




# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

---

Instructor Notes:

	
IBM Software Group	
<b>Essentials of Visual Modeling with UML 2.0</b> <b>Module 6: Class Diagrams</b>	
	
	

# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

---

## Instructor Notes:

Introduce the objectives for this module to the students.

## Objectives

- ♦ Describe the static view of the system and show how to capture it in a model.
- ♦ Demonstrate how to read and interpret a class diagram.
- ♦ Model an association and aggregation and show how to model it in a class diagram.
- ♦ Model generalization on a class diagram.

2



# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

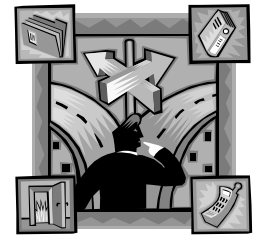
---

## Instructor Notes:

The purpose of this section is to introduce, not teach everything about class diagrams. Remember, this is new to the students.

## Where Are We?

- ☆ ♦ Class diagrams
- ♦ Class relationships
  - Association
  - Aggregation
  - Generalization



3



# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

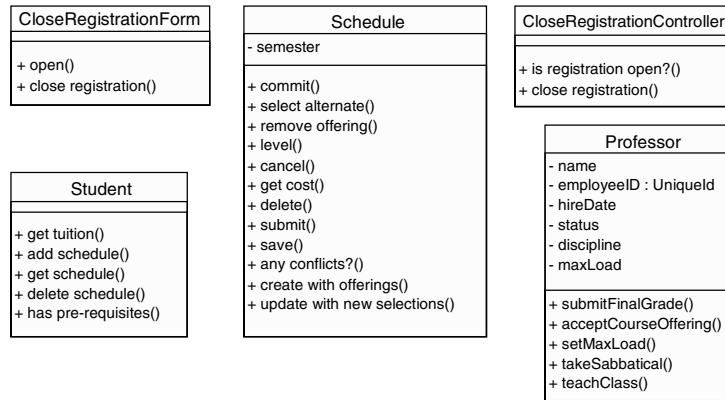
## Instructor Notes:

- ☆ Introduce class diagrams to the students.

Interaction diagrams show the dynamic aspects of a system, while class diagrams show the static aspects of a system.

## What Is a Class Diagram?

### ◆ Static view of a system



A **class diagram** shows the existence of classes and their relationships in the logical design of a system. A class diagram may represent all or part of the class structure of a system.

Class diagrams show the static structure of the model, in particular, the things that exist such as classes, their internal structure, and their relationships to other classes. Class diagrams do not show temporal information.

The static view of a system primarily supports the functional requirements of a system.

# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

---

## Instructor Notes:

Set the context for this module and present some reasons for using class diagrams.

Class diagrams are not entity-relationship diagrams. They go one step further with behavior.

The database schema may not always match 1:1 with the class diagram, but they should be fairly close.

## Class Diagram Usage

- ♦ When modeling the static view of a system, class diagrams are typically used in one of three ways, to model:
  - The vocabulary of a system
  - Collaborations
  - A logical database schema

5



Class diagrams allow you to model the vocabulary of your system when you determine the abstractions that are part of, or outside of, the boundaries of your system. Class diagrams specify these abstractions and their responsibilities.

A **collaboration** is a grouping of classes and other elements that work together to provide a solution that is bigger than the sum of the elements in the collaboration. No class stands alone, but works in collaboration with other elements to carry out some sort of solution. Class diagrams are one way to model these collaborations.

A **database schema** is similar to the blueprints for the conceptual design of a database. Many of the systems that you'll design have persistent objects, which means that they have to be stored in a database for later retrieval. You can model schemas for these databases using class diagrams. The UML's class diagrams are a superset of entity-relationship (E-R) diagrams. However, where typical E-R diagrams focus only on data, class diagrams take it one step further and allow the modeling of behavior, too. Behavior, modeled as operations, are generally turned into triggers or stored procedures on the database.

# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

## Instructor Notes:

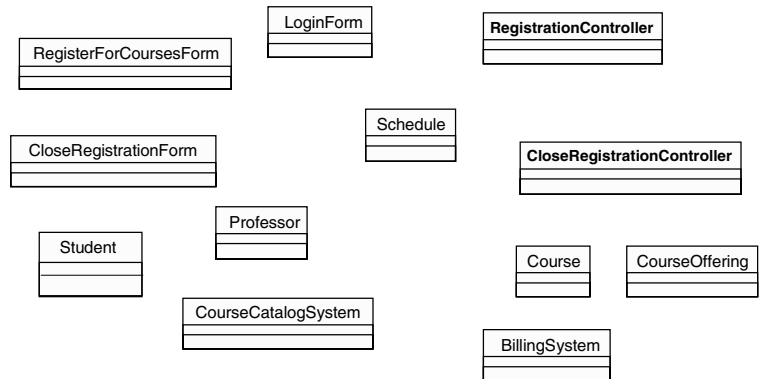
Provide an example of a class diagram.

Ask: how many classes have you worked with on your largest project? Explain how difficult it can become to keep up with all of these classes.

Offer examples from your own experience.

## Example: Class Diagram

- ♦ Is there a better way to organize class diagrams?



6



It's not unusual for a system under development to contain hundreds, even thousands of different classes. Managing such large numbers generates its own problems. How can you organize classes and not lose the organization of the model?

# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

## Instructor Notes:

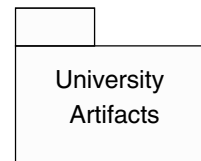
A package is really a “virtual bag.” You can place just about anything inside of it that you like.

Packages allow us to organize our models into pieces that make sense.

They support the OO concepts of hierarchy and modularity.

## Review: What Is a Package?

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



7



A **Package** can be defined as:

A general purpose mechanism for organizing elements into groups. (*The Unified Modeling Language User Guide*, Booch, 1999.)

- Models can have hundreds, even thousands, of model elements. The sheer number of these elements can quickly become overwhelming. It's critical to group model elements into logical collections to keep the model manageable and readable.
- Packages are a generic mechanism for grouping elements into semantically related groups. A package contains classes that are needed by a number of different packages, but are treated as a “behavioral unit.”
- A package is simply a grouping mechanism. No semantics are defined for its instances. Thus, packages do not necessarily have a representation in implementation (except perhaps to represent a directory).
- In the UML, a package is represented as a tabbed folder.
- Package diagrams depict dependencies between packages and are now formalized in UML 2.

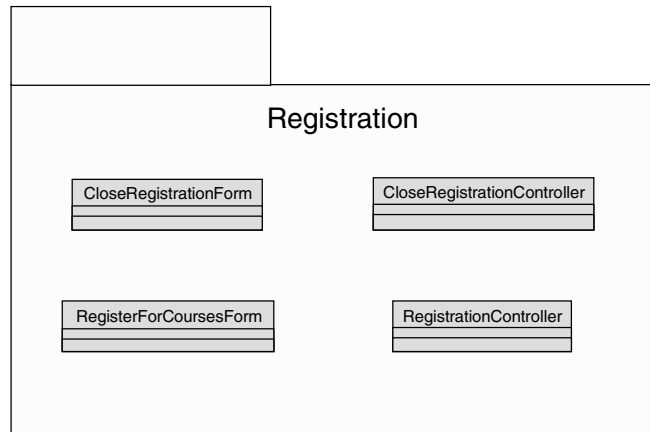
# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

---

## Instructor Notes:

Show the students how a package would look on a class diagram.

### Example: Registration Package



These four classes - CloseRegistrationForm, RegisterForCoursesForm, CloseRegistrationController, and RegistrationController - have all been assigned to the Registration package because they are highly cohesive.



# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

---

## Instructor Notes:

The goal here is to introduce the students to some basic relationship types.

## Where Are We?

- ◆ Class diagrams
- ◆ Class relationships
  - ☆
    - Association
    - Aggregation
    - Generalization



9



# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

## Instructor Notes:

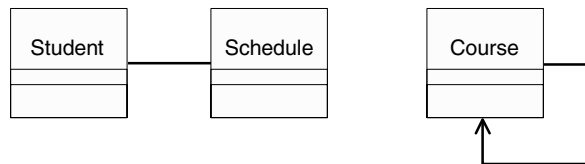
☆ Introduce the concept of association to the students. This is new and may pose trouble for some students.

There are also dependency relationships. However, dependencies are beyond the scope of this course since they are contingent on parameter, local variable or global reference. These topics are discussed in the Object-Oriented Analysis and Design course. For the beginner, understanding associations is enough.

Draw these examples on the board as objects to help students understand the concept.

## What Is an Association?

- ♦ The semantic relationship between two or more classifiers that specifies connections among their instances.
- ♦ A structural relationship specifying that objects of one thing are connected to objects of another thing.



10

IBM

An **Association** can be defined as:

The semantic relationship between two or more classifiers that specifies connections among their instances. In other words, an association is a structural relationship that specifies that objects (instances of classes) of one thing are connected to objects of another thing.

- The way that you show these structural relationships between classes is through the use of associations. Associations are represented on class diagrams by a line connecting the associating classes. Data may flow in either direction or both directions across a link.
- Most associations are simple. That is, they exist between exactly two classes. They are drawn as solid paths connecting pairs of class symbols. Ternary relationships are also possible.
- Sometimes, a class has an association to itself. This does not always mean that an instance of that class has an association to itself. More often, it means that one instance of the class has associations to other instances of the same class.
- This example shows that a student object is related to a schedule object. The second example demonstrates how a course object can be related to another course object.

# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

## Instructor Notes:

Have the students brainstorm what the class diagram looks like.

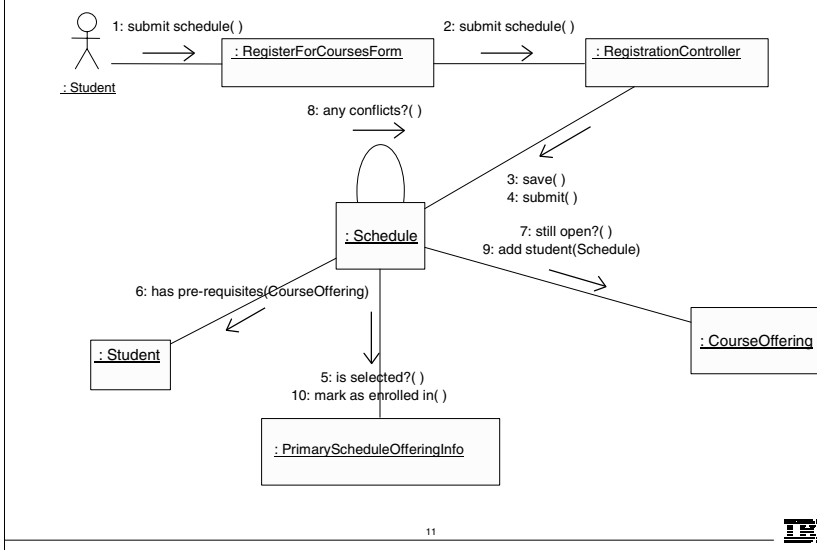
One way to do this is to use the “whiteboard” to display the information that they find.

**Note:** The reflexive message on the Schedule object does not mean that there is a reflexive association on the Schedule class.

Also, the actor is not represented on the class diagram.

The total number of associations that can be found on this slide are 5. The link between the actor and the Form is not an association and the reflexive message on Schedule does not indicate an association to self.

## Example: What Associations Can You Find?



Using the information that you just learned, what do you think the class diagram representing this communication diagram looks like?

# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

## Instructor Notes:

Introduce the concept of multiplicity.

### What Is Multiplicity?

- ♦ Multiplicity is the number of instances one class relates to ONE instance of another class.
- ♦ For each association, there are two multiplicity decisions to make, one for each end of the association.
  - For each instance of Professor, many Course Offerings may be taught.
  - For each instance of Course Offering, there may be either one or zero Professor as the instructor.



12



### **Multiplicity** can be defined as:

The number of instances one class relates to one instance of another class.

- For each role, you can specify the multiplicity of its class and how many objects of the class can be associated with one object of the other class.
- Multiplicity is indicated by a text expression on the role. The expression is a comma-separated list of integer ranges.
- It is important to remember that multiplicity is referring to instances of classes (objects) and their relationships. In this example, a Course Offering object may have either zero or one Professor object related to it. Conversely, a Professor object may have zero or more Course Offering objects related to it.
- Multiplicity must be defined on both ends of the association.

# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

## Instructor Notes:

### Show the valid multiplicity indicators.

The use of "N" instead of "\*" is Booch, not UML (for example, the use of "0..N" and "N" is not UML).

The multiplicity specified for a relationship is for all instances of that relationship, not simply for a particular use-case realization or for a particular point in time.

## Multiplicity Indicators

Unspecified	
Exactly One	1
Zero or More	0..*
Zero or More	*
One or More	1..*
Zero or One (optional value)	0..1
Specified Range	2..4
Multiple, Disjoint Ranges	2, 4..6

13



- Multiplicity is indicated by a text expression on the role.
- The expression is a comma-separated list of integer ranges.
- A range is indicated by an integer (the lower value), two dots, and an integer (the upper value).
- A single integer is a valid range, and the symbol "\*" indicates "many." That is, an asterisk "\*" indicates an unlimited number of objects.
- The symbol "\*" by itself is equivalent to "0..\*" That is, it represents any number including none. This is the default value.
- Optional value has the multiplicity 0..1.

# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

## Instructor Notes:

Provide the students with an opportunity to see how multiplicity can be interpreted from a class diagram.

Here are the answers to the student notes questions.

1. There is an association relationship between each of the classes.

RegisterForCoursesForm to

RegistrationController has a lower and upper bound of 1. Schedule to CourseOffering is 0 (low) to 4 (high).

CourseOffering to Schedule is 0 (low) to many (high).

2. A

RegisterForCoursesForm instance is mandatory because it must be associated with exactly one instance of RegistrationController.

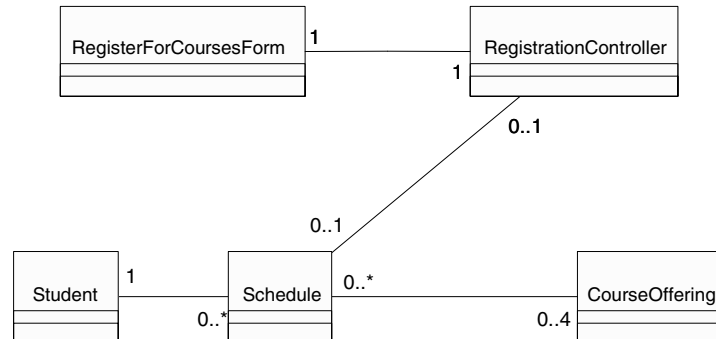
3. Zero to four.

4. One.

5. No, a schedule must be associated to one student.

6. For each RegisterForCoursesForm, there is one RegistrationController which has zero or one Schedule.

## Example: Multiplicity



1. Describe the following relationships between: RegisterForCoursesForm and RegistrationController; Schedule to CourseOffering; and CourseOffering to Schedule. What is the lower and upper bounds for these relationships?
2. Which relationships are mandatory? What do the mandatory relationships tell you about the different classes?
3. How many course offerings can appear on a Schedule?
4. How many students are assigned to each Schedule?
5. Can a Schedule exist without a student?
6. How many schedules can be open on a RegisterForCoursesForm?

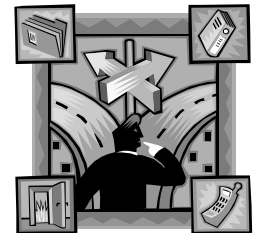
# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

---

Instructor Notes:

## Where Are We?

- ◆ Class diagrams
- ◆ Class relationships
  - Association
  - ☆ ▪ Aggregation
  - Generalization



15



# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

## Instructor Notes:

- ☆ Introduce the concept of aggregation to the students.

Composition is not covered in this course. Composition is a stronger form of aggregation that indicates coincident lifetimes.

## What Is an Aggregation?

- ◆ A special form of association that models a whole-part relationship between the aggregate (the whole) and its parts.
  - An aggregation is an “is a part-of” relationship.
- ◆ Multiplicity is represented like other associations.



16



An **Aggregation** can be defined as:

A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.

- Aggregation is used to model a whole-part relationship between model elements. There are many examples of whole-part relationships: a Library contains Books, Departments are made up of Employees, a Computer is composed of a number of Devices. To model an aggregation, the aggregate (Department) has an aggregation association to its constituent parts (Employee).
- A hollow diamond is attached to the end of an association path on the side of the aggregate (the whole) to indicate aggregation.
- An aggregation relationship that has a multiplicity greater than one for the aggregate is called **shared**. Destroying the aggregate does not necessarily destroy the parts. By implication, a shared aggregation forms a graph or a tree with many roots. Shared aggregations are used when one instance is a part of two other instances. So, the same instance can participate in two different aggregations.



# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

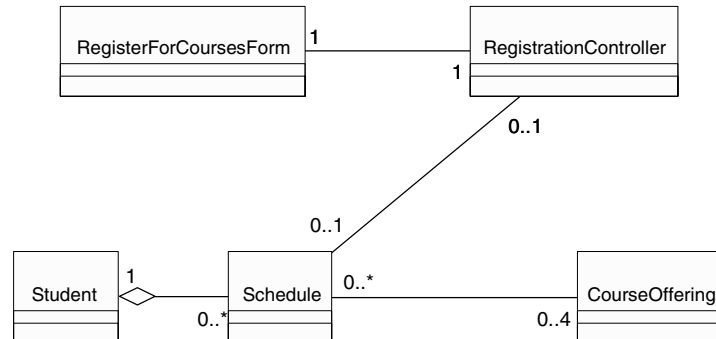
## Instructor Notes:

Give the students an opportunity to see how aggregation can be interpreted from a class diagram.

Have the students answer each of the questions in the student notes. Here are the answers to the student notes questions.

1. The `Schedule` to `Student` instance is an example of an aggregation.
2. `Schedule` is a part of the `Student` aggregate (whole).
3. This relationship is an aggregate because it has a multiplicity greater than one. It models a whole-part relationship.

## Example: Aggregation



17



1. Which relationship is an aggregation?
2. How would you read this aggregate relationship?
3. Why is this relationship an aggregate?

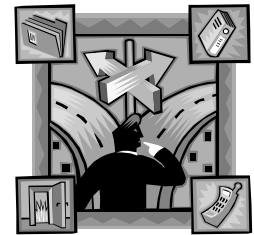
# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

---

Instructor Notes:

## Where Are We?

- ◆ Class diagrams
- ◆ Class relationships
  - Association
  - Aggregation
  - ☆ Generalization



18



# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

## Instructor Notes:

Generalization relationships are also permitted between packages. However, packages do not have semantics. Therefore, generalization between packages is not common. Generalization among subsystems is practical.

According to Grady Booch:

The terms “inheritance” and “generalization” are, practically speaking, interchangeable. The UML standard is to call the relationship “generalization,” so as not to confuse people with language-specific meanings of inheritance.

To confuse matters more, some call this an “is-a” or a “kind of” relationship (especially those into conceptual modeling in the cognitive sciences).

So, for most users, it’s fair to use either term. For power users, people who care about things like the UML metamodel and specifying formal semantics of the same, the relationship is called “generalization” and applying such a relationship between two classes, for example, results in the subclass inheriting the structure and operations of the superclass (inheritance is the mechanism).

## Review: What Is Generalization?

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

19

IBM

### Generalization can be defined as:

A specialization/generalization relationship, in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent). (*The Unified Modeling Language User Guide*, Booch, 1999.)

- The subclass may be used where the superclass is used, but not vice versa. The child inherits from the parent.
- Generalization is transitive. You can always test your generalization by applying the “is a kind of” rule. You should always be able to say that your specialized class “is a kind of” the parent class.
- The terms “generalization” and “inheritance” are generally interchangeable, but if you need to distinguish, generalization is the name of the relationship. Inheritance is the mechanism that the generalization relationship represents/models.

### Inheritance can be defined as:

The mechanism by which more-specific elements incorporate the structure and behavior of more-general elements. (*The Unified Modeling Language User Guide*, Booch, 1999.)

- Single inheritance: The subclass inherits from only one superclass (has only one parent).
- Multiple inheritance: The subclass inherits from more than one superclass (has multiple parents).

# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

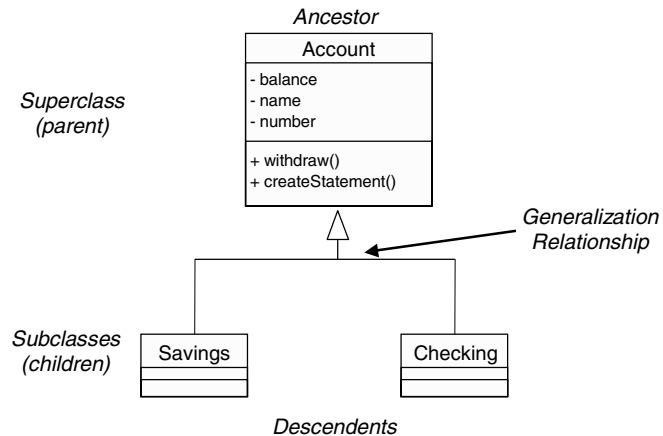
## Instructor Notes:

Demonstrate how generalization looks on a class diagram.

This is the UML representation of generalization that was introduced earlier in the course.

## Example: Single Inheritance

- ◆ One class inherits from another.



The generalization is drawn from the subclass class to the superclass/parent class.

The terms "ancestor" and "descendent" may be used instead of "superclass" and "subclass."

# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

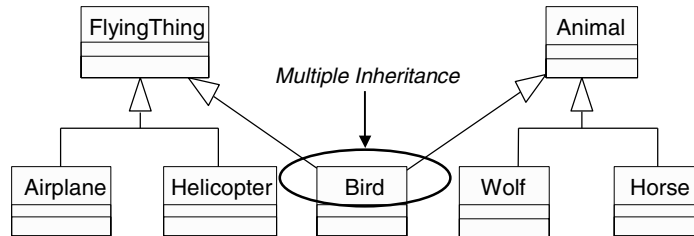
## Instructor Notes:

Demonstrate how generalization looks on a class diagram.

Some languages do not support multiple inheritance. This is the UML representation of generalization that was introduced earlier in the course.

## Example: Multiple Inheritance

- ♦ A class can inherit from several other classes.



***Use multiple inheritance only when needed and always with caution!***

21



Multiple inheritance means that a class can inherit from several other classes. For example, Bird inherits from both FlyingThing and Animal.

Multiple inheritance is conceptually straight forward and may be needed to model the real world accurately. However, there are potential implementation problems when you use multiple inheritance, and not all implementation languages support it. Thus, be judicious with your use of multiple inheritance. Use it only where it accurately describes the concept you are trying to model and reduces the complexity of your model. Be aware, however, that this representation probably needs to be adjusted in design and implementation.

Generally, a class inherits from only one class.

# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

---

## Instructor Notes:

A. A **class diagram** shows the existence of classes and their relationships in the logical design of a system.

B. **Packages** allow us to organize our models into pieces that make sense.

C. An **association** is a structural relationship that specifies that objects (instances of classes) are connected to objects of another. An **aggregation** is a special form of association that models a whole-part relationship between an aggregate (the whole) and its parts. **Generalization** is a relationship among classes where one class shares the structure and/or behavior of one or more classes.

D. Associations are represented on class diagrams by a line connecting the associating classes. Data may flow in either direction or both directions across a link.

E. **Multiplicity** is the number of instances one class relates to ONE instance of another class. For each role, you can specify the multiplicity of its class and how many objects of the class can be associated with one object of the other class.

## Review

- ♦ What does a class diagram represent?
- ♦ What benefits do packages provide to the model?
- ♦ Define association, aggregation, and generalization.
- ♦ How do you find associations?
- ♦ What is multiplicity? What information does multiplicity provide the modeler?



22



# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

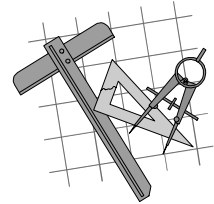
---

## Instructor Notes:

Provide the students with an opportunity to draw their first class diagram.

## Exercise

- ♦ Given:
  - A set of classes and their relationships
- ♦ Draw:
  - A class diagram



23



Document a class diagram using the following information:

- A class diagram containing the following classes: Personal Planner Profile, Personal Planner Controller, Customer Profile, and Buyer Record.
- Associations drawn using the following information:
  1. Each Personal Planner Profile object can be associated with up to one Personal Planner Controller object.
  2. Each Personal Planner Controller object must be related to one Personal Planner Profile.
  3. A Personal Planner Controller object can be associated with up to one Buyer Record and Customer Profile object.
  4. An instance of the Buyer Record class can be related to zero or one Personal Planner Controller.
  5. Zero or one Personal Planner Controller objects are associated with each Customer Profile instance.

# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

---

Instructor Notes:

