

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

---

Instructor Notes:

The cover page features the IBM logo in the top right corner. Below it, the text 'IBM Software Group' is centered. The main title 'Essentials of Visual Modeling with UML 2.0' and subtitle 'Module 3: Concepts of Object Orientation' are prominently displayed. The Rational software logo is centered below the title. A decorative horizontal bar with various icons (including a person, a gear, and a document) spans the width of the page. The bottom section of the cover is a large, empty white space.

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

---

Instructor Notes:

Introduce the objectives for this module.

## Objectives

- ◆ Describe abstraction, encapsulation, modularity, and hierarchy.
- ◆ Describe the physical structure of a class.
- ◆ Describe the relationship between a class and an object.
- ◆ Define polymorphism and generalization.

2



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

---

Instructor Notes:

Introduce the first topic in this module.

## Where Are We?

- ☆ ♦ What is an object?
- ♦ Four principles of OO
- ♦ What is a class?
- ♦ Polymorphism and generalization
- ♦ Organizing model elements



3



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

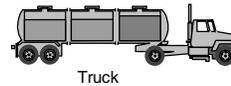
## Instructor Notes:

- ☆ Stress that objects can represent anything.  
Remember, most of your students do not know what an object is.

## What Is an Object?

- ◆ Informally, an object represents an entity, either physical, conceptual, or software.

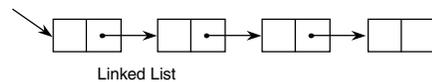
- Physical entity



- Conceptual entity



- Software entity



4

IBM

Objects allow the software developer to represent real-world concepts in their software design. These real-world concepts can represent a physical entity such as a person, truck, or space shuttle.

Objects can be concepts like a chemical process or algorithms.

Object can even represent software entities like a linked list.

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

## Instructor Notes:

☆ Emphasize that state, identity and behavior are the key characteristics of an object.

Take a moment to explain the graphic on this slide.

**Attributes** are documented on the inside of the doughnut.

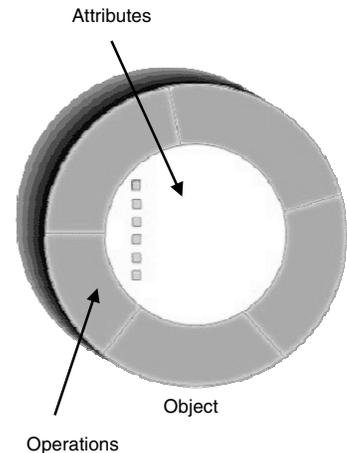
**Operations** are documented on the borders, which become clearer to the student as you discuss topics like encapsulation.

Note that the doughnut is not part of the UML notation. UML notation is discussed later.

## A More Formal Definition

◆ An object is an entity with a well-defined boundary and *identity* that encapsulates *state* and *behavior*.

- State is represented by attributes and relationships.
- Behavior is represented by operations, methods, and state machines.



5



An **object** is an entity that has a well-defined boundary. That is, the purpose of the object should be clear.

An object has two key components: attributes and operations.

Attributes and relationships represent an object's state. Operations represent the behavior of the object.

Object behavior and state are discussed in the next few slides.

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

## Instructor Notes:

Explain the concept of state to the students.

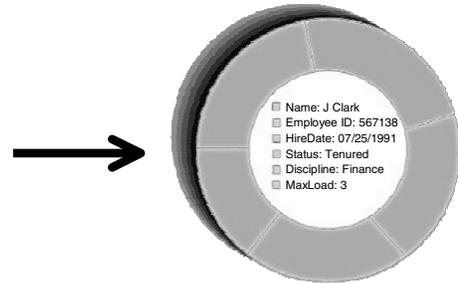
Provide some supplemental state exercises to ensure that the class understands the concept of state. For example an online order (Pending, Approved, Shipped, BackOrdered, Cancelled, etc.) or a car's Cruise Control (Engaged, Disengaged, On, Accelerating, Decelerating).

## An Object Has State

- ◆ State is a condition or situation during the life of an object, which satisfies some condition, performs some activity, or waits for some event.
- ◆ The state of an object normally changes over time.



Name: J Clark  
Employee ID: 567138  
Date Hired: July 25, 1991  
Status: Tenured  
Discipline: Finance  
Maximum Course Load: 3 classes



Professor Clark

6



The **state** of an object is one of the possible conditions in which an object may exist. State normally changes over time.

The state of an object is usually implemented by a set of properties called attributes, along with the values of the properties and the links the object may have with other objects.

State is not defined by a "state" attribute or set of attributes. Instead, state is defined by the total of an object's attributes and links. For example, if Professor Clark's status changed from Tenured to Retired, the state of the Professor Clark object would change.

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

## Instructor Notes:

Explain the concept of behavior to the students.

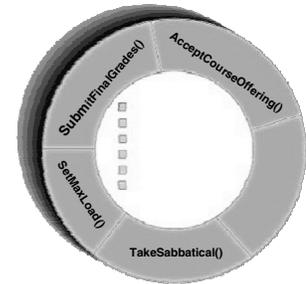
Objects “do” some interesting things. This is where the work in an OO system is done. The melding of data and behavior is what makes OO such a powerful development technique.

## An Object Has Behavior

- ◆ Behavior determines how an object acts and reacts.
- ◆ The visible behavior of an object is modeled by a set of messages it can respond to (operations that the object can perform).



Professor Clark's behavior  
Submit Final Grades  
Accept Course Offering  
Take Sabbatical  
Set Max Load



Professor Clark

7



The second characteristic of an object is that it has **behavior**. Objects are intended to mirror the concepts that they are modeled after, including behavior.

Behavior determines how an object acts and reacts to requests from other objects.

Object behavior is represented by the operations that the object can perform. For example, Professor Clark can choose to take a sabbatical once every five years. The Professor Clark object represents this behavior through the TakeSabbatical() operation.

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

## Instructor Notes:

Explain the concept of identity to the students.

Every object is unique, even if it shares the exact same characteristics of another object.

## An Object Has Identity

- ◆ Each object has a unique identity, even if the state is identical to that of another object.



Professor "J Clark"  
teaches Biology



Professor "J Clark"  
teaches Biology

8



In the real world, two people can share the same characteristics: name, birth date, job description. Yet, there is no doubt that they are two individuals with a unique **identity**.

The same concept holds true for objects. Although two objects may share the same state (attributes and relationships), they are separate, independent objects with their own unique identity.

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

---

Instructor Notes:

## Where Are We?

- ◆ What is an object?
- ☆ ◆ Four principles of OO
- ◆ What is a class?
- ◆ Polymorphism and generalization
- ◆ Organizing model elements



9



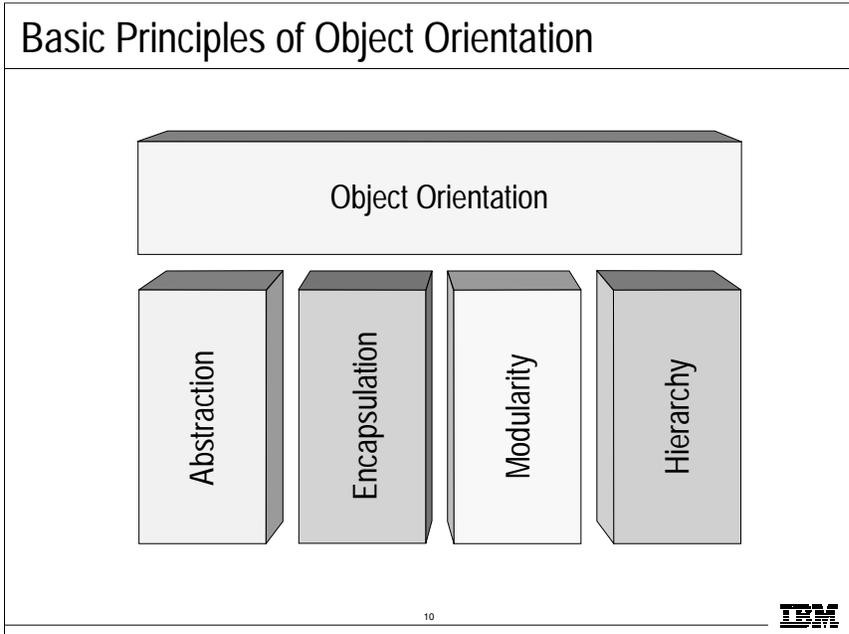
# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

## Instructor Notes:

### Introduce the four basic principles of OO.

Be sure the students understand objects before you begin this next section.

You've introduced objects first to help students better apply each of these principles.



There are four basic principles of object orientation:

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

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

## Instructor Notes:

Explain the concept of abstraction to the students.

A car ("a mobile, powered vehicle for transporting people from place to place") is an example of an abstraction if it suppresses less important details.

The abstract use of car is not concrete. However, if you describe the car as a 1995 Blue Ford Mustang, then it becomes a concrete manifestation and not an abstraction.

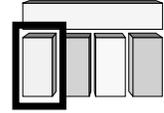
Discuss the makings of a good abstraction.

- Concise
- Single coherent concept

Abstraction removes the unnecessary details to make something easier to understand.

## What Is Abstraction?

- ◆ The essential characteristics of an entity that distinguishes it from all other kinds of entities.
- ◆ Defines a boundary relative to the perspective of the viewer.
- ◆ Is not a concrete manifestation, denotes the ideal essence of something.



11



**Abstraction** can be defined as:

Any model that includes the most important, essential, or distinguishing aspects of something while suppressing or ignoring less important, immaterial, or diversionary details. The result of removing distinctions so as to emphasize commonalities. (*Dictionary of Object Technology*, Firesmith, Eykholt, 1995.)

- Abstraction allows us to manage complexity by concentrating on the essential characteristics of an entity that distinguishes it from all other kind of entities.
- An abstraction is domain and perspective dependent. That is, what is important in one context may not be in another.
- OO allows us to model our system using abstractions from the problem domain (for example, classes and objects).

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

## Instructor Notes:

Explain how these are examples of abstraction.

### Example: Abstraction

Student

Professor

Course Offering (9:00 a.m., Monday-Wednesday-Friday)

Course (e.g. Algebra)

12

The following are examples of abstraction:

- A student is a person enrolled in classes in the university.
- A professor is a person teaching classes at the university.
- A course is a class offered by the university.
- A course offering is a specific offering for a course including the days of the week and the times.

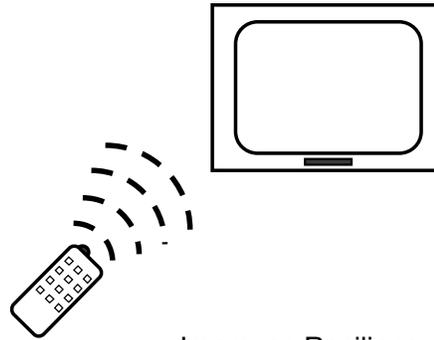
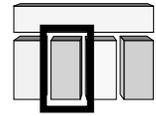
# Essentials of Visual Modeling w/ UML 2.0 - Instructor Notes

## Instructor Notes:

- ☆ • Define the concept of encapsulation to the students.
- Encapsulation is putting the “data bits” and operations that manipulate them in the same place.
- Encapsulation does NOT allow direct manipulation of things that have been encapsulated without using the supplied interface.
- An example is a car’s accelerator. Generally speaking, you put your foot down and the car goes faster. You don’t worry about the cables, electronics, engine, and the rest.

## What Is Encapsulation?

- ◆ Hides implementation from clients.
  - Clients depend on interface.



Improves Resiliency

13



### Encapsulation can be defined as:

The physical localization of features (for example, properties, behaviors) into a single blackbox abstraction that hides their implementation (and associated design decisions) behind a public interface. (*Dictionary of Object Technology*, Firesmith, Eykholt, 1995.)

- Encapsulation is often referred to as “information hiding,” making it possible for the clients to operate without knowing how the implementation implements the interface.
- Encapsulation eliminates direct dependencies on the implementation (clients depend on/use the interface). Thus, it’s possible to change the implementation without updating the clients as long as the interface is unchanged.
- Clients are not affected by changes in implementation, thus reducing the “ripple effect,” where a correction to one operation forces the corresponding correction in a client operation and so on. As a result of encapsulation, maintenance is easier and less expensive.
- Encapsulation offers two kinds of protection. It protects an object’s internal state from being corrupted by its clients and client code from changes in the object’s implementation.

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

## Instructor Notes:

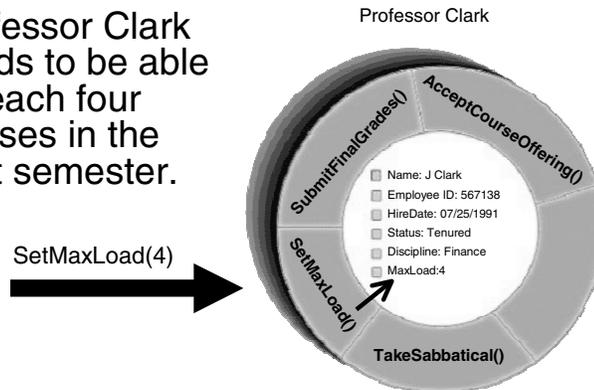
Point out that the requesting object does not need to know the structure of the Professor object to request a state change.

The object that owns the attributes is the only one allowed to change its own attributes.

Note that encapsulation can also be illustrated using interfaces. However, the scope of this course does not include this discussion.

## Encapsulation Illustrated

- ◆ Professor Clark needs to be able to teach four classes in the next semester.



14



The key to encapsulation is an object's **message interface**. The object interface ensures that all communication with the object takes place through a set of predefined operations. Data inside the object is only accessible by the object's operations. No other object can reach inside the object and change its attribute values.

For example, Professor Clark needs to have her maximum course load increased from three classes to four classes per semester. Another object makes a request to Professor Clark to set the maximum course load to four. The attribute, MaxLoad, is then changed by the SetMaxLoad() operation.

Encapsulation is beneficial in this example because the requesting object does not need to know how to change the maximum course load. In the future, the number or variables that are used to define the maximum course load may be increased, but it doesn't affect the requesting object. It depends on the operation interface for the Professor Clark object.

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

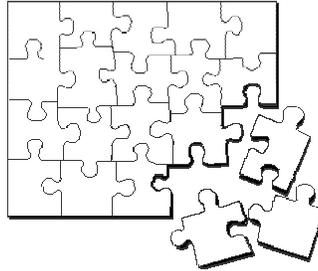
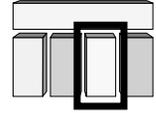
## Instructor Notes:

- ☆ Explain the concept of modularity to the students.

Modularity supports a separation of concerns.

## What Is Modularity?

- ◆ Breaks up something complex into manageable pieces.
- ◆ Helps people understand complex systems.



15



### **Modularity** can be defined as:

The logical and physical decomposition of things (for example, responsibilities and software) into small, simple groupings (for example, requirements and classes, respectively), which increase the achievements of software-engineering goals. (*Dictionary of Object Technology*, Firesmith, Eykholt, 1995.)

- Another way to manage complexity is to break something that is large and complex into a set of smaller, more manageable pieces. These pieces can then be independently developed as long as their interactions are well understood.
- Packages (described later in the course) support the definition of modular components.

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

## Instructor Notes:

A car is an example of modularity. It is made of up of a body, chassis, engine, wheels, and so on.

## Example: Modularity

- ◆ For example, break complex systems into smaller modules.



16



Often, the system under development is too complex to understand. To better understand this, imagine that the system is broken into smaller blocks that are maintained independently. Breaking down a system in this way is called **modularity**. It is critical for understanding a complex system.

For example, the system under development is a Course Registration System. The system itself is too large and abstract to understand the details. Therefore, the development team broke this system into three modular systems, each independent of the others.

- Billing System
- Course Catalog System
- Student Management System

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

## Instructor Notes:

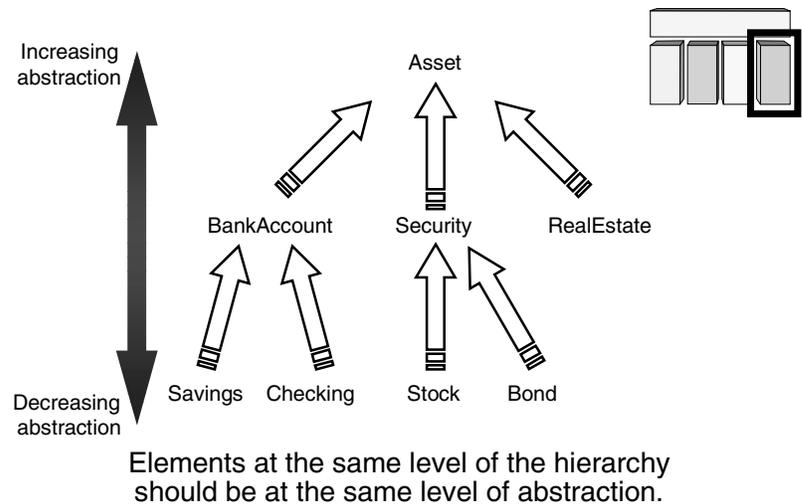
☆ Explain the concept of hierarchy to the students.

Hierarchy is a taxonomic organization. The use of hierarchy makes it easy to recognize similarities and differences.

A taxonomic organization is one that represents an orderly classification. For example, plants and animals are classified taxonomically.

Note that this is not part of the UML notation. UML notation is discussed later.

## What Is Hierarchy?



## Hierarchy can be defined as:

Any ranking or ordering of abstractions into a tree-like structure.  
Kinds: Aggregation hierarchy, class hierarchy, containment hierarchy, inheritance hierarchy, partition hierarchy, specialization hierarchy, type hierarchy. (*Dictionary of Object Technology*, Firesmith, Eykholt, 1995.)

- Hierarchy organizes in a particular order or rank (for example, complexity, responsibility, and so on). This organization is dependent on perspective. Using a hierarchy to describe differences or variations of a particular concept provides for more descriptive and cohesive abstractions and a better allocation of responsibility.
- In any one system, there may be multiple abstraction hierarchies (for example, a financial application may have different types of customers and accounts).
- Hierarchy is neither an organizational chart nor a functional decomposition.
- Hierarchy is a taxonomic organization. The use of hierarchy makes it easy to recognize similarities and differences. For example, botany organizes plants into families. Chemistry organizes elements into a periodic table.

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

## Instructor Notes:

- ☆ Demonstrate how an object is modeled in UML.

Many of your students may now wonder, "Hey, what is a class?" Classes are defined on the next slide.

## Representing Objects in the UML

- ◆ An object is represented as a rectangle with an underlined name.



Professor J Clark

J Clark :  
Professor

Named Object

: Professor

Anonymous Object

18



An object is represented with a rectangle and the name of the class.

The name of the object is always underlined. To name an object, place its name before the colon.

An object can be either named or anonymous. To remain anonymous, do not include a name.

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

---

Instructor Notes:

## Where Are We?

- ◆ What is an object?
- ◆ Four principles of OO
- ☆ ◆ What is a class?
- ◆ Polymorphism and generalization
- ◆ Organizing model elements



19



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

---

## Instructor Notes:

☆ Explain what a class is to the students. Remember, many of your students are not familiar with this term.

The course introduces objects before classes because objects are the things that actually do most of the work. Classes are the templates for the objects.

## What Is a Class?

- ◆ A class is a description of a set of objects that share the same *attributes, operations, relationships, and semantics*.
  - An object is an instance of a class.
- ◆ A class is an abstraction in that it
  - Emphasizes relevant characteristics.
  - Suppresses other characteristics.

20



A **Class** can be defined as:

A description of a set of objects that share the same attributes, operations, relationships, and semantics. (*The Unified Modeling Language User Guide*, Booch, 1999.)

- There are many objects identified for any domain.
- Recognizing the commonalities among the objects and defining classes helps us deal with the potential complexity.
- The OO principle abstraction helps us deal with complexity.

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

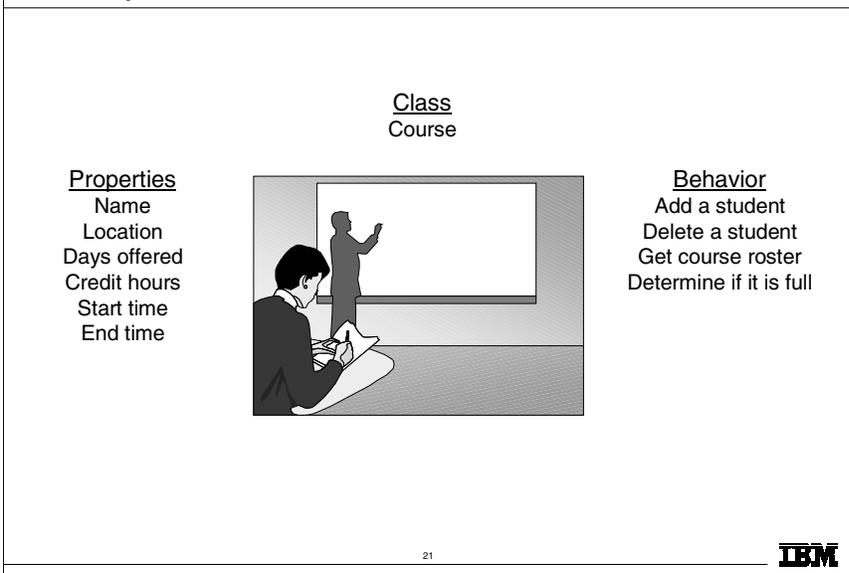
## Instructor Notes:

Point out the differences between a class and an object.

Emphasize that a class is not an object.

These properties are not "filled in" to represent a specific course. Rather, it serves as a template.

## A Sample Class



The class "Course" is an abstraction of the real-world representation of a college course. The class has properties: name, location, days offered, credit hours, start time, and end time. It also has behavior, like adding and deleting a student to the class, retrieving a current course roster, and determining if the course is full.

The class does not represent a specific course like Algebra 101 or Theatre Arts 102. Rather, it is a description of the types of properties and behavior a typical course may have.

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

## Instructor Notes:

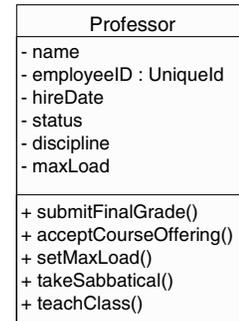
☆ Demonstrate how a class is modeled in UML.

Tell them that the UML represents public visibility with a plus (+) symbol and private visibility with a minus (-) symbol. Do not discuss protected visibility.

## Representing Classes in the UML

◆ A class is represented using a rectangle with three compartments:

- The class name
- The structure (attributes)
- The behavior (operations)



22



The UML notation for a class permits you to see an abstraction apart from any specific programming language, which lets you emphasize the most important parts about an abstraction – its name, attributes, and operations.

Graphically, a class is represented by a rectangle.

The UML represents public visibility with a plus (+) symbol and private visibility with a minus (-) symbol.

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

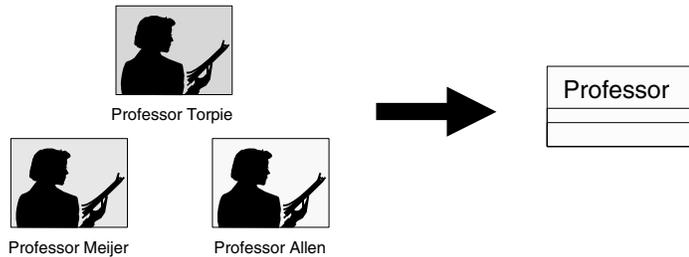
## Instructor Notes:

☆ Show how classes and objects are related.

Classic example: A class is to an object what a cookie cutter is to a cookie.

## The Relationship between Classes and Objects

- ◆ A class is an abstract definition of an object.
  - It defines the structure and behavior of each object in the class.
  - It serves as a template for creating objects.
- ◆ Classes are not collections of objects.



A class is a description of a set of objects that share the same responsibilities, relationships, operations, attributes, and semantics.

A class defines an object. A class defines a template for the structure and behavior of all its objects. The objects created from a class are also called the **instances** of the class.

The class is the static description and the object is the run-time instance of that class.

Modeling is from real-world objects. Software objects are based on the real-world objects, but exist only in the context of the system.

Use real-world objects, abstract out what you don't care about. Then, take these abstractions and go through the process of classification based on what you do care about. Classes in the model are the result of this classification process.

These classes are then used as templates within an executing software system to create software objects. These software objects represent the real-world objects you originally started with.

Some classes/objects may be defined that don't represent real-world objects. They are there to support the design and are "software only."

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

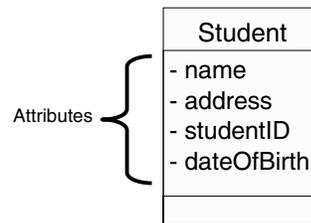
## Instructor Notes:

Define the term attribute for the students.

Remember, there are still operations in this class, but choose to suppress them.

## What Is an Attribute?

- ◆ An attribute is a named property of a class that describes the range of values that instances of the property may hold.
  - A class may have any number of attributes or no attributes at all.



24



An **Attribute** can be defined as:

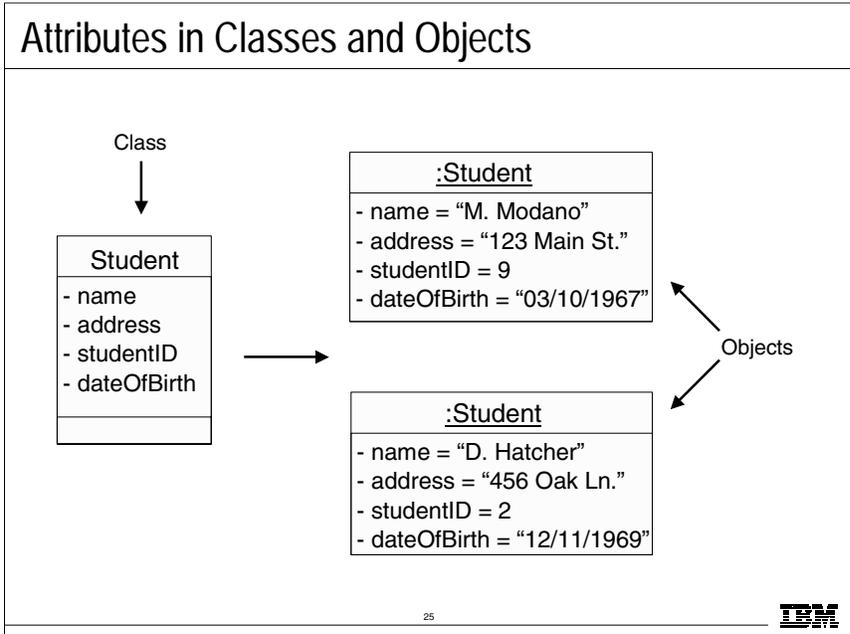
A named property of a class that describes the range of values that instances of the property may hold. (*The Unified Modeling Language User Guide*, Booch, 1999.)

- A class may have any number of attributes or no attributes at all. At any time, an object of a class has specific values for every one of its class's attributes.
- An attribute defined by a class represents a named property of the class or its objects. An attribute defines the type of its instances.
- An attribute has a **type**, which tells us what kind of attribute it is. Typical attributes are integer, Boolean, real, and enumeration. These are called **primitive** types. An attribute does not need to be a primitive type though.

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

## Instructor Notes:

Show that an attribute is defined on a class and instantiated on an object.



At the class level, the Student class attributes indicate that the Students have names, addresses, studentIDs, and a date of birth.

At the object level, the attributes indicate the values for the name, address, studentID, and date of birth.

Only the class instance (objects) should be able to change the value of the attributes.

The state of an object is defined by the value of its attributes and the existence of links to other objects.

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

## Instructor Notes:

Define the term operation for the students.

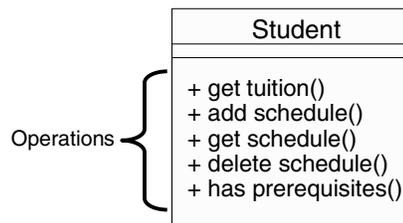
Point out that these should be called operations. Many people use the term *methods* instead of operations.

In the UML, methods and operations are NOT synonymous and have distinct definitions.

An operation is simply the advertisement of a service that is offered by a class. A method is the actual code that realizes that operation.

## What Is an Operation?

- ◆ A service that can be requested from an object to effect behavior. An operation has a signature, which may restrict the actual parameters that are possible.
- ◆ A class may have any number of operations or none at all.



26



An **Operation** can be defined as:

A service that can be requested from an object to effect behavior. An operation has a signature, which may restrict the actual parameters that are possible.

- The operations in a class describe what the class can do.
- An operation can either be a command or a question. A question should never change the state of the object. Only a command can change the state of the object.
- An operation is described with a return-type, name, and zero or more parameters. Together, the return-type, name, and parameters are called the **signature** of the operation.
- The outcome of the operation depends on the current state of the object. Often, but not always, invoking an operation on an object changes the object's data or state.

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

---

Instructor Notes:

## Where Are We?

- ◆ What is an object?
- ◆ Four principles of OO
- ◆ What is a class?
- ☆◆ Polymorphism and generalization
- ◆ Organizing model elements



27



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

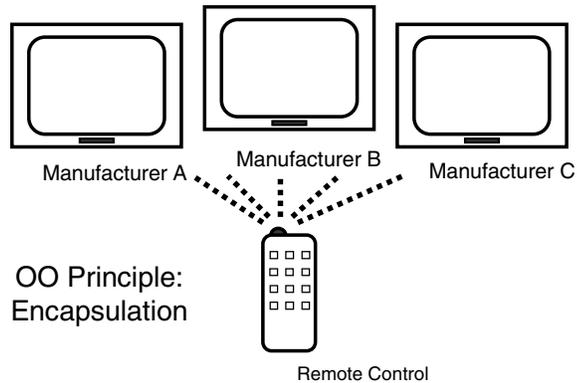
## Instructor Notes:

Define the term polymorphism for the students.

In a nutshell, polymorphism means that different things respond in different ways to the same command. For example, if you told your class to relax each one of the students would do something different, but they would still relax.

## What Is Polymorphism?

- ◆ The ability to hide many different implementations behind a single interface.



OO Principle:  
Encapsulation

28



The Greek term *polymorphos* means “having many forms.” Every implementation of the interface must include at least the interface. In some cases, the implementation can include more than the interface.

For example, a remote control can be used to monitor/support any type of television that relates to a specific interface (the specific interface the remote was designed to be used with).

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

## Instructor Notes:

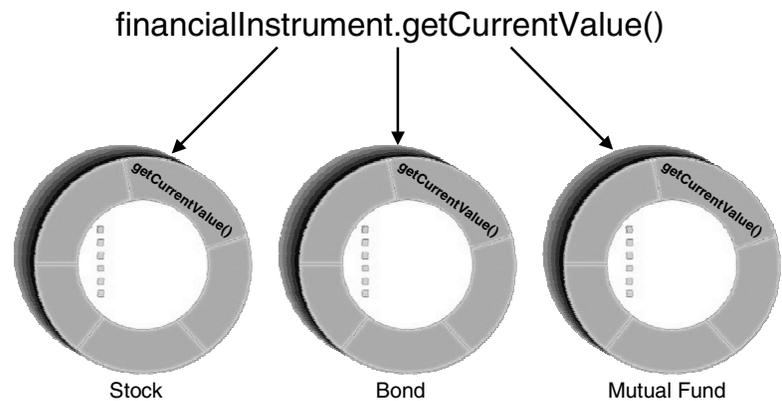
This diagram shows how three different objects can have the same operation: `getCurrentValue()`.

However, the way each interprets that operation is unique because the current value for each instrument is dependent on different variables.

Point out how the requesting object does not need to know anything about the calculations or the differences. All that it cares about is that the current value is calculated.

One of the huge benefits of polymorphism is that the calling object does not need to know what type of object it is calling. In this example, it doesn't care what type of asset it gets the value from, just that it gets the value.

## Example: Polymorphism



In this example, a requesting object would like to know the current value of a financial instrument. However, the current value for each financial instrument is calculated in a different fashion. The stock needs to determine the current asking price in the financial market that it is listed under. The bond needs to determine the maturity timelines and interest rates. A mutual fund needs to check the day's closing price from the fund management company.

In a non object-oriented development environment, you would write code that may look something like this:

```
IF financialInstrument = Stock THEN
    calcStockValue ()
ELSEIF financialInstrument = Bond THEN
    calcBondValue ()
ELSEIF financialInstrument = MutualFund THEN
    calcMutualFundValue ()
```

With object technology, each financial instrument can be represented by a class, and each class would know how to calculate its own value. The requesting object simply needs to ask the specific object (for example, `Stock`) to get its current value. The requesting object does not need to keep track of three different operation signatures. It only needs to know one. Polymorphism allows the same message to be handled in different ways depending on the object that receives it.

## Instructor Notes:

Introduce the concept of generalization. Remember, many of your students are not familiar with this term.

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

According to Grady Booch:

The terms “inheritance” and “generalization” are, practically speaking, interchangeable. The UML standard calls a 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).

## 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 in which a subclass inherits from one or more superclasses.
  - Single inheritance.
  - Multiple inheritance.
- ◆ Is an “is a kind of” relationship.

30



### 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 generalized 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.)

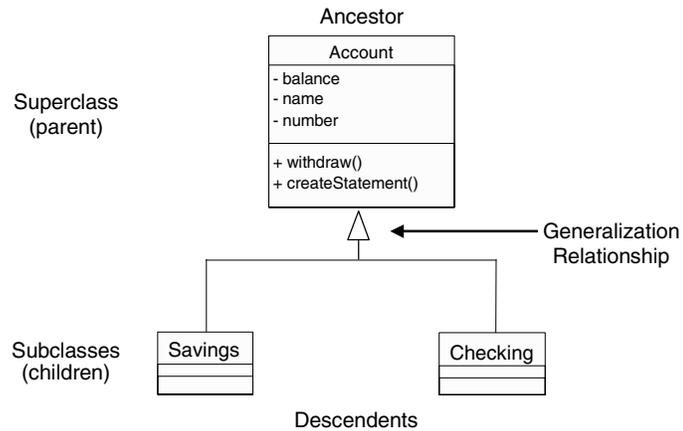
- Single inheritance: The subclass inherits from only one superclass (has only one parent).
- Multiple inheritance: The subclass inherits from more than one superclass (has multiple parents).

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

Instructor Notes:

## 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

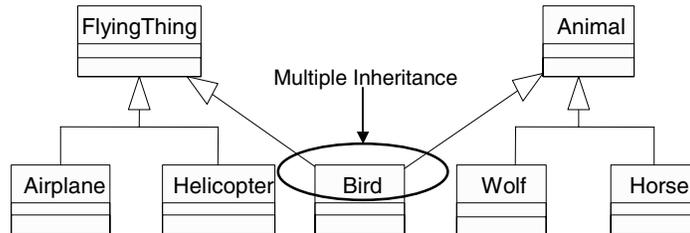
## Instructor Notes:

Provide an example of generalization.

Some languages do not support multiple inheritance.

## Example: Multiple Inheritance

- ◆ A class can inherit from several other classes.



Use multiple inheritance only when needed and always with caution!

32



Multiple inheritance means that a class can inherit 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, as not all implementation languages support it. Thus, be judicious with your use of multiple inheritance. Use it only where it accurately describes the concept 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:

Explain how generalization works.

Emphasize that when a change is made to a super class, all descendent classes inherit the change.

## What Is Inherited?

- ◆ A subclass inherits its parent's attributes, operations, and relationships.
- ◆ A subclass may:
  - Add additional attributes, operations, relationships.
  - Redefine inherited operations. (Use caution!)
- ◆ Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy.

Inheritance leverages the similarities among classes.

33



Generalization is more than finding common attributes, operations, and relationships. It is more about the responsibilities and essence of the classes.

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

---

Instructor Notes:

## Where Are We?

- ◆ What is an object?
- ◆ Four principles of OO
- ◆ What is a class?
- ◆ Polymorphism and generalization
- ☆ ◆ Organizing model elements



34



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

## Instructor Notes:

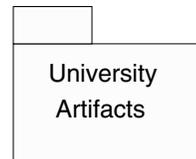
Introduce the term package to the students.

A package is a "virtual bag." You can place just about anything inside of it that you would like.

Packages allow us to organize our models into bits and pieces that make sense. They support the concept of modularity.

## 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.



35



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 contain hundreds and even thousands of model elements. The sheer number of these elements can quickly become overwhelming. Therefore, it's critical to group model elements into logical collections to maintain and easily read the model (application of modularity and hierarchy).
- Packages are a general grouping 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 maybe 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:

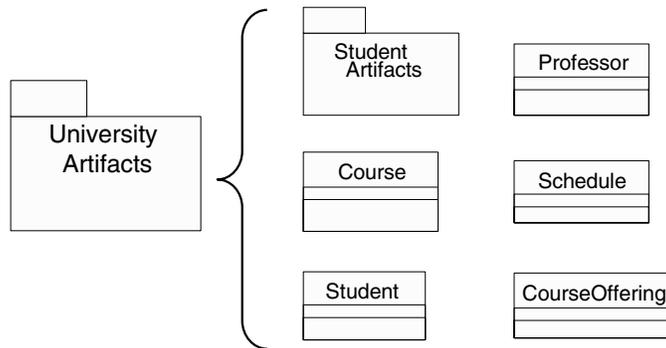
Explain how packages and classes are related to one another.

Make sure students understand that packages are a higher level of abstraction than a class.

Everything in your model must reside in a package. Therefore, it is assumed that, at the top level of your model, there is one owning package.

## A Package Can Contain Classes

- ◆ The package, University Artifacts, contains one package and five classes.



36



A package owns its elements and can even own other packages.

Owning is a **composite** relationship, meaning that the element is declared in the package. If the package is destroyed, the element is destroyed.

Every element is uniquely owned by exactly one package. For example, the package UniversityArtifacts owns the following classes: Course, Student, Schedule, Professor, and Course Offering. If the UniversityArtifacts package is destroyed then all of these classes are also destroyed. If you move the package to a different location in your model (architecturally speaking), then the classes move, too.

A package is an important mechanism for dealing with scale. Without packages, you would end up with large, flat models where all elements would be uniquely named.

Packages help you control the elements that compose your system as they evolve at different rates over time.

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

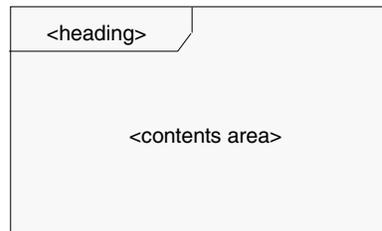
## Instructor Notes:

The UML 2 specification does not have consistent usage of `<kind>`, for example, the interactions chapter uses `sd` for all kinds of interaction diagrams (including timing), which is clearly erroneous.

The frame is primarily used in cases where the diagrammed element has graphical border elements, like ports for classes and components (in connection with composite structures), entry/exit points on state machines, and gates/fragments on sequence diagrams. The majority of these are not discussed in this course but will be in OOAD.

## Diagram Depiction

- ◆ Each diagram has a frame, a heading compartment in the upper left corner, and a contents area.
  - If the frame provides no additional value, it may be omitted and the border of the diagram area provided by the tool will be the implied frame.



37



A heading compartment is a string contained in a name tag (a rectangle with cutoff corner) in the upper leftmost corner with the following syntax:

`[<kind>]<name>[<parameters>]`

This `<kind>` can be:

- activity - activity diagram
- package - class diagram, package diagram
- communication - communication diagram
- component - component diagram
- class - composite structure diagram
- deployment - deployment diagram
- intover - interaction overview diagram
- object - object diagram
- state machine - state machine diagram
- sd - sequence diagram
- timing - timing diagram
- use case - use case diagram

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

## Instructor Notes:

A. An **object** is an entity with a well-defined boundary and identity that encapsulates state and behavior.

B. There are four basic principles of object orientation: abstraction, encapsulation, modularity and hierarchy.

C. A **class** is a description of a set of objects that share the same attributes, operations, relationships, and semantics. The class is the static description and the object is the run-time instance of that class.

D. An **attribute** is a named property of a class that describes the range of values that instances of the property may hold. An **operation** is the implementation of a service that can be requested from any object of the class to affect behavior.

E. **Polymorphism** is the ability to hide many different implementations behind a single interface. For example, a remote control can be used to monitor/support any type of television that relates to a specific interface (whichever interface the remote was designed to be used with).

F. **Generalization** is a relationship among classes where one class shares the structure and/or behavior of one or more classes.

G. Without **packages**, you would end up with large, flat models where all elements would be uniquely named.

Packages help you control the elements that compose your system as they evolve at different rates over time.

## Review

- ◆ What is an object?
- ◆ What are the four principles of object orientation? Describe each.
- ◆ What is a class? How are classes and objects related?
- ◆ What is an attribute? An operation?
- ◆ Define polymorphism. Provide an example of polymorphism.
- ◆ What is generalization?
- ◆ Why use packages?



38

IBM

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

---

## Instructor Notes:

- Reinforce the basic concepts of OO for the student.
- The purpose of this exercise is to ensure that the students understand all key concepts that have been discussed in the first three chapters.
- After a student answers a question correctly, take their number out of the hat. This ensures that everyone participates.
- Bring candy, coffee mugs, or some other "prize" to hand out.
- Stay with the game until all the questions are correctly answered.
- The key to education is repetition, repetition, repetition. By virtue of the game's design, some questions in the game may come up three to four times. If a question on polymorphism comes up that many times, you can be sure that your students will know polymorphism by the end of the game!

## Exercise: Principles of Object Orientation

- ◆ The "OO Quiz Show" Rules
  - Everyone in the class is assigned a number.
  - The instructor displays a question.
  - The instructor calls out a number.
  - If the student answers the question correctly, the class continues to the next question.
  - If the student does not answer the question correctly, the class goes back to the beginning.
  - The game is over when all questions have been answered correctly.

39



