristics: they all have *state* and they all have *behavior*.**
  - **Dogs have state (name, color, breed, hungry) and dogs have behavior (barking, fetching, and slobbering on your newly cleaned slacks).**
  - **Bicycles have state (current gear, current pedal cadence, two wheels, number of gears) and behavior (braking, accelerating, slowing down, changing gears).**



## Software Objects

- Software objects are modeled after real-world objects in that they, too, have state and behavior.
-
- A software object maintains its state in ***variables*** and implements its behavior with ***methods***.

OBJECT
ORIENTED
PROGRAMMING
© Can Stock Photo - csp11263730



## Symbols for Class And Association

**Class-Symbols**

Class (Variant 1):

| Classname |
|---|
| attribute_name:attribute_type |
| method_name(argument_name):return_type |

Class (Variant 2):

| Classname |
|---|
| |
| method_name(argument_name):return_type |

Class (Variant 3):

| Classname |
|---|

Abstract Class:

| *AbstractClassName* |
|---|
| |
| *abstractMethod()* |

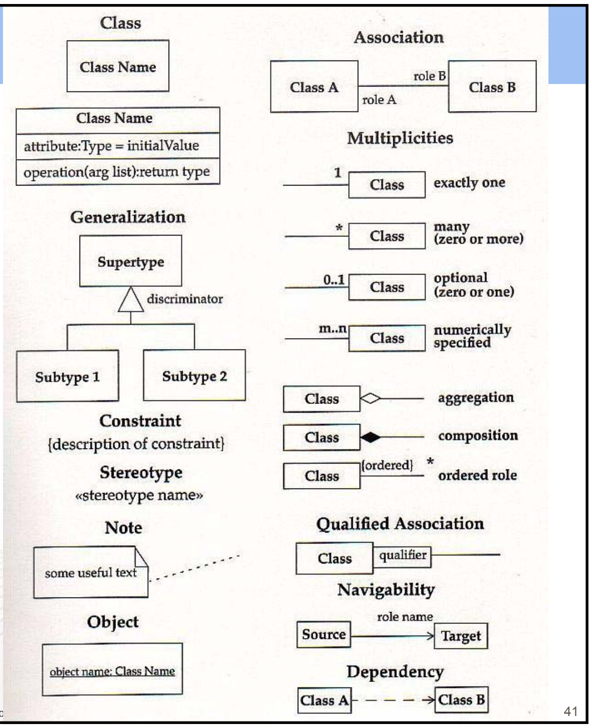
**Association-Symbols**

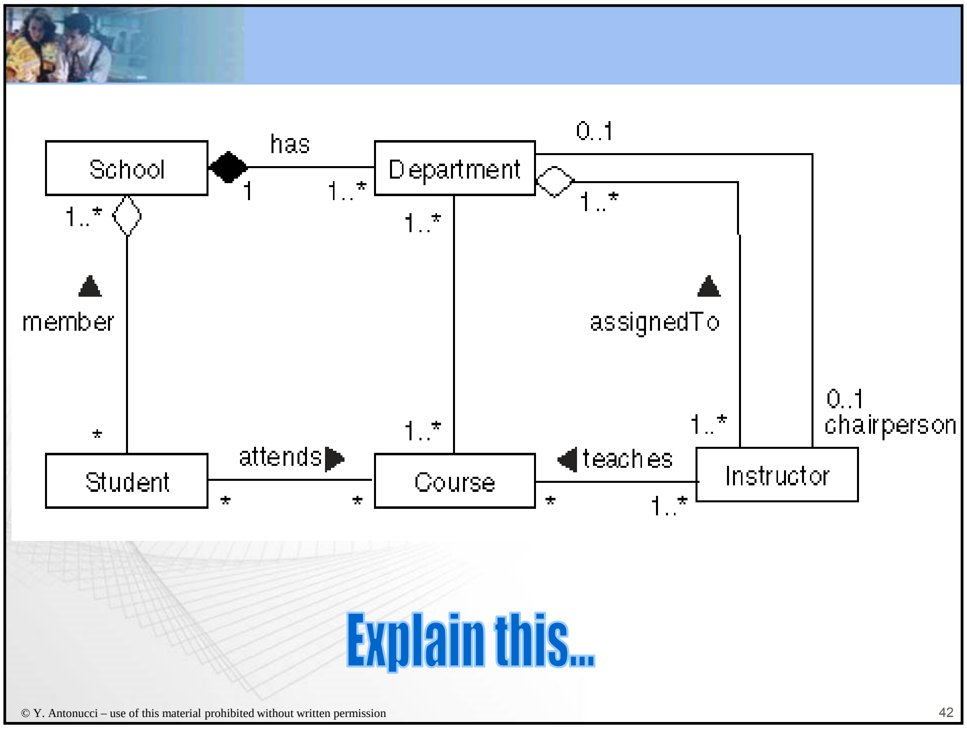
| Association Type | Class | Association Symbol | Class |
|---|---|---|---|
| Inheritance/Generalisation | Subclass | Is a | Superclass |
| Aggregation | Whole | Consists of | Part |
| Composition | Whole | | Part |
| Uni-Directional Association | Client | | Supplier |
| Bi-Directional Association | A | | B |
| Dependency | Client | | Supplier |
| Template Instantiation | Template<float> | | Template<br>T |

## Exercise

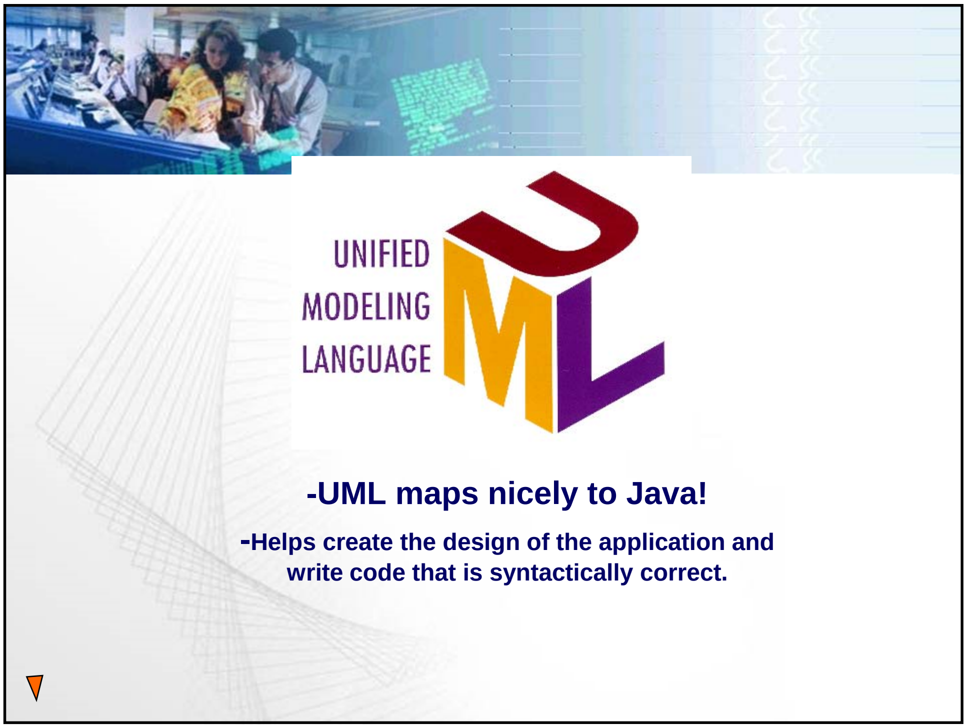
- Indicate on the list below which are associations and which are aggregations
  - Car ...... Road
  - Head ...... Mouth
  - Book ...... Chapter
  - Computer ...... Printer

## Class and Association's Notation



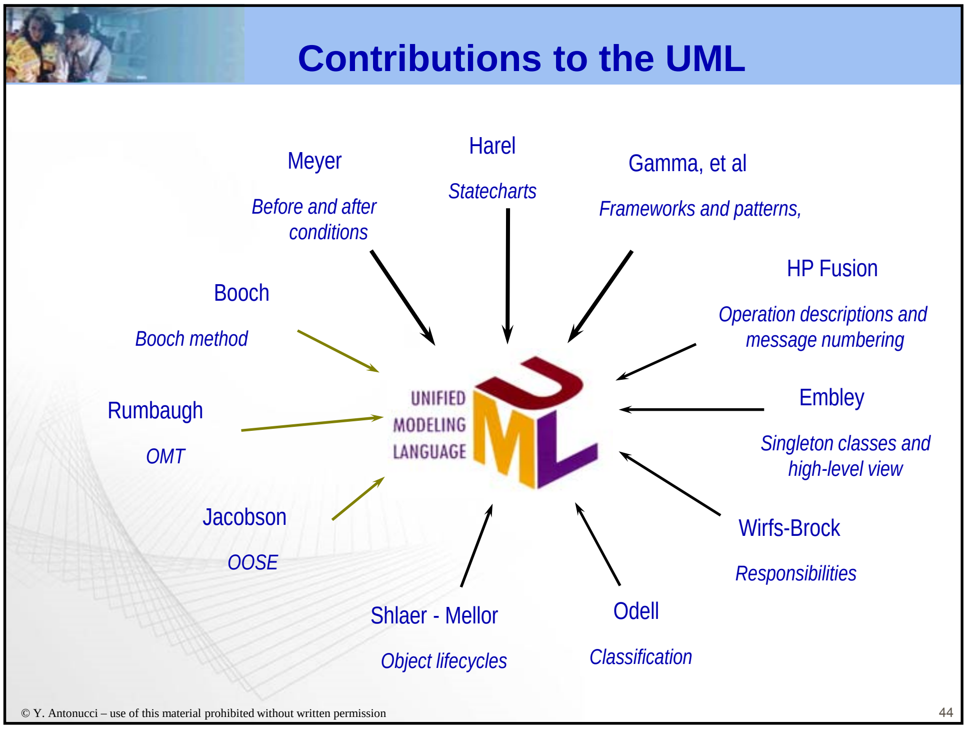


Explain this...



-UML maps nicely to Java!

-Helps create the design of the application and write code that is syntactically correct.



## Contributions to the UML

Meyer

*Before and after conditions*

Harel

*Statecharts*

Gamma, et al

*Frameworks and patterns,*

Booch

*Booch method*

HP Fusion

*Operation descriptions and message numbering*

Rumbaugh

*OMT*

Embley

*Singleton classes and high-level view*

Jacobson

*OOSE*

Wirfs-Brock

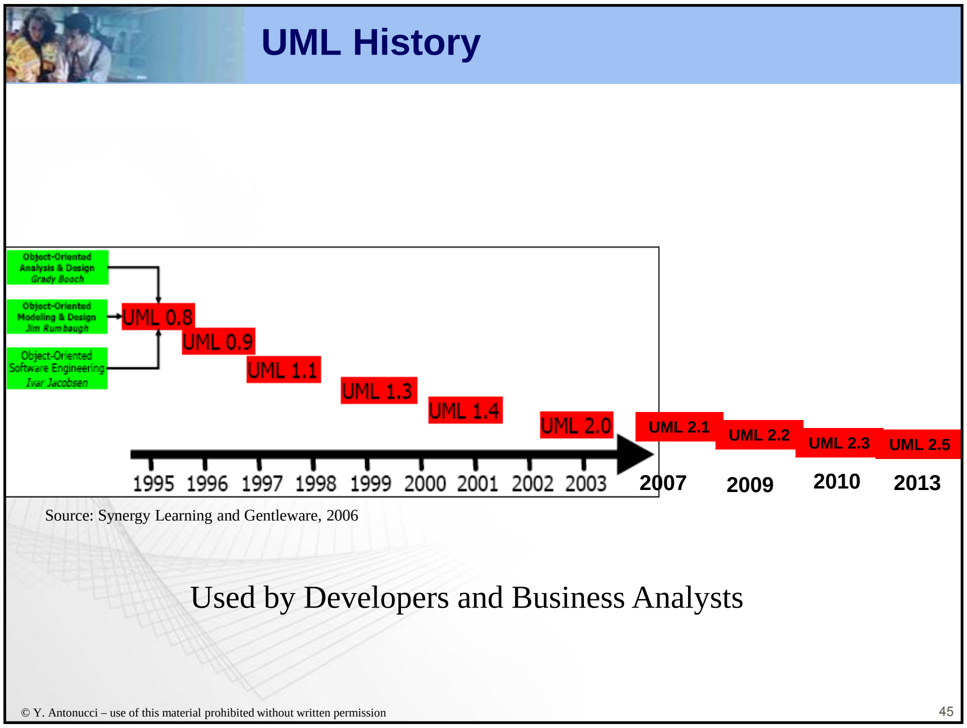
*Responsibilities*

Shlaer - Mellor

*Object lifecycles*

Odell

*Classification*



## UML History

Object-Oriented Analysis & Design *Grady Booch*

Object-Oriented Modeling & Design *Jim Rumbaugh*

Object-Oriented Software Engineering *Ivar Jacobsen*

UML 0.8

UML 0.9

UML 1.1

UML 1.3

UML 1.4

UML 2.0

UML 2.1

UML 2.2

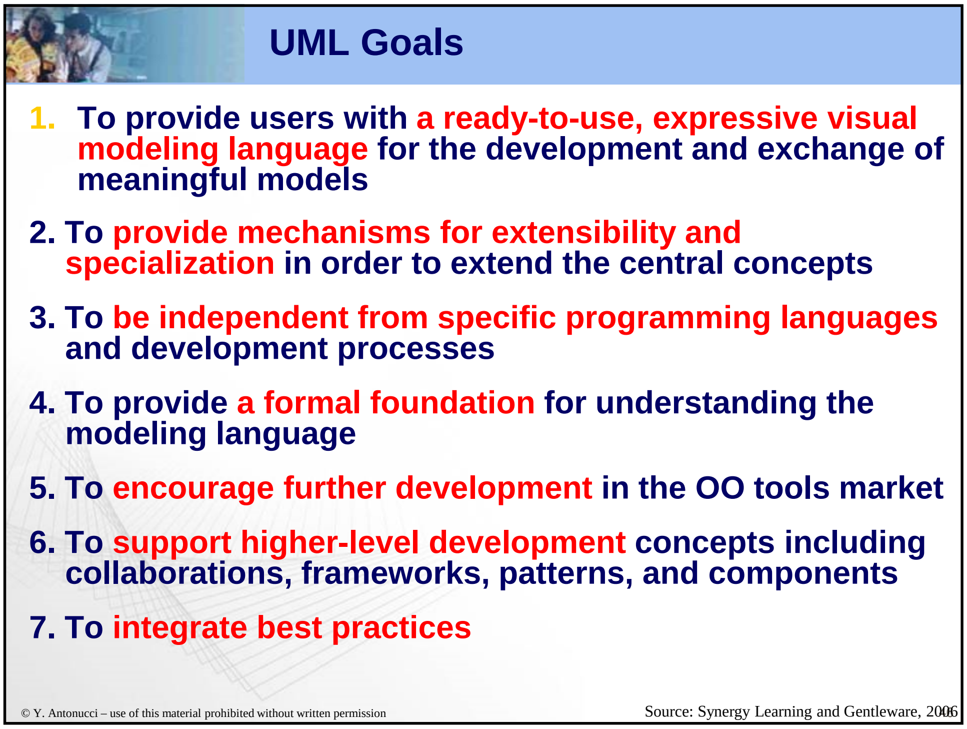
UML 2.3

UML 2.5

1995 1996 1997 1998 1999 2000 2001 2002 2003 2007 2009 2010 2013

Source: Synergy Learning and Gentleware, 2006

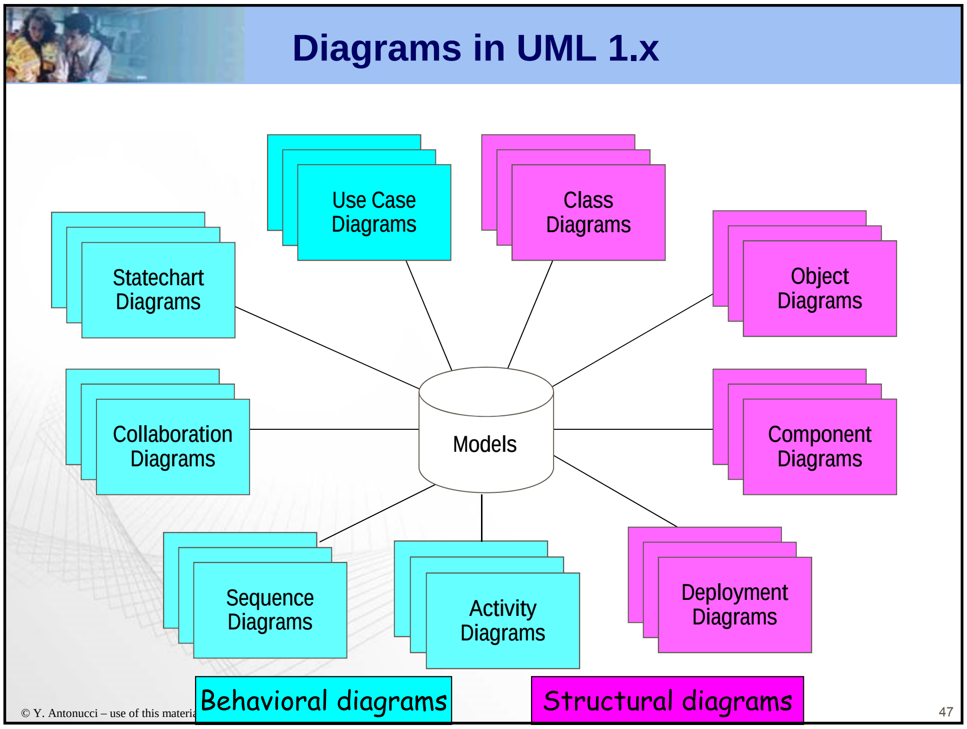
Used by Developers and Business Analysts



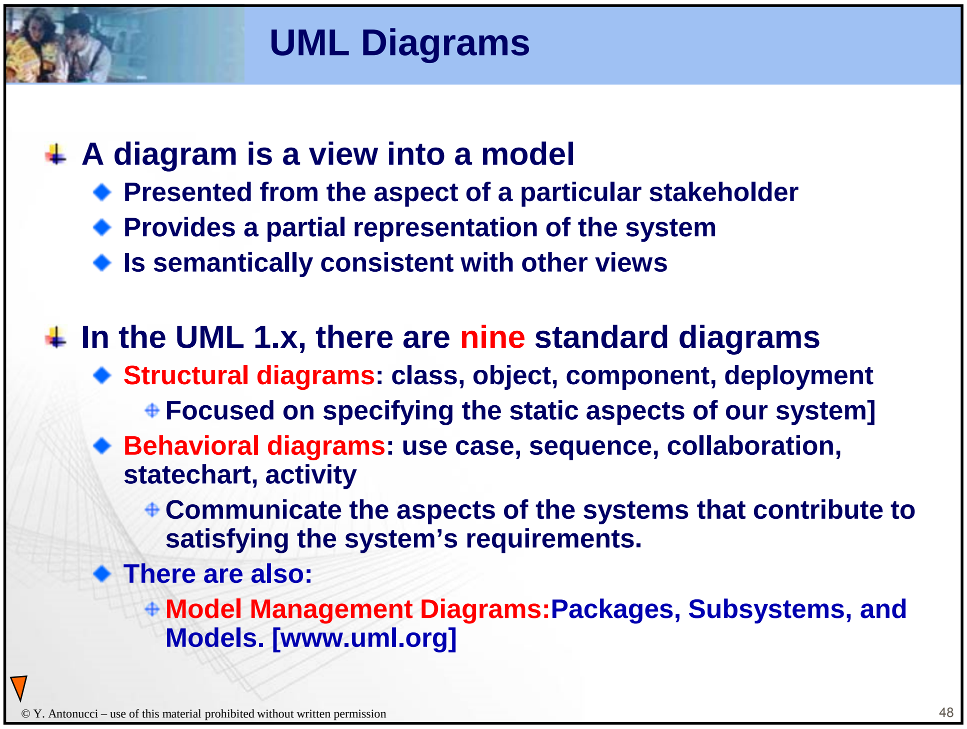
## UML Goals

1. To provide users with a ready-to-use, expressive visual modeling language for the development and exchange of meaningful models
2. To provide mechanisms for extensibility and specialization in order to extend the central concepts
3. To be independent from specific programming languages and development processes
4. To provide a formal foundation for understanding the modeling language
5. To encourage further development in the OO tools market
6. To support higher-level development concepts including collaborations, frameworks, patterns, and components
7. To integrate best practices

Source: Synergy Learning and Gentleware, 2006

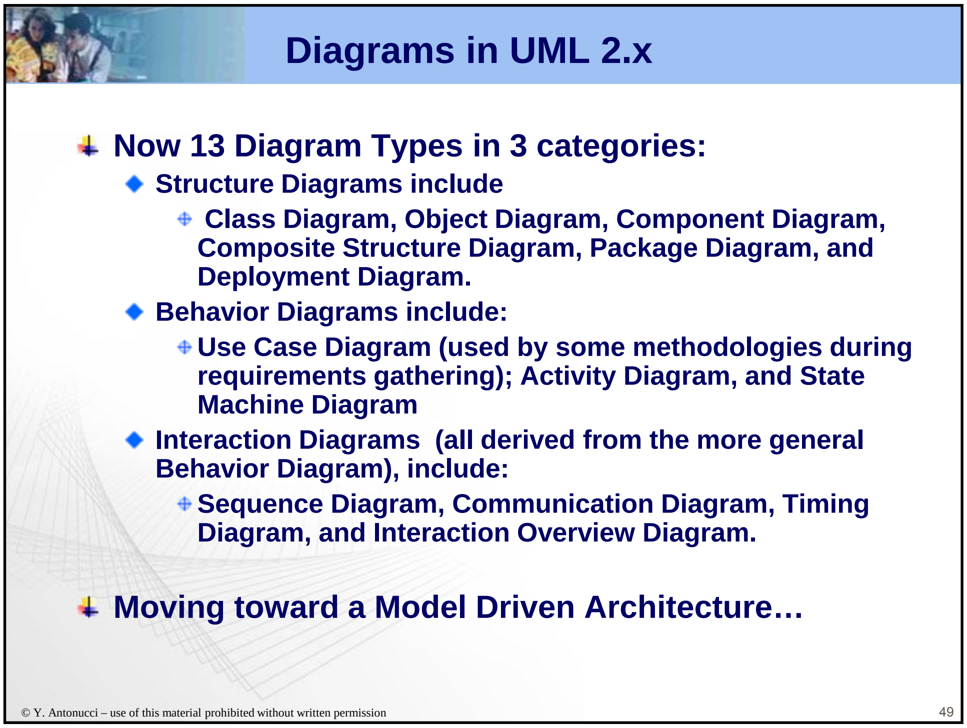


## Diagrams in UML 1.x



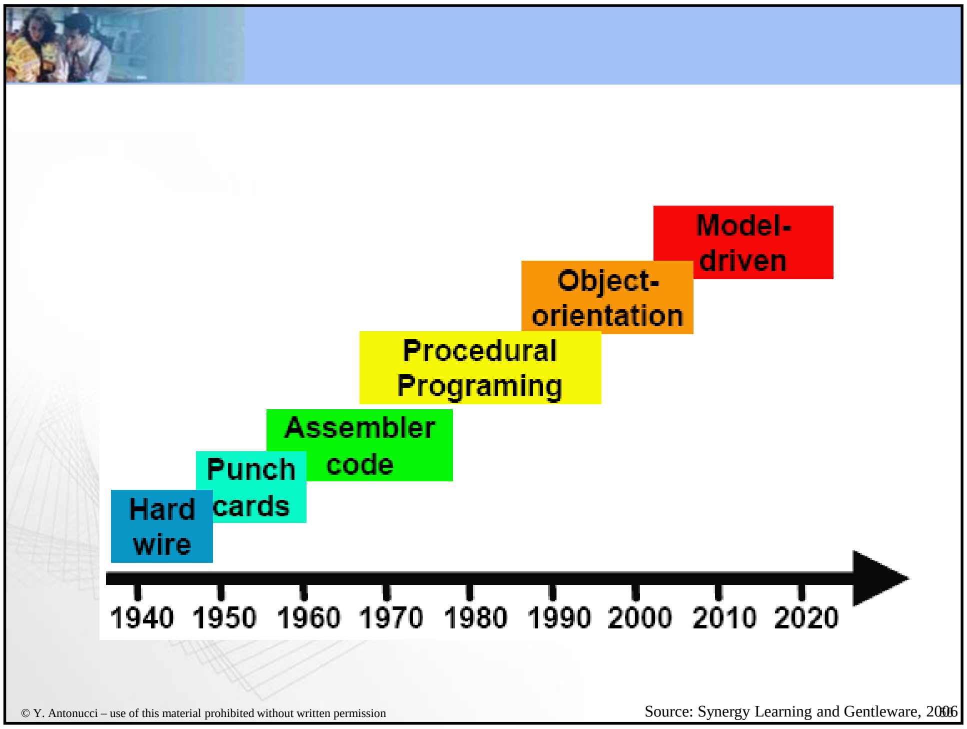
## UML Diagrams

- A diagram is a view into a model
  - Presented from the aspect of a particular stakeholder
  - Provides a partial representation of the system
  - Is semantically consistent with other views
- In the UML 1.x, there are nine standard diagrams
  - Structural diagrams: class, object, component, deployment
    - Focused on specifying the static aspects of our system]
  - Behavioral diagrams: use case, sequence, collaboration, statechart, activity
    - Communicate the aspects of the systems that contribute to satisfying the system's requirements.
  - There are also:
    - Model Management Diagrams:Packages, Subsystems, and Models. [www.uml.org]

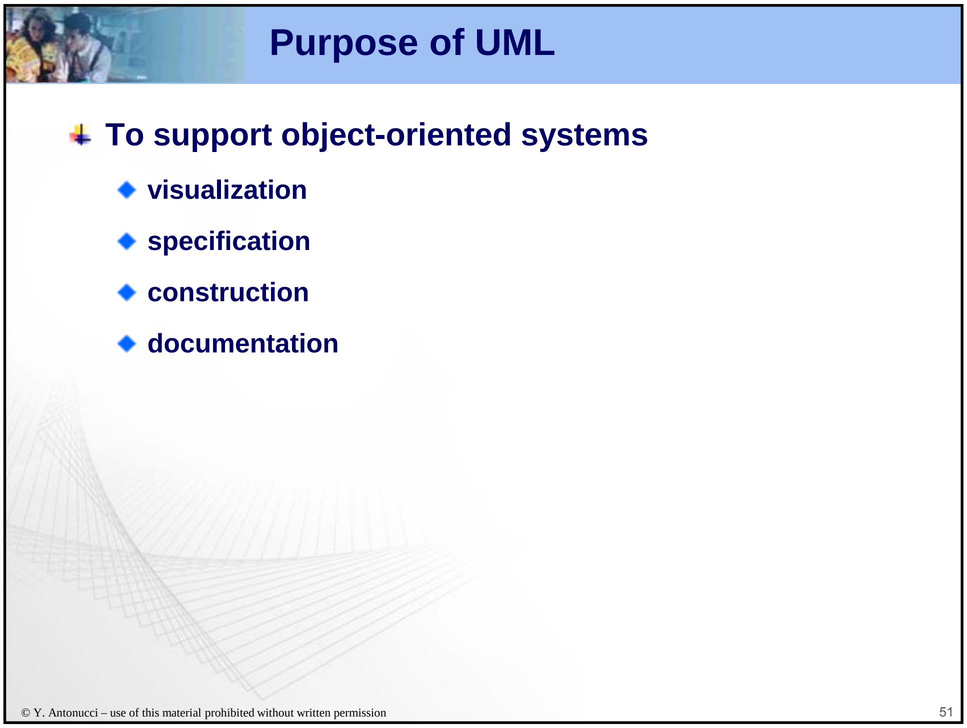


## Diagrams in UML 2.x

- Now 13 Diagram Types in 3 categories:
  - Structure Diagrams include
    - Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, and Deployment Diagram.
  - Behavior Diagrams include:
    - Use Case Diagram (used by some methodologies during requirements gathering); Activity Diagram, and State Machine Diagram
  - Interaction Diagrams (all derived from the more general Behavior Diagram), include:
    - Sequence Diagram, Communication Diagram, Timing Diagram, and Interaction Overview Diagram.
- Moving toward a Model Driven Architecture…



Source: Synergy Learning and Gentleware, 2006



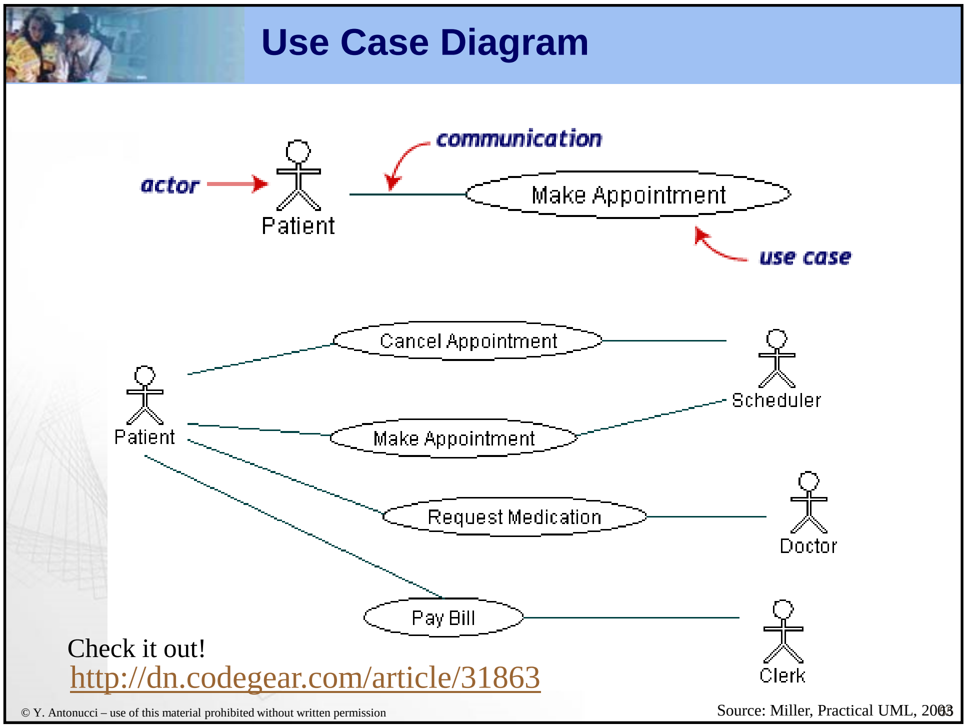
## Purpose of UML

- To support object-oriented systems
  - visualization
  - specification
  - construction
  - documentation

## Building Blocks of the UML

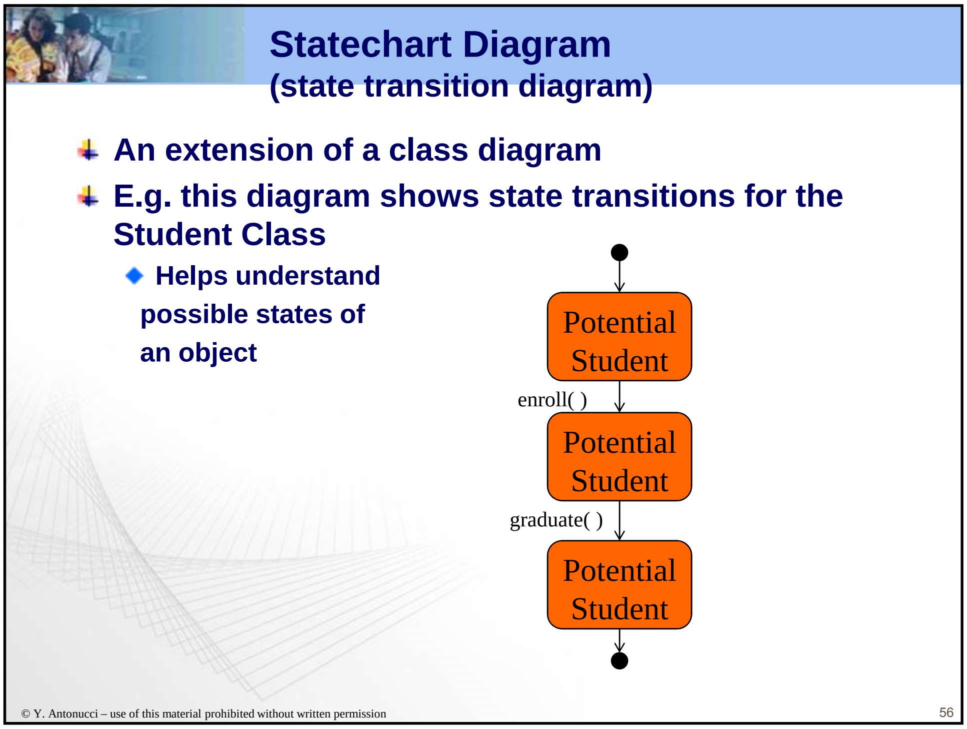
- Things
- Relationships
- Diagrams

## Use Case Diagram



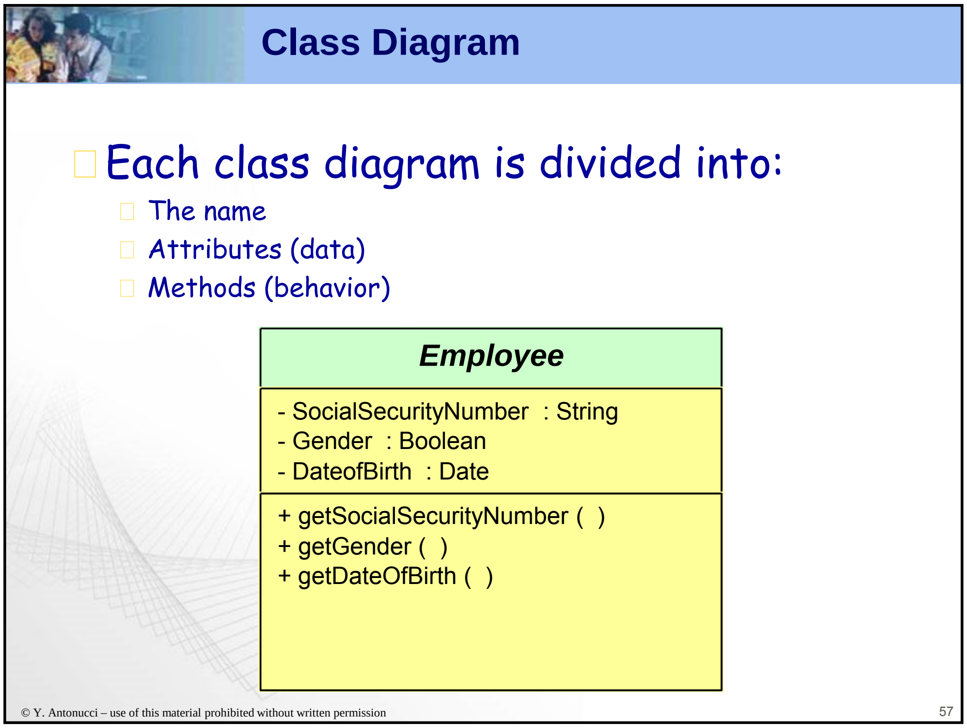
Check it out!
http://dn.codegear.com/article/31863

Source: Miller, Practical UML, 2003



## Statechart Diagram
## (state transition diagram)

- An extension of a class diagram
- E.g. this diagram shows state transitions for the Student Class
  - Helps understand possible states of an object



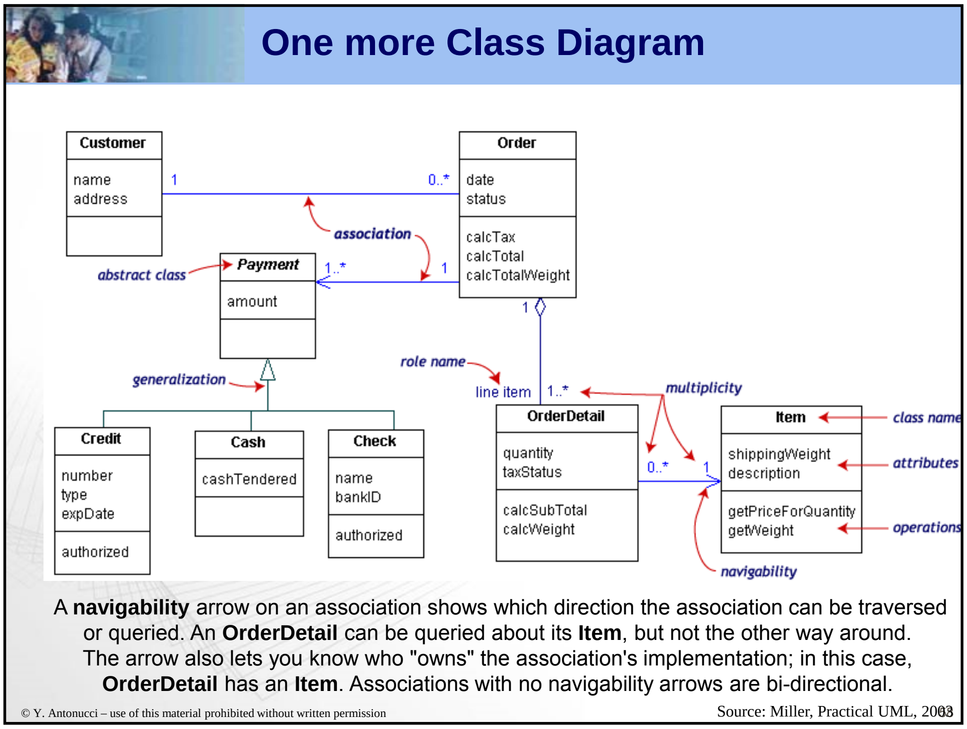
## Class Diagram

- Each class diagram is divided into:
  - The name
  - Attributes (data)
  - Methods (behavior)

***Employee***

- SocialSecurityNumber : String
- Gender : Boolean
- DateofBirth : Date

+ getSocialSecurityNumber ( )
+ getGender ( )
+ getDateOfBirth ( )

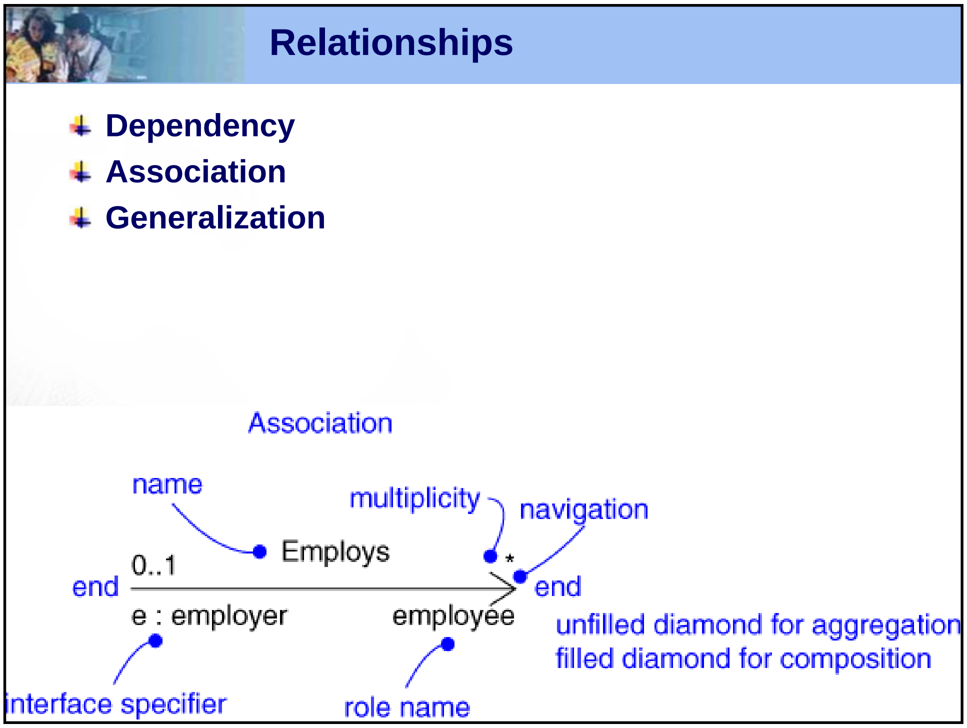


## One more Class Diagram

A **navigability** arrow on an association shows which direction the association can be traversed or queried. An **OrderDetail** can be queried about its **Item**, but not the other way around. The arrow also lets you know who "owns" the association's implementation; in this case, **OrderDetail** has an **Item**. Associations with no navigability arrows are bi-directional.

Source: Miller, Practical UML, 2003

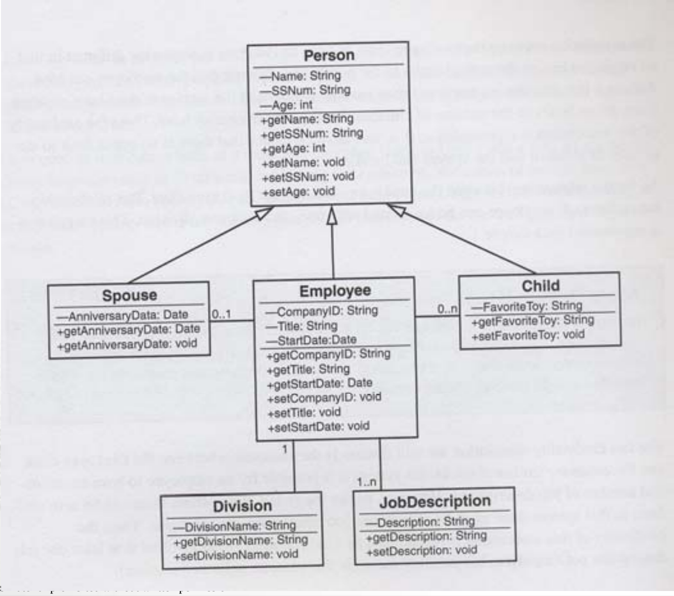
63



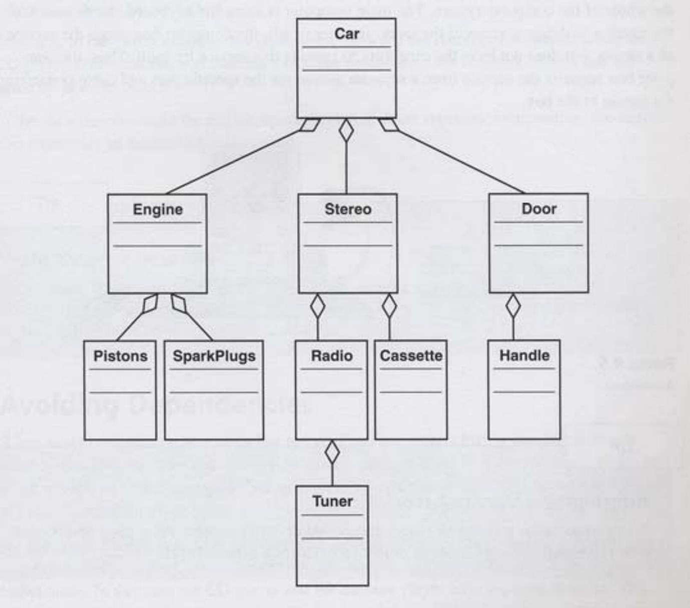
## Relationships

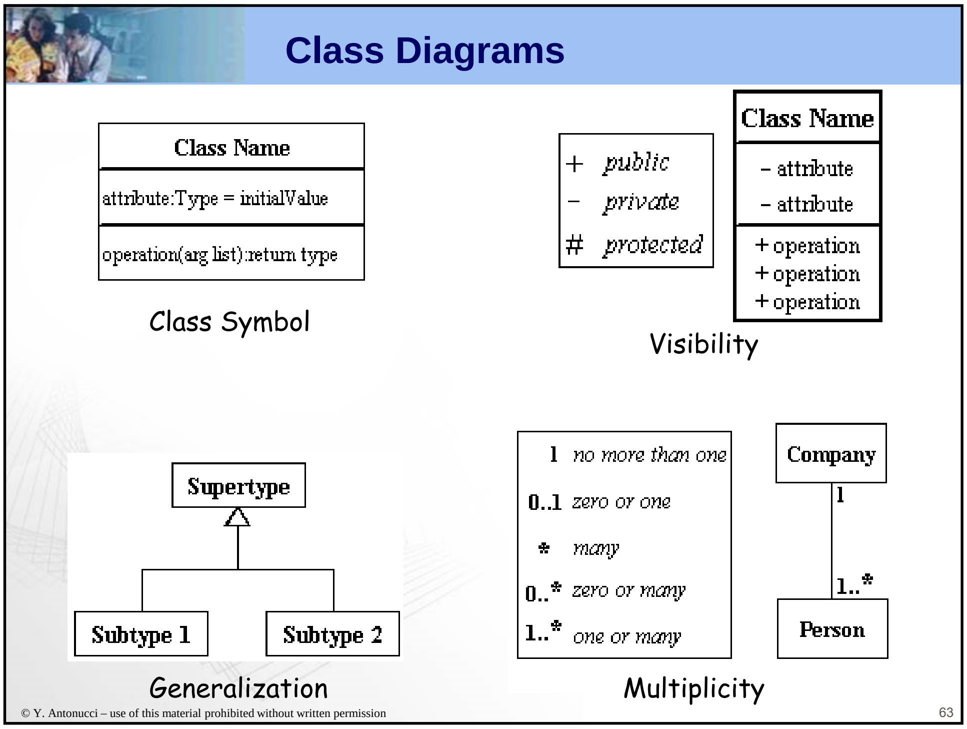
- Dependency
- Association
- Generalization

## UML – showing Multiplicity



## UML – showing Composition





## Class Diagrams

Class Name
attribute:Type = initialValue
operation(arg list):return type

Class Symbol

+ *public*
− *private*
# *protected*

Class Name
- attribute
- attribute
+ operation
+ operation
+ operation

Visibility

Supertype
Subtype 1
Subtype 2

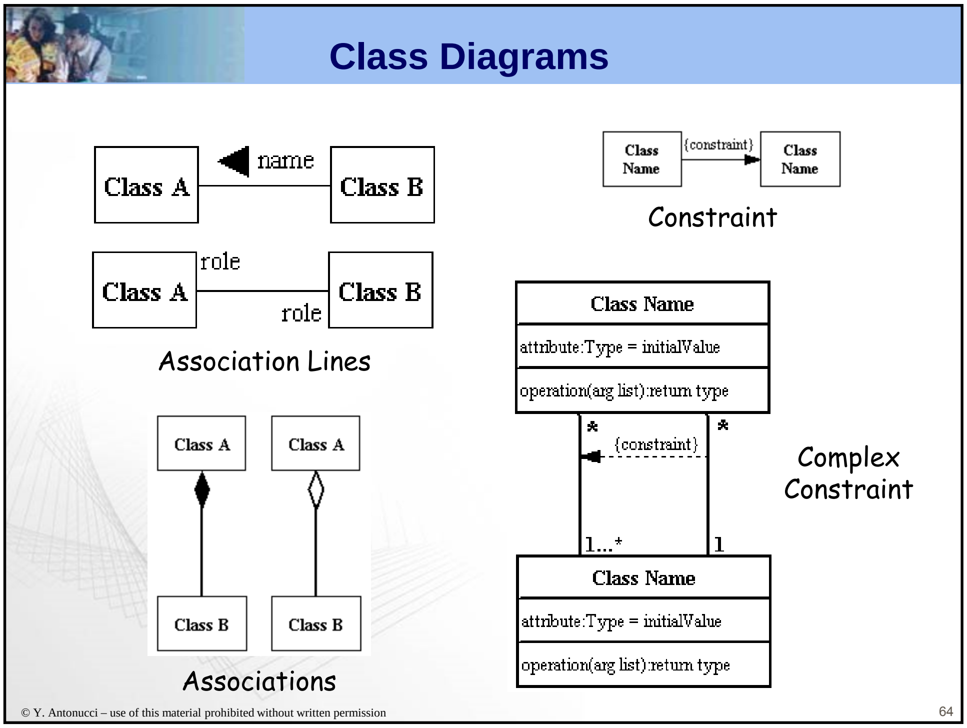
Generalization

1 *no more than one*
0..1 *zero or one*
* *many*
0..* *zero or many*
1..* *one or many*

Company
1
1..*
Person

Multiplicity

63



## Class Diagrams

Class A ◀ name Class B

Class A role role Class B

Association Lines

Class A
Class B
Class A
Class B

Associations

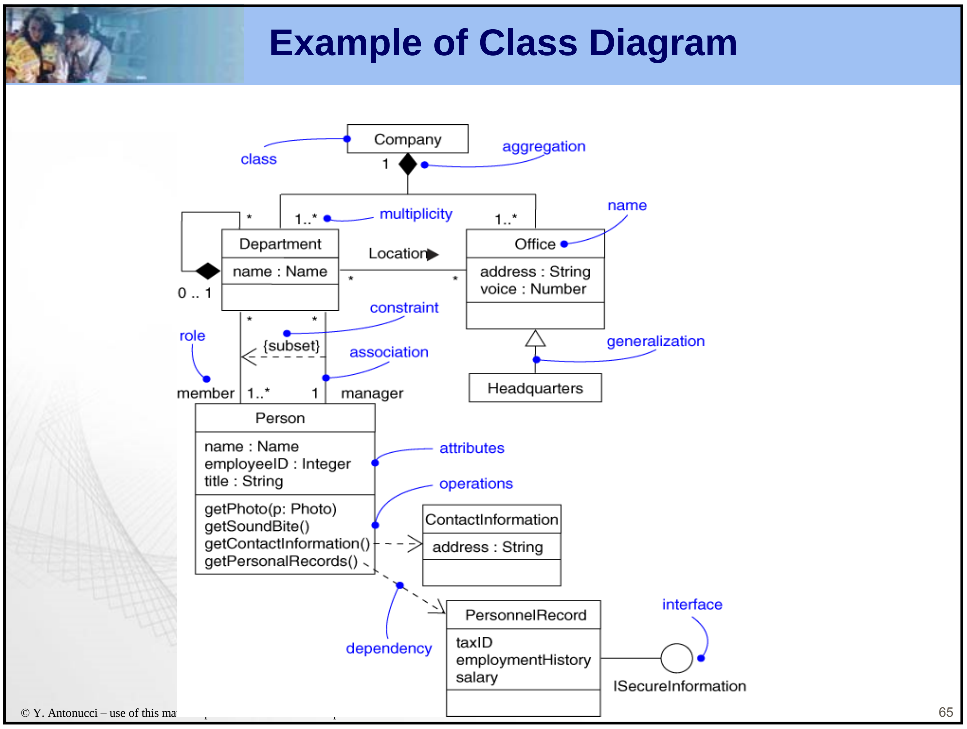
Class Name {constraint} Class Name

Constraint

Class Name
attribute:Type = initialValue
operation(arg list):return type
*
{constraint}
*
1...*
1
Class Name
attribute:Type = initialValue
operation(arg list):return type

Complex Constraint

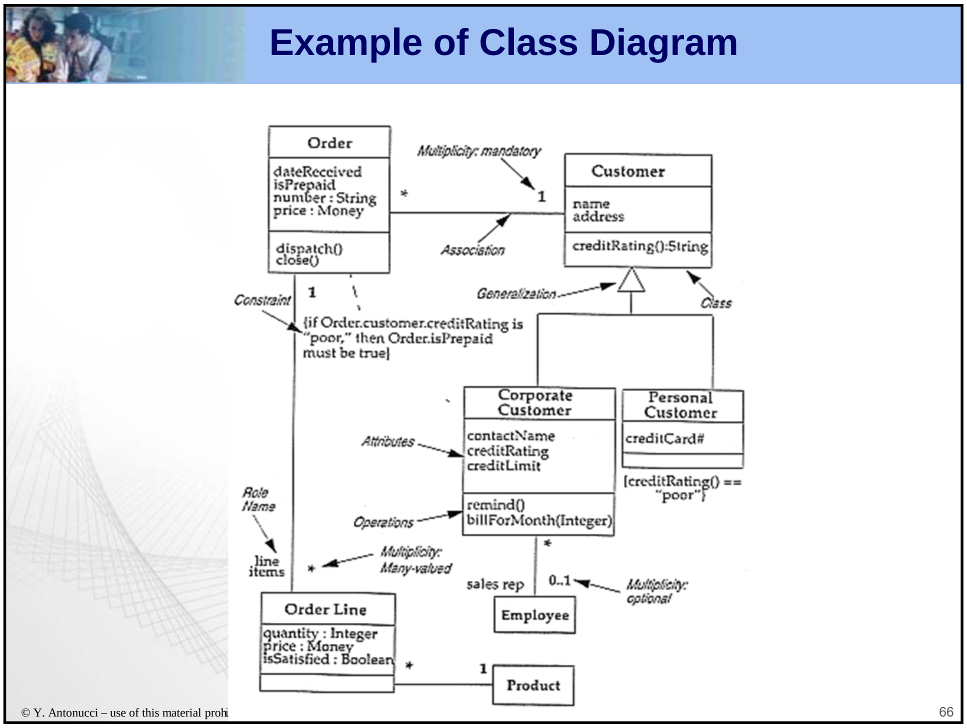
64



## Example of Class Diagram

© Y. Antonucci – use of this ma

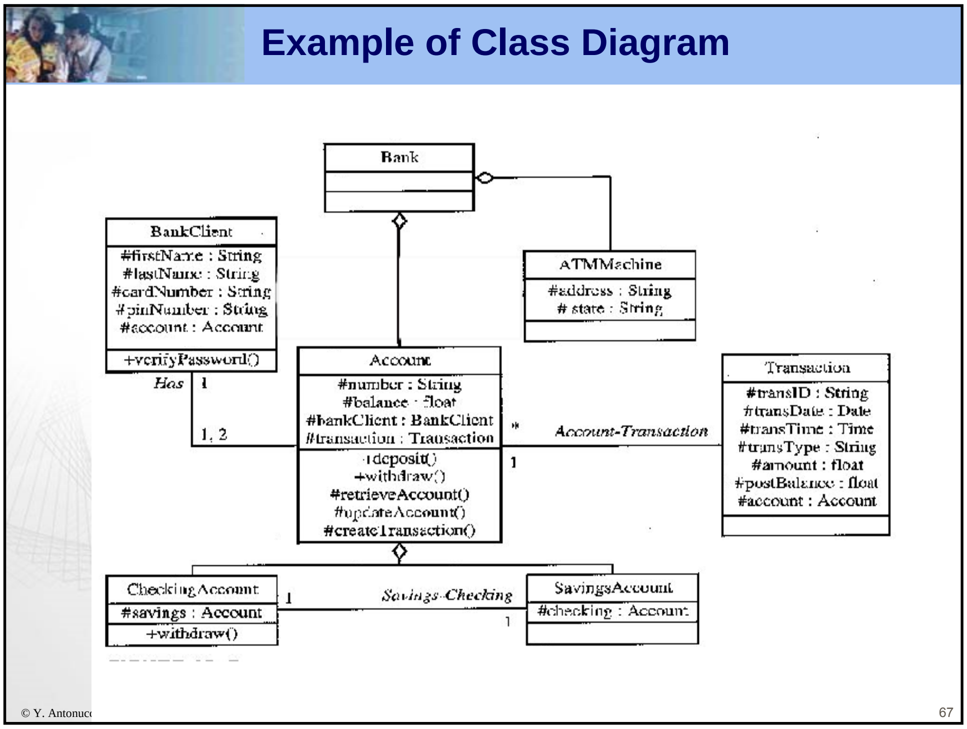
65



## Example of Class Diagram

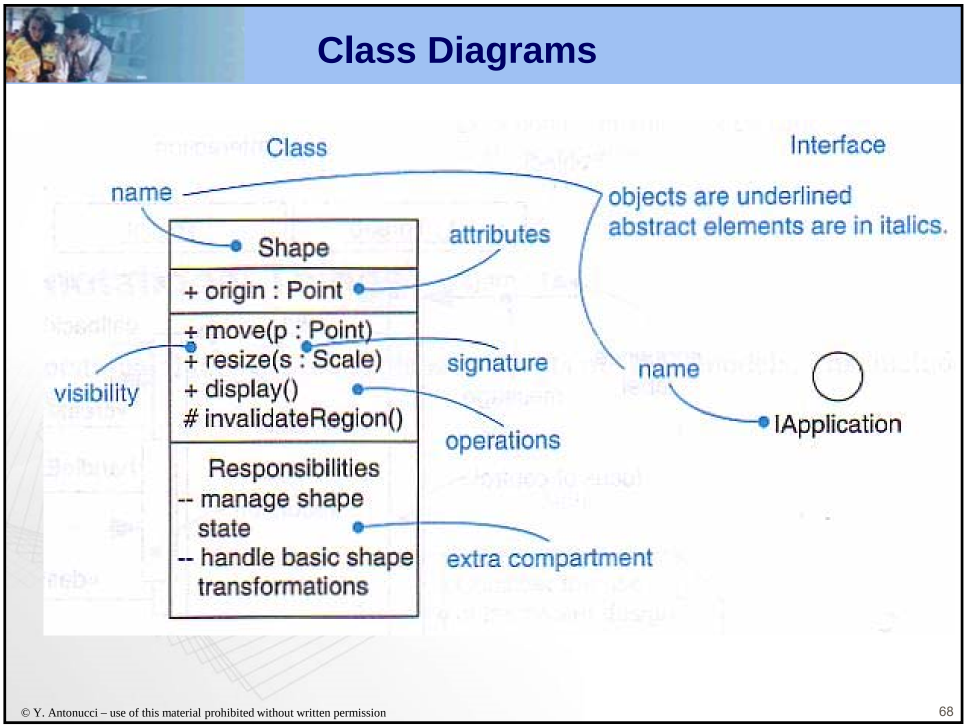
© Y. Antonucci – use of this material prohi

66



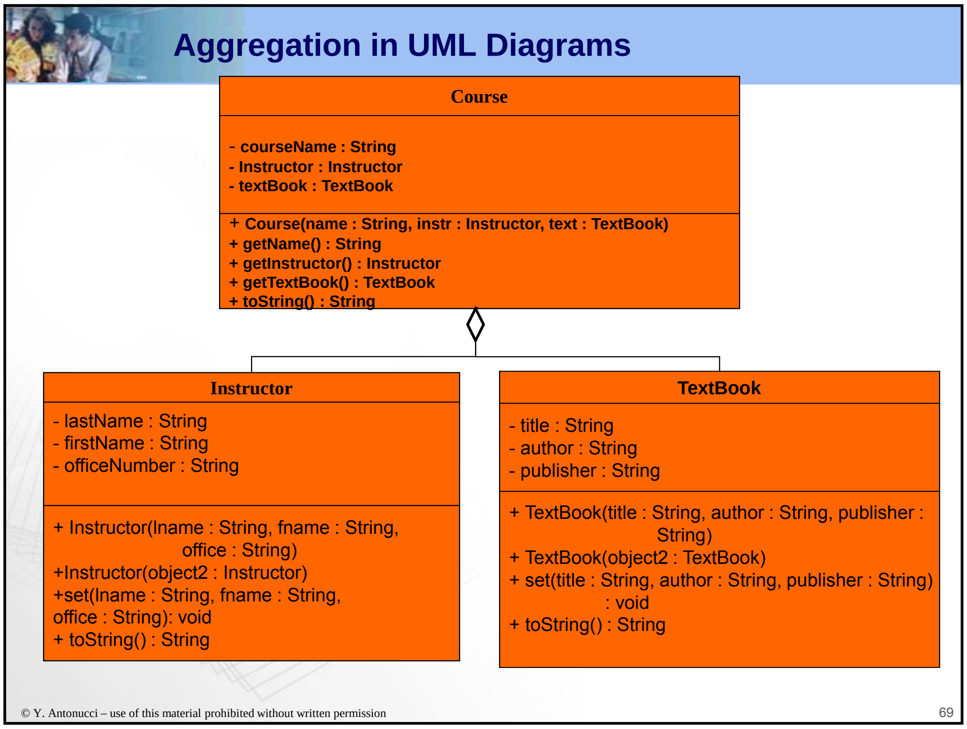
## Example of Class Diagram

© Y. Antonucc



## Class Diagrams

© Y. Antonucci – use of this material prohibited without written permission



## Aggregation in UML Diagrams

**Course**

- courseName : String
- Instructor : Instructor
- textBook : TextBook

+ Course(name : String, instr : Instructor, text : TextBook)
+ getName() : String
+ getInstructor() : Instructor
+ getTextBook() : TextBook
+ toString() : String

**Instructor**

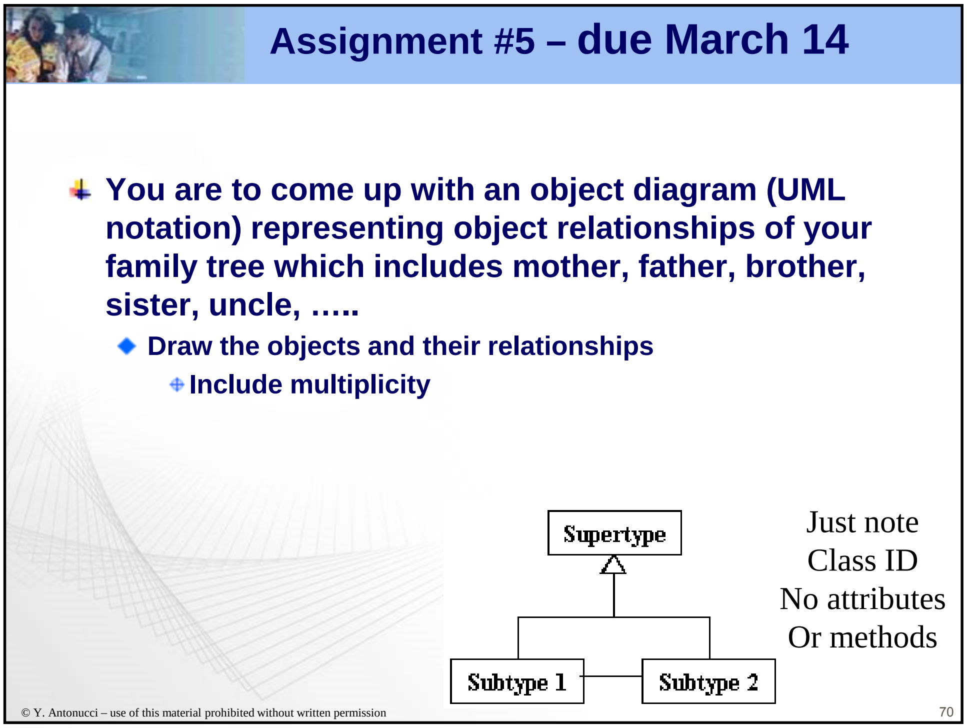
- lastName : String
- firstName : String
- officeNumber : String

+ Instructor(lname : String, fname : String, office : String)
+Instructor(object2 : Instructor)
+set(lname : String, fname : String, office : String): void
+ toString() : String

**TextBook**

- title : String
- author : String
- publisher : String

+ TextBook(title : String, author : String, publisher : String)
+ TextBook(object2 : TextBook)
+ set(title : String, author : String, publisher : String) : void
+ toString() : String



## Assignment #5 – due March 14

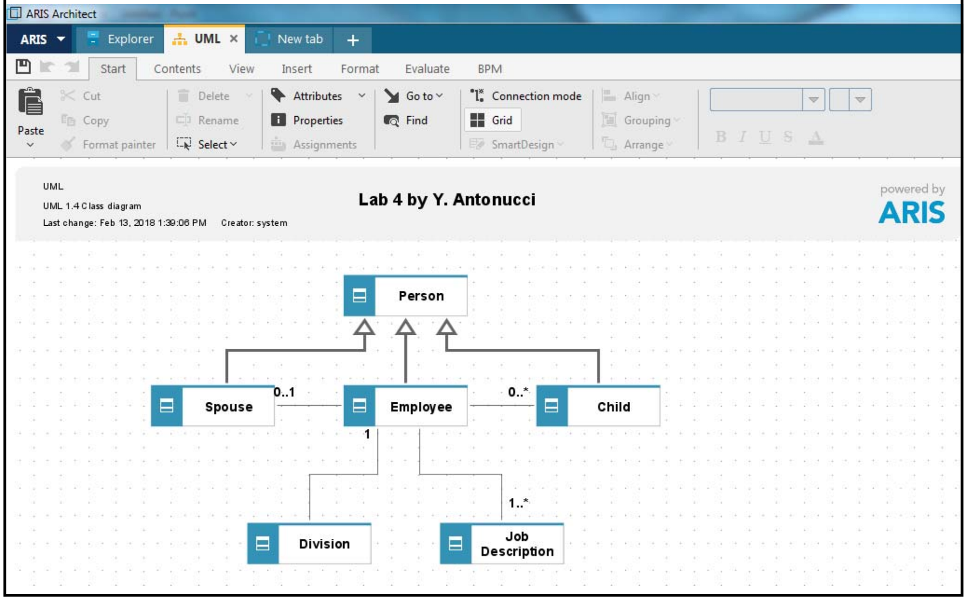
- You are to come up with an object diagram (UML notation) representing object relationships of your family tree which includes mother, father, brother, sister, uncle, …..
  - Draw the objects and their relationships
    - Include multiplicity

Supertype
Subtype 1
Subtype 2

Just note
Class ID
No attributes
Or methods

# Intro to UML in ARIS- LAB #4

## class diagram in ARIS



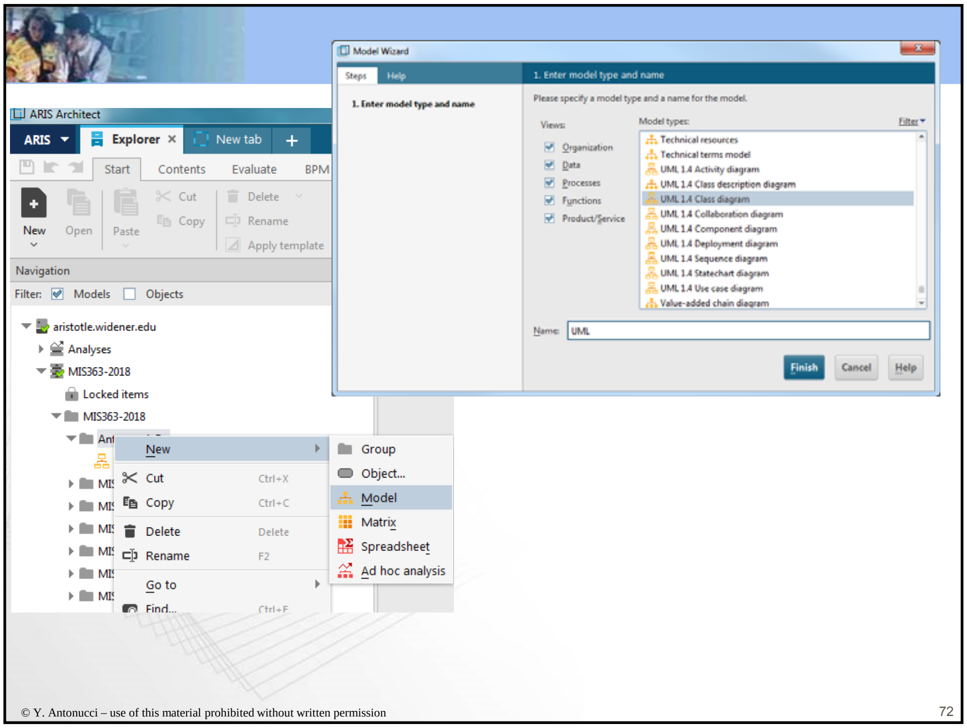
UML

UML 1.4 Class diagram

Last change: Feb 13, 2018 1:39:06 PM Creator: system

Lab 4 by Y. Antonucci

powered by ARIS



Model Wizard

1. Enter model type and name

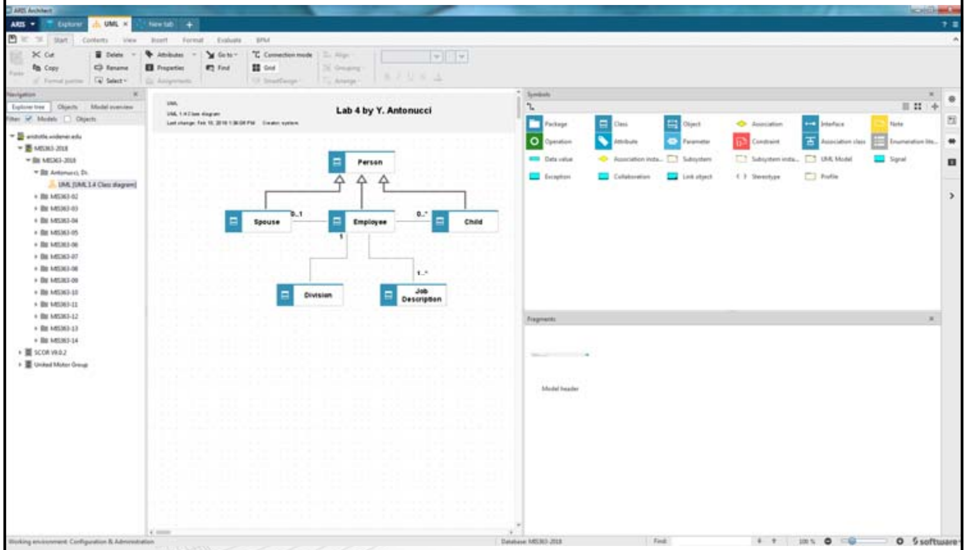
Please specify a model type and a name for the model.

Model types: UML 1.4 Class diagram

Name: UML

© Y. Antonucci – use of this material prohibited without written permission

72

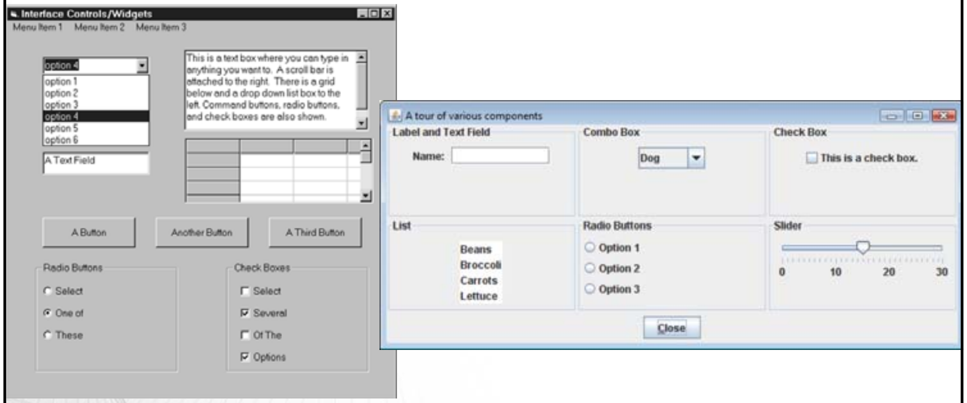


## Types of objects in Computer Systems

**Classified according to application type or system component - which is important to the developer**

1. **User Interface Objects**
   - objects that physically appear on the screen and end-users directly interact with them. --They have attributes, they exhibit behaviors, they interact with each other and most important we interact with them
   - E.g. Button, Text Box….
2. **Operating Environment Objects**
   - Exists in the network or controlled by OS (client/server)
     - A server object provides services for others and client object requests services form others.
   - directories, copy,….
3. **Task Related Objects**
   - objects are used to actually complete work. These things that a computer application deals with or creates.
   - E.g. Document objects, Multimedia objects, problem domain objects such as customer, product…

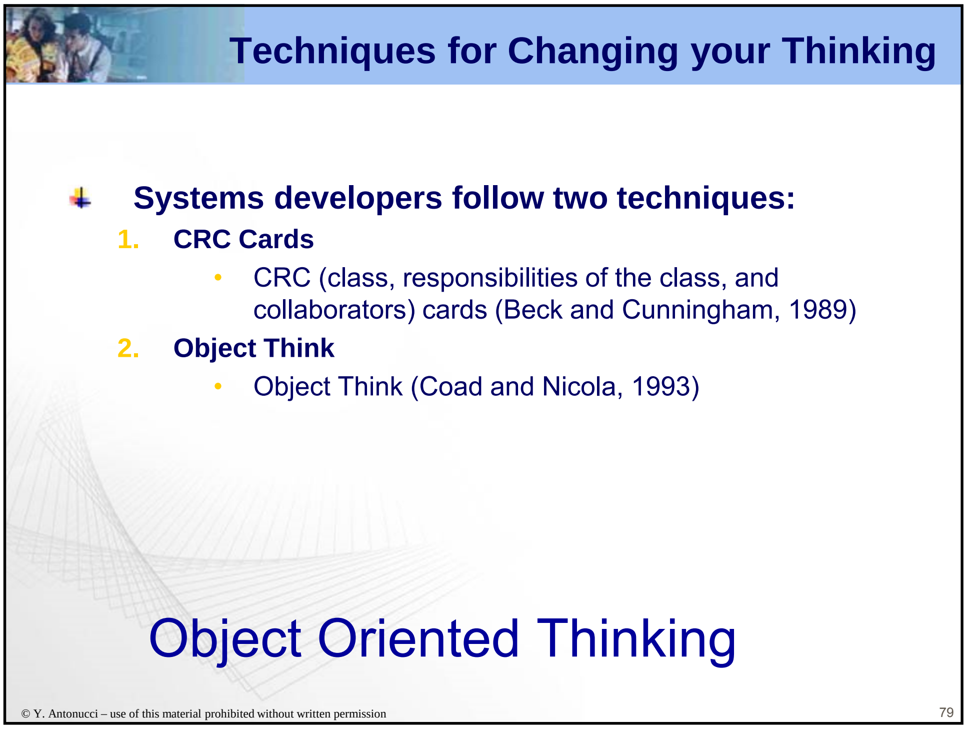
## Common User Interface Objects



This is a text box where you can type in anything you want to. A scroll bar is attached to the right. There is a grid below and a drop down list box to the left. Command buttons, radio buttons, and check boxes are also shown.

- Some common GUI components are:
  - buttons, labels, text fields, check boxes, radio buttons, combo boxes, and sliders.

## Object Oriented Thinking

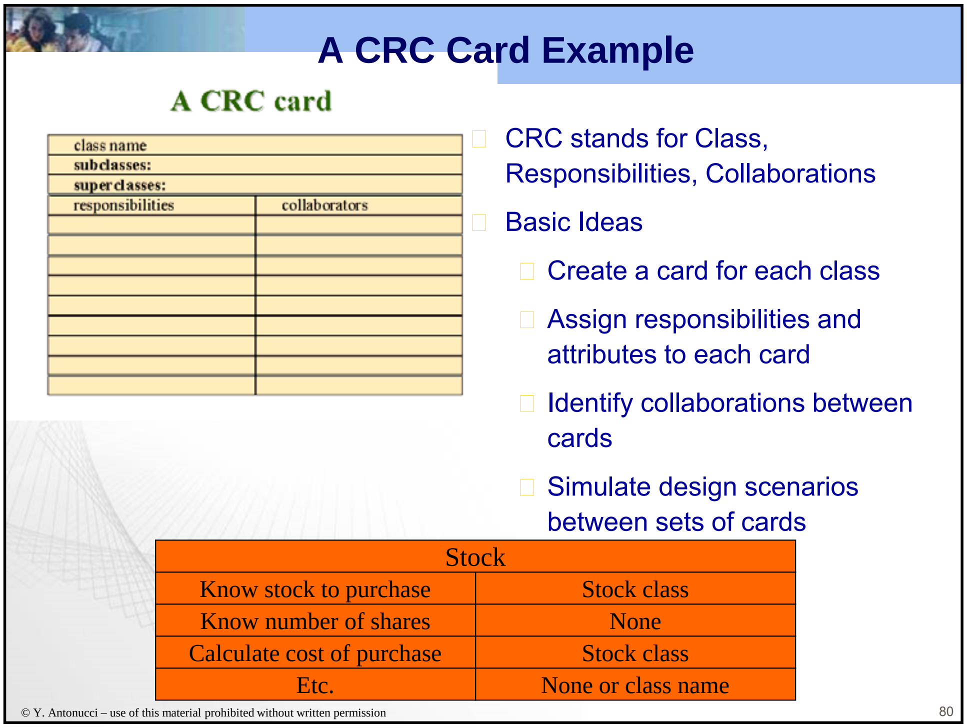


## Techniques for Changing your Thinking

- **Systems developers follow two techniques:**
  1. **CRC Cards**
     - CRC (class, responsibilities of the class, and collaborators) cards (Beck and Cunningham, 1989)
  2. **Object Think**
     - Object Think (Coad and Nicola, 1993)

Object Oriented Thinking

79



## A CRC Card Example

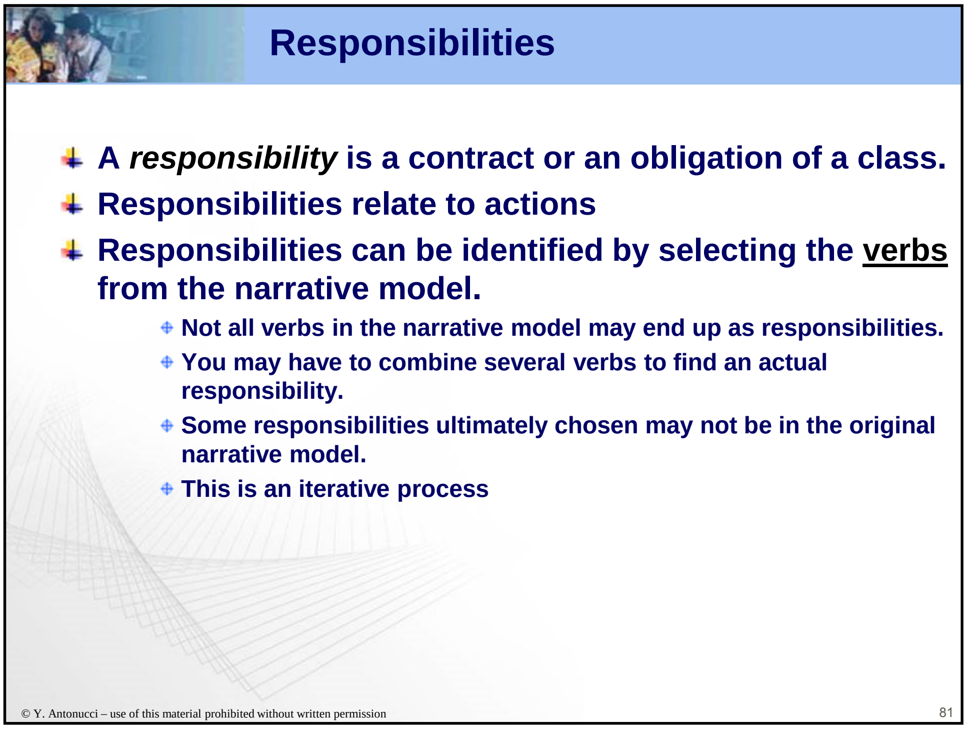
**A CRC card**

|  |  |
|---|---|
| class name |  |
| **subclasses:** |  |
| **superclasses:** |  |
| responsibilities | collaborators |

- CRC stands for Class, Responsibilities, Collaborations
- Basic Ideas
  - Create a card for each class
  - Assign responsibilities and attributes to each card
  - Identify collaborations between cards
  - Simulate design scenarios between sets of cards

| Stock |  |
|---|---|
| Know stock to purchase | Stock class |
| Know number of shares | None |
| Calculate cost of purchase | Stock class |
| Etc. | None or class name |

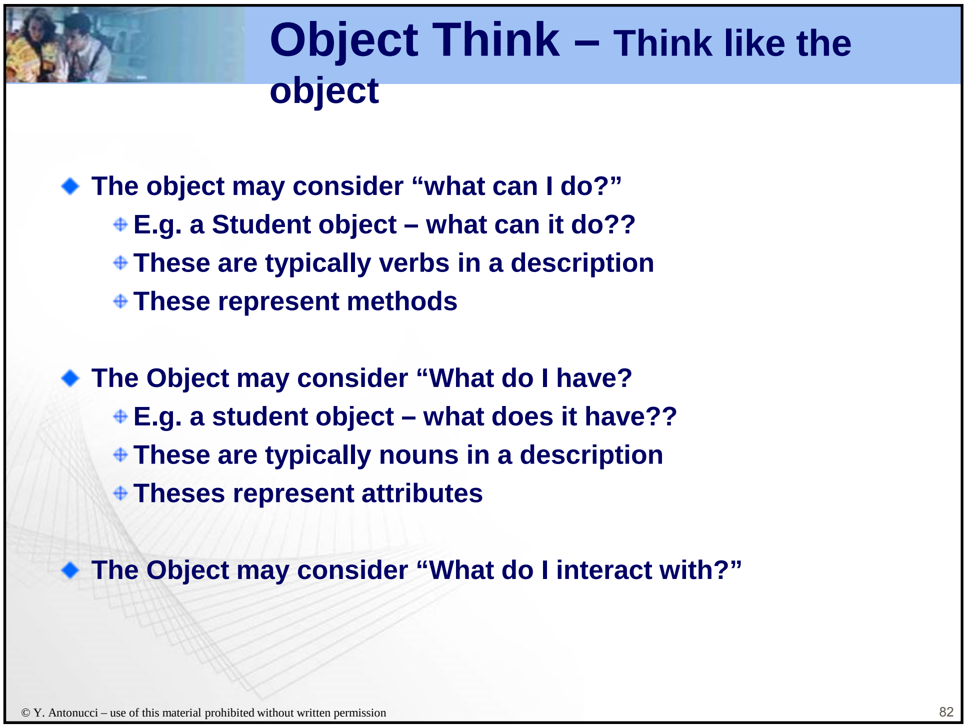
80



## Responsibilities

- A *responsibility* is a contract or an obligation of a class.
- Responsibilities relate to actions
- Responsibilities can be identified by selecting the <u>verbs</u> from the narrative model.
  - Not all verbs in the narrative model may end up as responsibilities.
  - You may have to combine several verbs to find an actual responsibility.
  - Some responsibilities ultimately chosen may not be in the original narrative model.
  - This is an iterative process

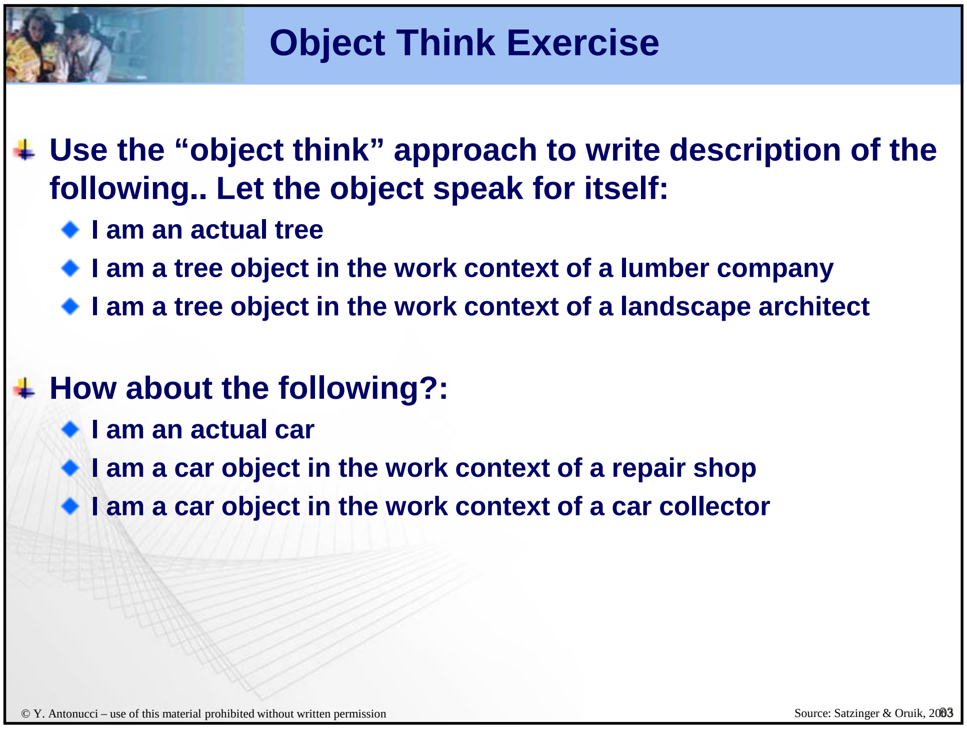
81



## Object Think – Think like the object

- The object may consider “what can I do?”
  - E.g. a Student object – what can it do??
  - These are typically verbs in a description
  - These represent methods
- The Object may consider “What do I have?
  - E.g. a student object – what does it have??
  - These are typically nouns in a description
  - Theses represent attributes
- The Object may consider “What do I interact with?”

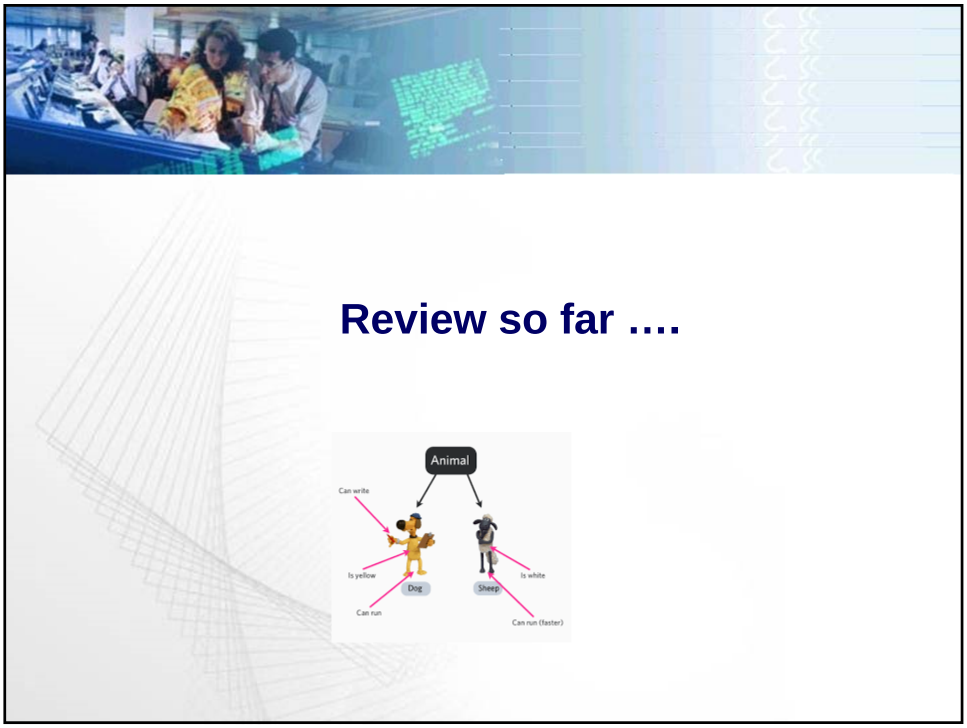
82



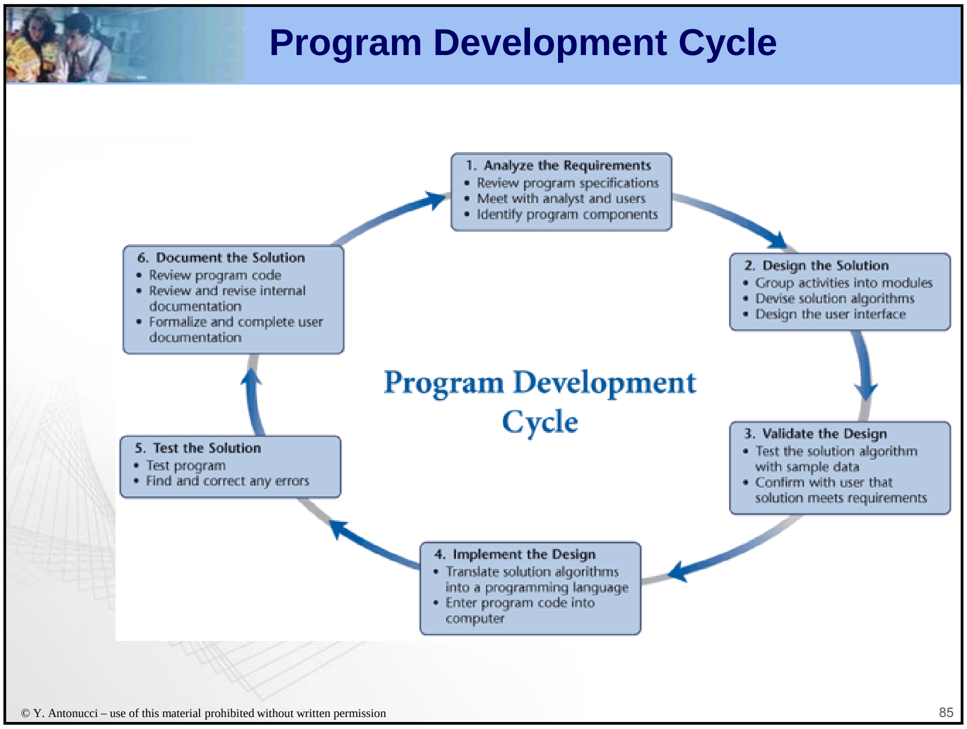
## Object Think Exercise

- Use the “object think” approach to write description of the following.. Let the object speak for itself:
  - I am an actual tree
  - I am a tree object in the work context of a lumber company
  - I am a tree object in the work context of a landscape architect
- How about the following?:
  - I am an actual car
  - I am a car object in the work context of a repair shop
  - I am a car object in the work context of a car collector

Source: Satzinger & Oruik, 2003



## Review so far ….



## Program Development Cycle

Program Development Cycle

1. Analyze the Requirements
- Review program specifications
- Meet with analyst and users
- Identify program components

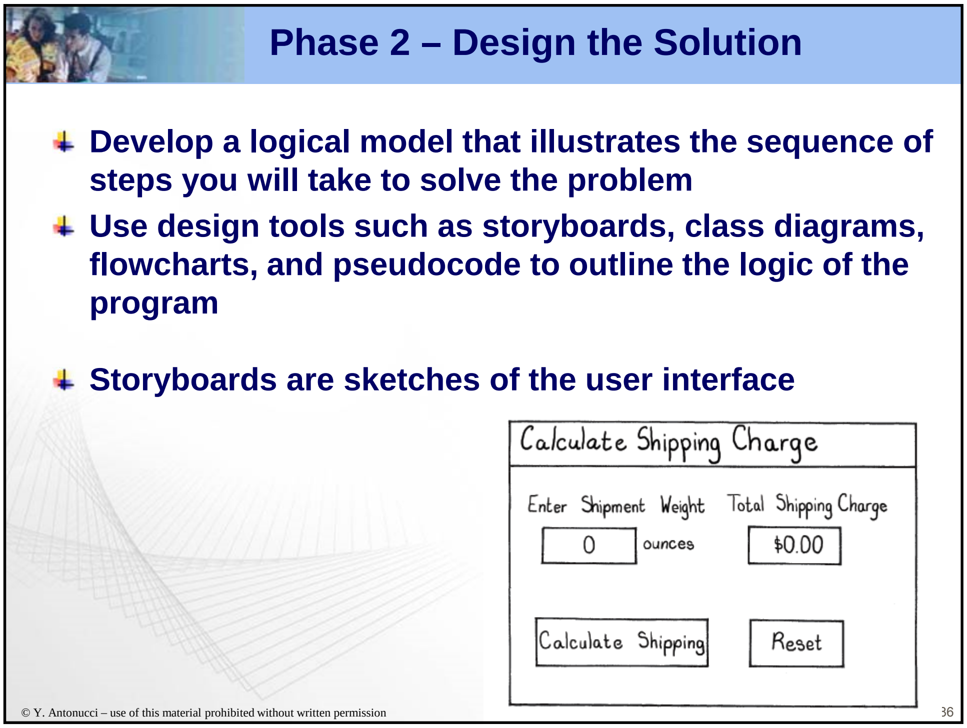
2. Design the Solution
- Group activities into modules
- Devise solution algorithms
- Design the user interface

3. Validate the Design
- Test the solution algorithm with sample data
- Confirm with user that solution meets requirements

4. Implement the Design
- Translate solution algorithms into a programming language
- Enter program code into computer

5. Test the Solution
- Test program
- Find and correct any errors

6. Document the Solution
- Review program code
- Review and revise internal documentation
- Formalize and complete user documentation



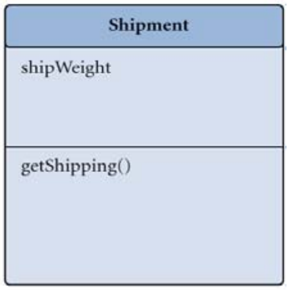
## Phase 2 – Design the Solution

- Develop a logical model that illustrates the sequence of steps you will take to solve the problem
- Use design tools such as storyboards, class diagrams, flowcharts, and pseudocode to outline the logic of the program
- Storyboards are sketches of the user interface

Calculate Shipping Charge
Enter Shipment Weight
0
ounces
Total Shipping Charge
$0.00
Calculate Shipping
Reset

## Phase 2 – Design the Solution

- Class Diagrams illustrate the attributes and methods of a class of objects
  - Attributes define the characteristics of a class
  - Methods are instructions a class uses to manipulate values, generate outputs, or perform actions



Flowcharts graphically represent the logic used to develop an algorithm

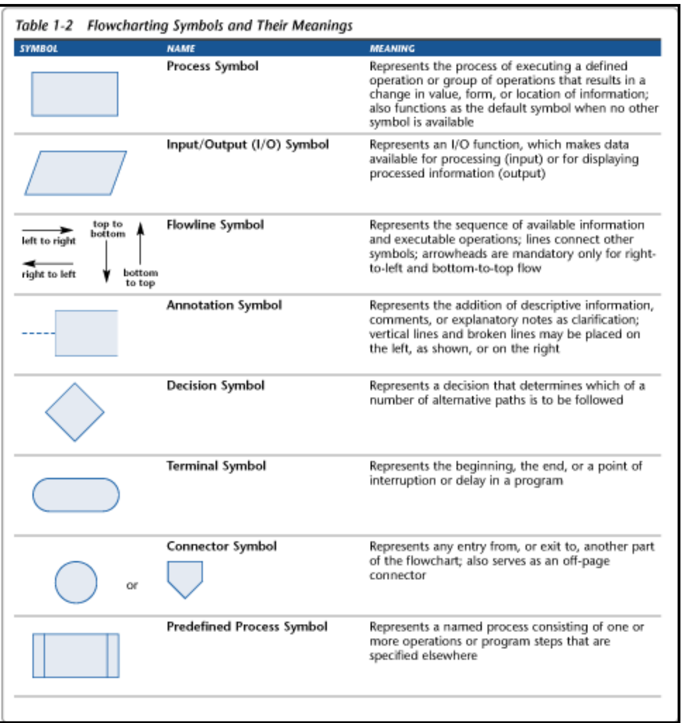
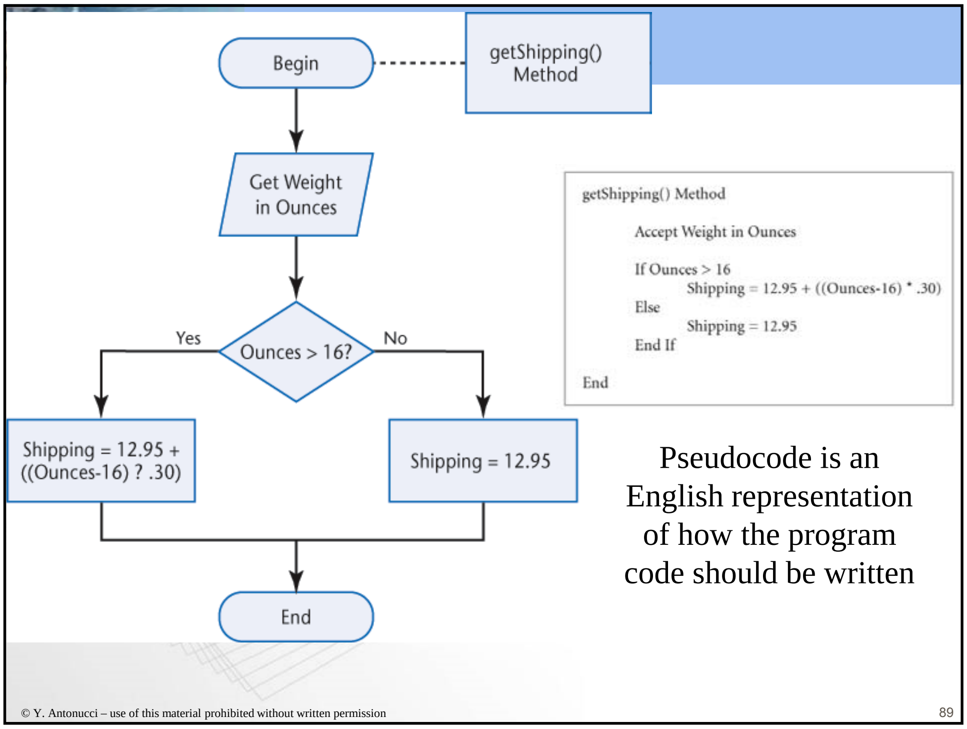


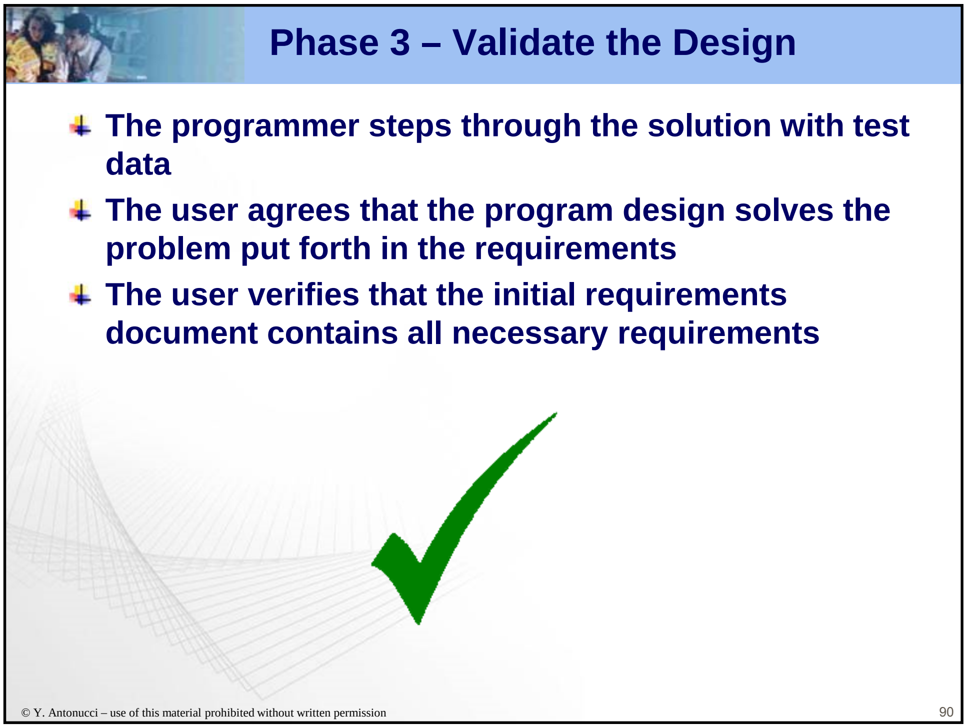
*Table 1-2 Flowcharting Symbols and Their Meanings*

| SYMBOL | NAME | MEANING |
|---|---|---|
| ▭ | Process Symbol | Represents the process of executing a defined operation or group of operations that results in a change in value, form, or location of information; also functions as the default symbol when no other symbol is available |
| ▱ | Input/Output (I/O) Symbol | Represents an I/O function, which makes data available for processing (input) or for displaying processed information (output) |
| → left to right<br>← right to left<br>↓ top to bottom<br>↑ bottom to top | Flowline Symbol | Represents the sequence of available information and executable operations; lines connect other symbols; arrowheads are mandatory only for right-to-left and bottom-to-top flow |
| - - -▭ | Annotation Symbol | Represents the addition of descriptive information, comments, or explanatory notes as clarification; vertical lines and broken lines may be placed on the left, as shown, or on the right |
| ◇ | Decision Symbol | Represents a decision that determines which of a number of alternative paths is to be followed |
| ⬭ | Terminal Symbol | Represents the beginning, the end, or a point of interruption or delay in a program |
| ○ or ⬠ | Connector Symbol | Represents any entry from, or exit to, another part of the flowchart; also serves as an off-page connector |
| ▥ | Predefined Process Symbol | Represents a named process consisting of one or more operations or program steps that are specified elsewhere |



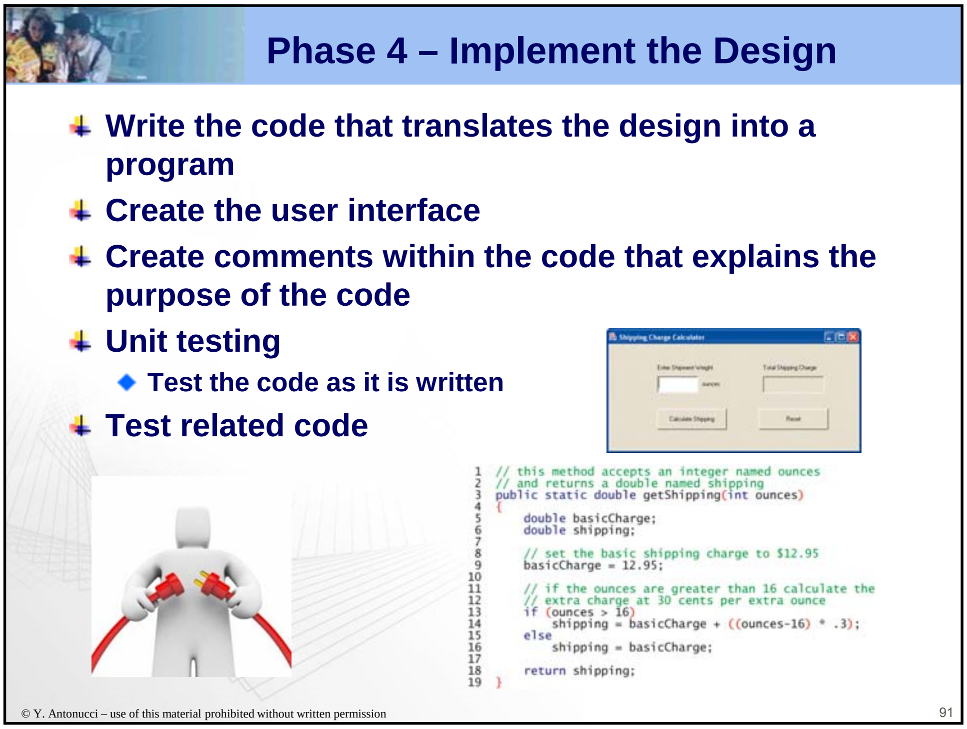
```
getShipping() Method

        Accept Weight in Ounces

        If Ounces > 16
                Shipping = 12.95 + ((Ounces-16) * .30)
        Else
                Shipping = 12.95
        End If

End
```

Pseudocode is an English representation of how the program code should be written



## Phase 3 – Validate the Design

- The programmer steps through the solution with test data
- The user agrees that the program design solves the problem put forth in the requirements
- The user verifies that the initial requirements document contains all necessary requirements

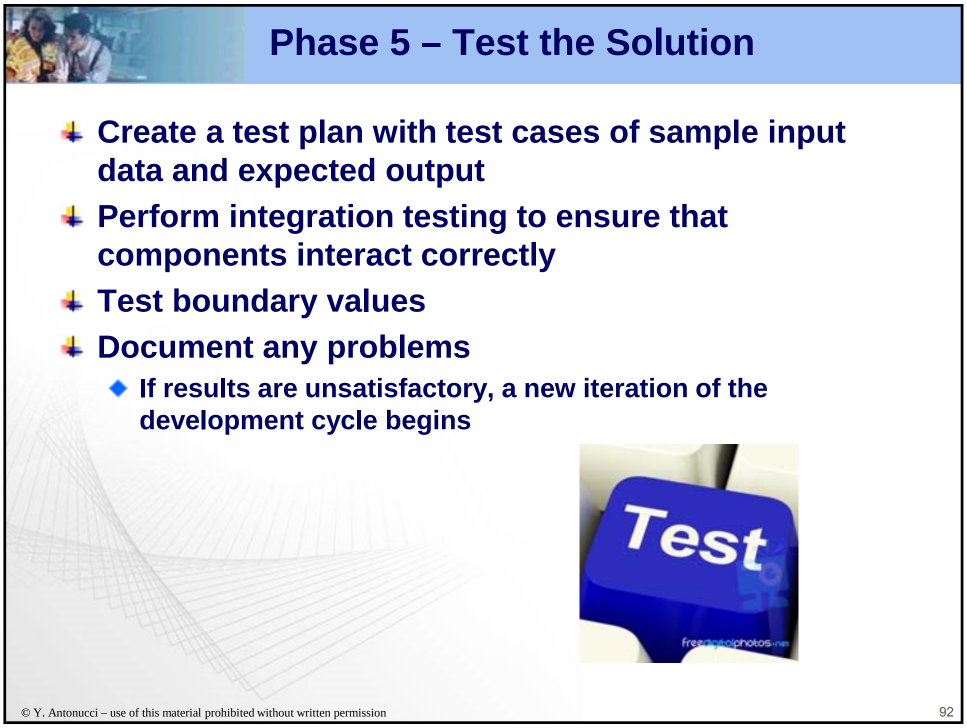


## Phase 4 – Implement the Design

- Write the code that translates the design into a program
- Create the user interface
- Create comments within the code that explains the purpose of the code
- Unit testing
  - Test the code as it is written
- Test related code

Shipping Charge Calculator
Enter Shipment Weight
ounces
Total Shipping Charge
Calculate Shipping
Reset

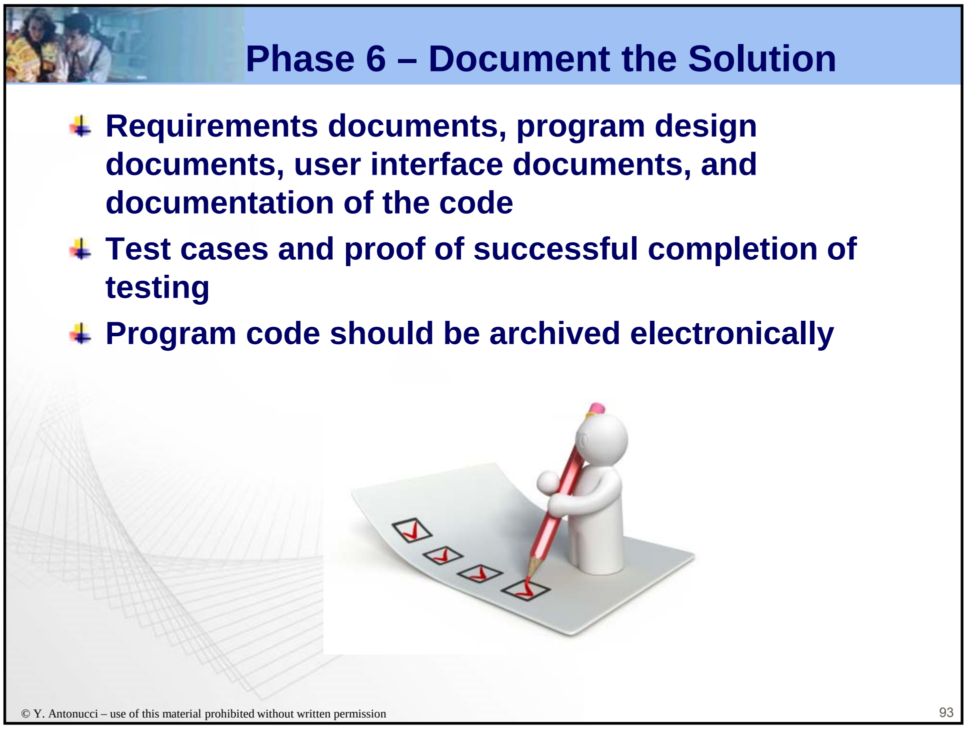
```
// this method accepts an integer named ounces
// and returns a double named shipping
public static double getShipping(int ounces)
{
    double basicCharge;
    double shipping;

    // set the basic shipping charge to $12.95
    basicCharge = 12.95;

    // if the ounces are greater than 16 calculate the
    // extra charge at 30 cents per extra ounce
    if (ounces > 16)
        shipping = basicCharge + ((ounces-16) * .3);
    else
        shipping = basicCharge;

    return shipping;
}
```



## Phase 5 – Test the Solution

- Create a test plan with test cases of sample input data and expected output
- Perform integration testing to ensure that components interact correctly
- Test boundary values
- Document any problems
  - If results are unsatisfactory, a new iteration of the development cycle begins

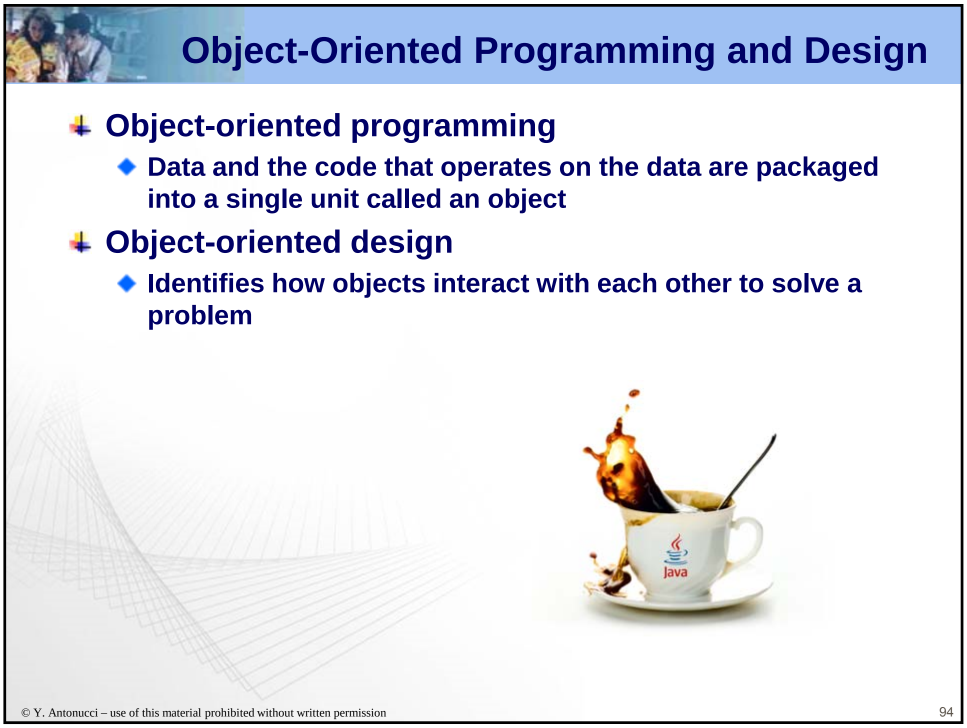
Test
freedigitalphotos.net



## Phase 6 – Document the Solution

- Requirements documents, program design documents, user interface documents, and documentation of the code
- Test cases and proof of successful completion of testing
- Program code should be archived electronically

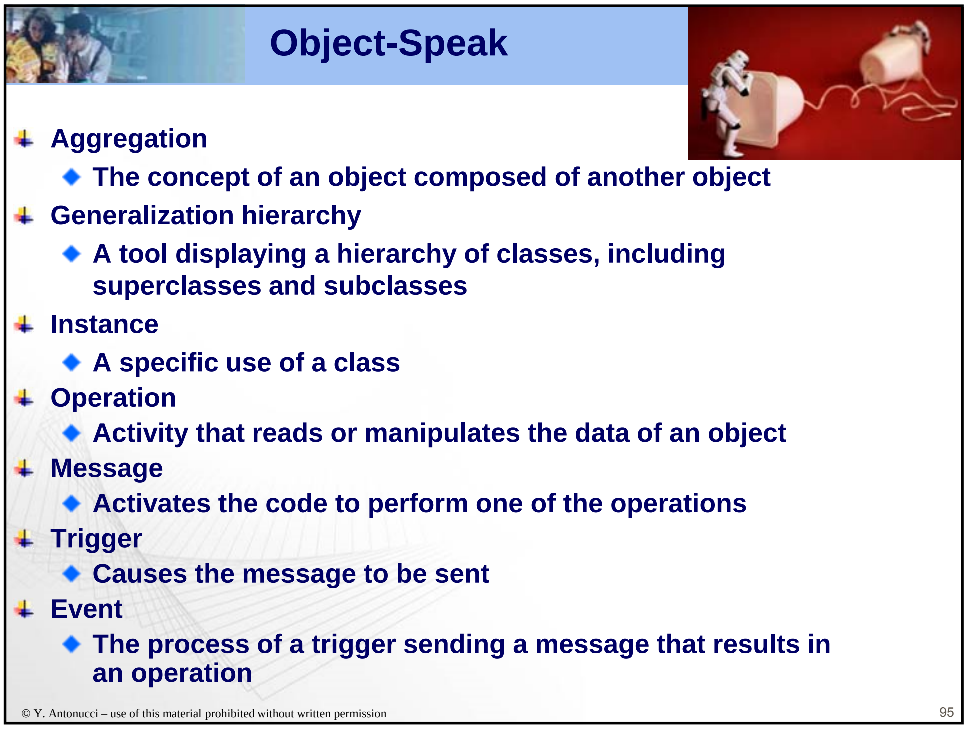
93



## Object-Oriented Programming and Design

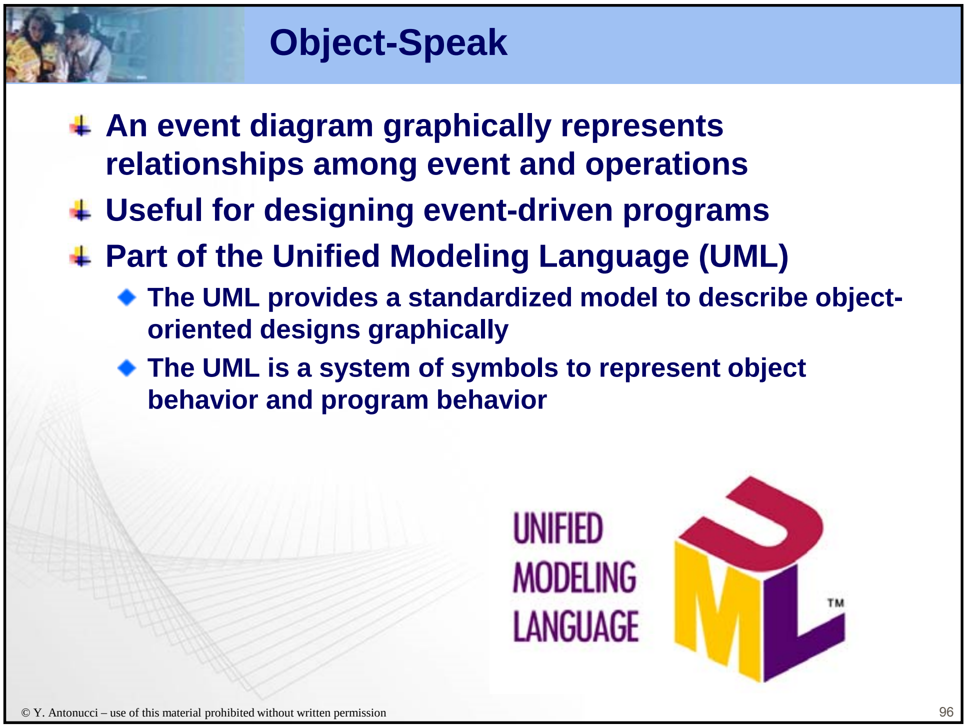
- Object-oriented programming
  - Data and the code that operates on the data are packaged into a single unit called an object
- Object-oriented design
  - Identifies how objects interact with each other to solve a problem

94



## Object-Speak

- Aggregation
  - The concept of an object composed of another object
- Generalization hierarchy
  - A tool displaying a hierarchy of classes, including superclasses and subclasses
- Instance
  - A specific use of a class
- Operation
  - Activity that reads or manipulates the data of an object
- Message
  - Activates the code to perform one of the operations
- Trigger
  - Causes the message to be sent
- Event
  - The process of a trigger sending a message that results in an operation



## Object-Speak

- An event diagram graphically represents relationships among event and operations
- Useful for designing event-driven programs
- Part of the Unified Modeling Language (UML)
  - The UML provides a standardized model to describe object-oriented designs graphically
  - The UML is a system of symbols to represent object behavior and program behavior

UNIFIED
MODELING
LANGUAGE
TM

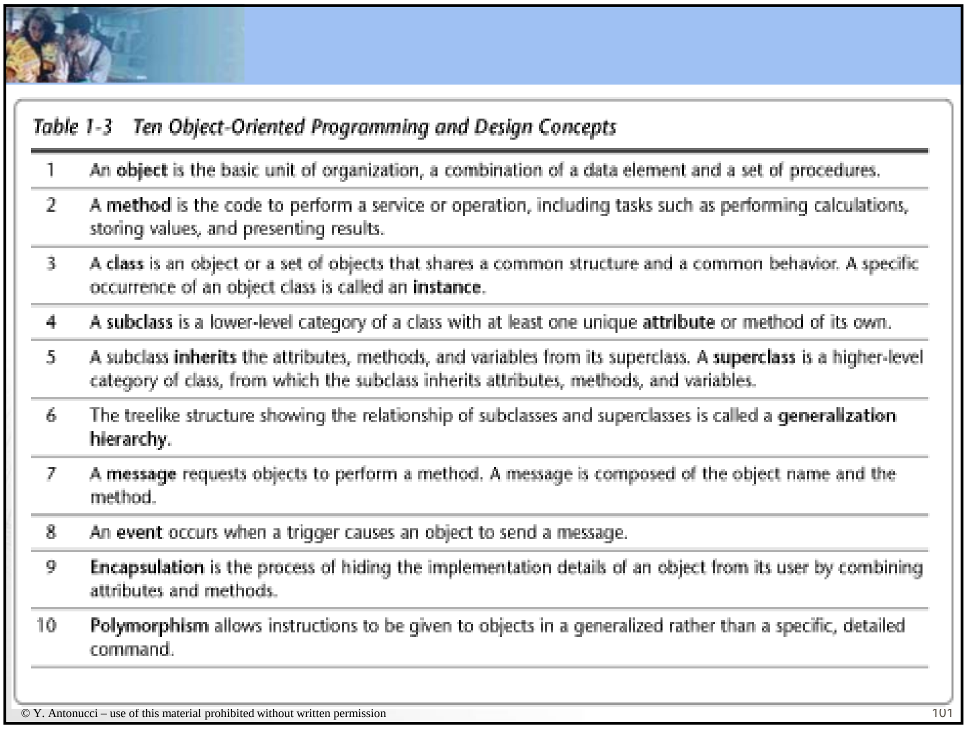
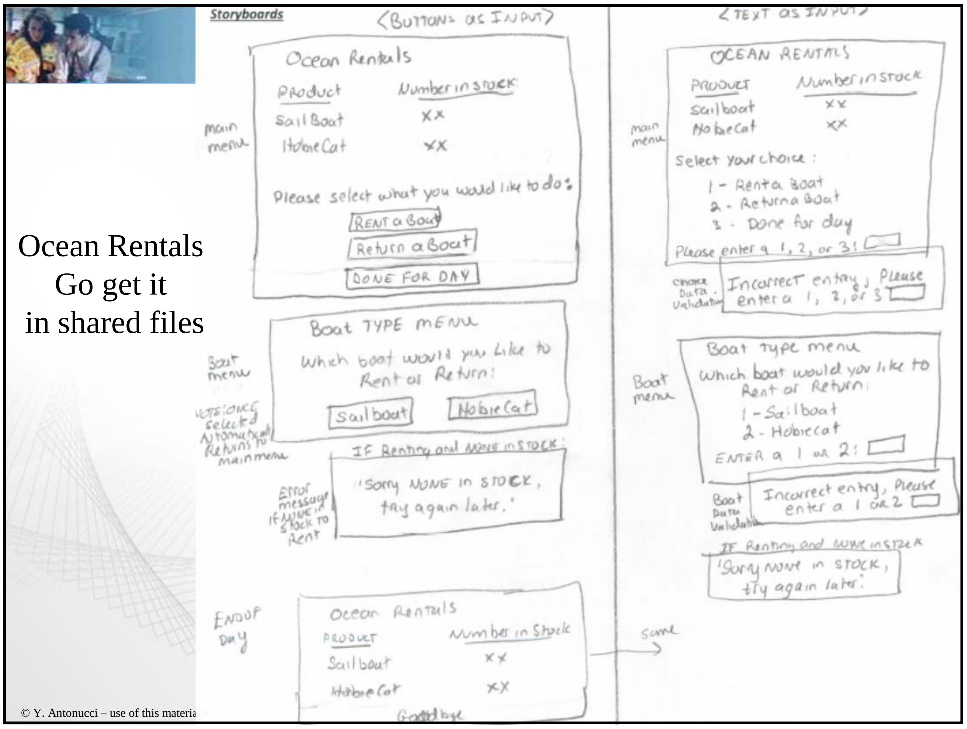


*Table 1-3 Ten Object-Oriented Programming and Design Concepts*

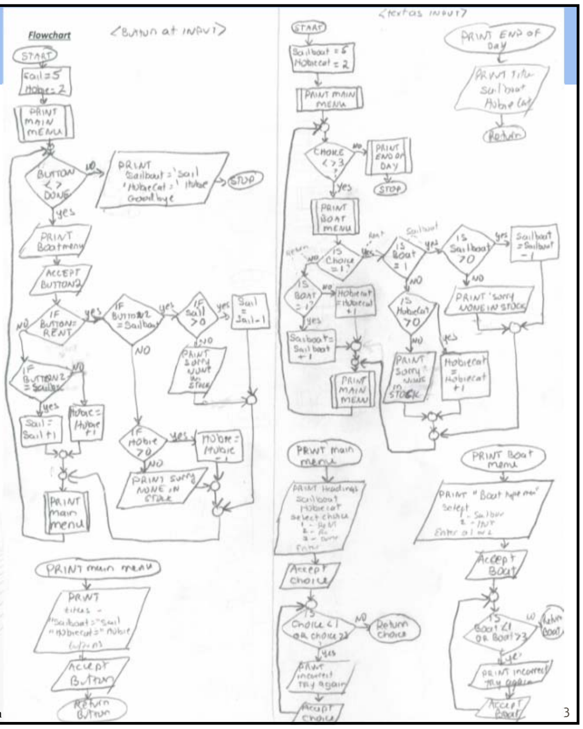
| | |
|---|---|
| 1 | An **object** is the basic unit of organization, a combination of a data element and a set of procedures. |
| 2 | A **method** is the code to perform a service or operation, including tasks such as performing calculations, storing values, and presenting results. |
| 3 | A **class** is an object or a set of objects that shares a common structure and a common behavior. A specific occurrence of an object class is called an **instance**. |
| 4 | A **subclass** is a lower-level category of a class with at least one unique **attribute** or method of its own. |
| 5 | A subclass **inherits** the attributes, methods, and variables from its superclass. A **superclass** is a higher-level category of class, from which the subclass inherits attributes, methods, and variables. |
| 6 | The treelike structure showing the relationship of subclasses and superclasses is called a **generalization hierarchy**. |
| 7 | A **message** requests objects to perform a method. A message is composed of the object name and the method. |
| 8 | An **event** occurs when a trigger causes an object to send a message. |
| 9 | **Encapsulation** is the process of hiding the implementation details of an object from its user by combining attributes and methods. |
| 10 | **Polymorphism** allows instructions to be given to objects in a generalized rather than a specific, detailed command. |

101



Ocean Rentals
Go get it
in shared files

What do you see as Different between these Two designs?



## Java Basics…

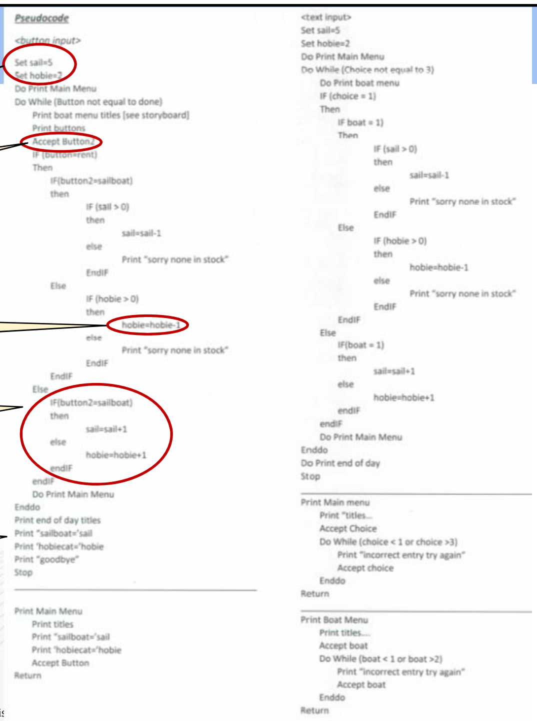
VARIABLES: How to Declare them, data types, assign values…

INPUT: How to..
There are several types:
- User Input (text and object entry)
- Files
- Databases

CALCULATIONS: Mathematical Operations

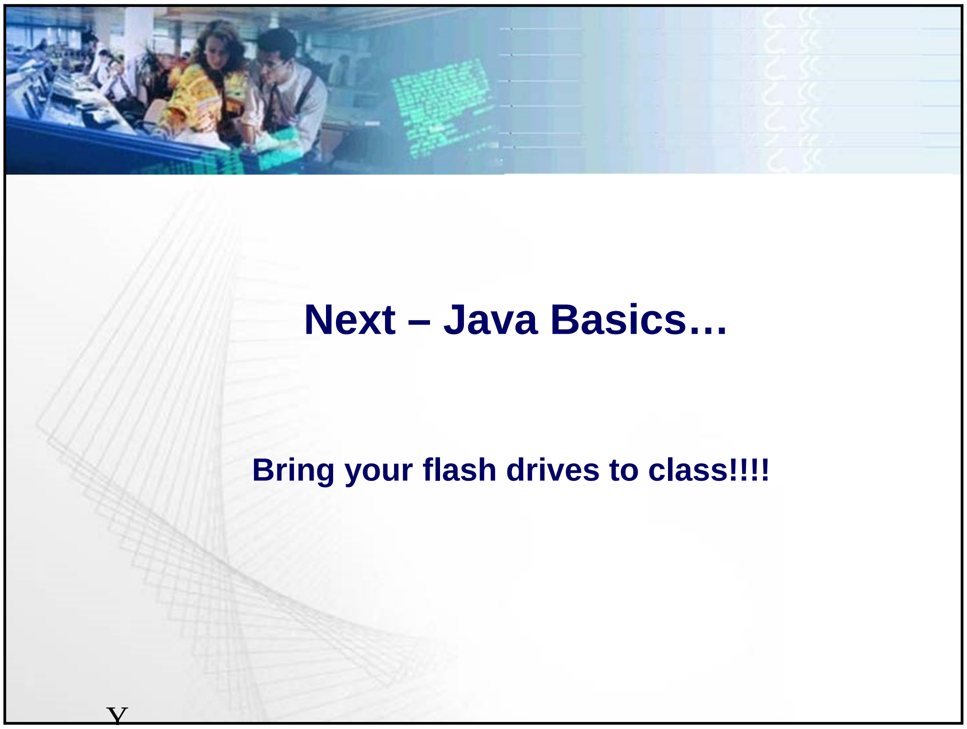
DECISIONS and LOOPS

OUTPUT: How to print, display…



```
Pseudocode

<button input>
Set sail=5
Set hobie=2
Do Print Main Menu
Do While (Button not equal to done)
     Print boat menu titles [see storyboard]
     Print buttons
     Accept Button2
     IF (button=rent)
     Then
          IF(button2=sailboat)
          then
                    IF (sail > 0)
                    then
                              sail=sail-1
                    else
                              Print "sorry none in stock"
                    EndIF
          Else
                    IF (hobie > 0)
                    then
                              hobie=hobie-1
                    else
                              Print "sorry none in stock"
                    EndIF
          EndIF
     Else
          IF(button2=sailboat)
          then
                    sail=sail+1
          else
                    hobie=hobie+1
          endIF
     endIF
     Do Print Main Menu
Enddo
Print end of day titles
Print "sailboat='sail
Print 'hobiecat='hobie
Print "goodbye"
Stop
________________________________
Print Main Menu
     Print titles
     Print "sailboat='sail
     Print 'hobiecat='hobie
     Accept Button
Return
```

```
<text input>
Set sail=5
Set hobie=2
Do Print Main Menu
Do While (Choice not equal to 3)
     Do Print boat menu
     IF (choice = 1)
     Then
          IF boat = 1)
          Then
                    IF (sail > 0)
                    then
                              sail=sail-1
                    else
                              Print "sorry none in stock"
                    EndIF
          Else
                    IF (hobie > 0)
                    then
                              hobie=hobie-1
                    else
                              Print "sorry none in stock"
                    EndIF
          EndIF
     Else
          IF(boat = 1)
          then
                    sail=sail+1
          else
                    hobie=hobie+1
          endIF
     endIF
     Do Print Main Menu
Enddo
Do Print end of day
Stop
________________________________
Print Main menu
     Print "titles…
     Accept Choice
     Do While (choice < 1 or choice >3)
          Print "incorrect entry try again"
          Accept choice
     Enddo
Return
________________________________
Print Boat Menu
     Print titles….
     Accept boat
     Do While (boat < 1 or boat >2)
          Print "incorrect entry try again"
          Accept boat
     Enddo
Return
```



## Next – Java Basics…

**Bring your flash drives to class!!!!**

Y