

► ► ► **Module 2**
Concepts of Object Orientation



**Mastering Object-Oriented Analysis
and Design with UML**
Module 2: Concepts of Object Orientation

Topics

What Is Object Technology?	2-4
Basic Principles of Object Orientation.....	2-15
Representing Classes in the UML	2-24
Review.....	2-51

Objectives: Concepts of Object Orientation

Objectives: Concepts of Object Orientation

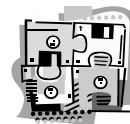
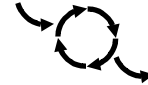
- ♦ Explain the basic principles of object orientation
- ♦ Define the basic concepts and terms of object orientation and the associated UML notation
- ♦ Demonstrate the strengths of object orientation
- ♦ Present some basic UML modeling notation

Best Practices Implementation

Best Practices Implementation

- ◆ Object technology helps implement these Best Practices.

- Develop Iteratively: tolerates changing requirements, integrates elements progressively, facilitates reuse.
- Use Component-Based Architectures: architectural emphasis, component-based development.
- Model Visually: easy understanding, easy modification.



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

3

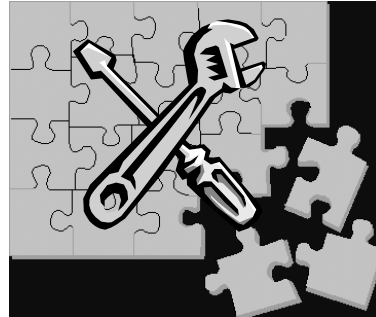
Rational
the software development company

- Iterative development allows you to account for technological changes. If a technology changes, becomes a standard, or new technology appears, the project can take advantage of it. This is particularly true for platform changes or lower-level infrastructure changes.
- Integration is not one "big bang" at the end. Elements are integrated progressively. Actually, the iterative approach is almost continuous integration. Noniterative integration takes up to 40% of the total effort at the end of a project and is hard to plan accurately. What used to be a long, uncertain, and difficult process is broken down into six-to-nine smaller integrations that start with far fewer elements to integrate.
- By clearly articulating the major components and the critical interfaces between them, an architecture allows you to plan for reuse, both internally, (the identification of common parts) and externally (the incorporation of off-the-shelf components). On a larger scale, it also allows the reuse of the architecture itself in the context of a line of products that addresses different functionality in a common domain.
- An object-oriented model aims at reflecting the world we experience in reality. Thus, the objects themselves often correspond to phenomena in the real world that the system is to handle. For example, an object can be an invoice in a business system or an employee in a payroll system.
- A model, correctly designed using object technology, is easy to understand and modify. It clearly corresponds to reality, and changes in a particular phenomenon concern only the object that represents that phenomenon.

What Is Object Technology?

What Is Object Technology?

- ♦ Object technology
 - A set of principles guiding software construction together with languages, databases, and other tools that support those principles. (*Object Technology: A Manager's Guide*, Taylor, 1997)



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

4

Rational
the software development company

- Object technology is used for creating models that reflect a specific domain using the terminology of the domain.
- Models created using object technology are easy to create, change, expand, validate, and verify.
- Systems built using object technology are flexible to change, have well-defined architectures, and allow reusable components to be created and implemented.
- Models created using object technology are conveniently implemented in software using object-oriented programming languages.
- Object technology is not just a theory, but a well-proven technology used in a large number of projects and for building many types of systems.
- Successful implementation of object technology requires a method that integrates a development process and a modeling language with suitable construction techniques and tools. (*UML Toolkit*, Eriksson and Penker, 1997)

Strengths of Object Technology

Strengths of Object Technology

- ♦ Provides a single paradigm
 - A single language used by users, analysts, designers, and implementers
- ♦ Facilitates architectural and code reuse
- ♦ Models more closely reflect the real world
 - More accurately describes corporate entities
 - Decomposed based on natural partitioning
 - Easier to understand and maintain
- ♦ Provides stability
 - A small change in requirements does not mean massive changes in the system under development
- ♦ Is adaptive to change

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

5

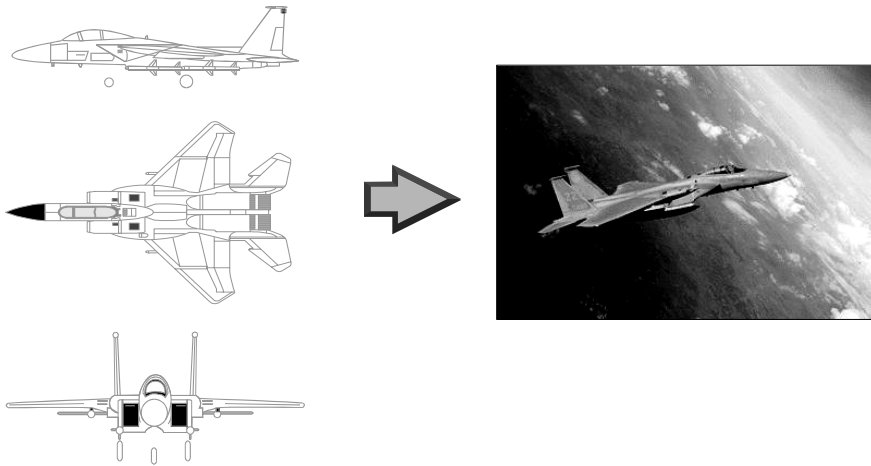
Rational
the software development company

- The UML represents a convergence of Best Practices throughout the object-technology industry. It provides a consistent language that can be applied to both system and business engineering.
- Object technology can help a company change its systems almost as fast as the company itself changes.

What Is a Model?

What Is a Model?

- ♦ A model is a simplification of reality.



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

6

Rational
the software development company

According to Grady Booch, a model provides the blueprints of a system. It can encompass detailed plans, as well as more general plans that give a 30,000-foot view of the system under construction. A good model includes those elements that are not relevant to the given level of abstraction. Every system can be described from different aspects using different models, and each model is therefore a semantically closed abstraction of the system. A model can be structural, emphasizing the organization of the system, or it can be behavioral, emphasizing the dynamics of the system.

Why Do We Model?

Why Do We Model?

- ♦ We build models to better understand the system we are developing.
- ♦ Modeling achieves four aims. It:
 - Helps us to visualize a system as we want it to be.
 - Permits us to specify the structure or behavior of a system.
 - Gives us a template that guides us in constructing a system.
 - Documents the decisions we have made.
- ♦ We build models of complex systems because we cannot comprehend such a system in its entirety.

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

7

Rational
the software development company

According to *The Unified Modeling Language Use Guide* (Booch, Rumbaugh, and Jacobson, 1998), modeling achieves four aims:

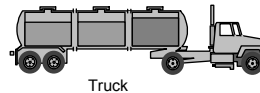
- Models help us to **visualize** a system as we want it to be. A model helps the software team communicate the vision for the system being developed. It is very difficult for a software team to have a unified vision of a system that is only described in specification and requirement documents. Models bring about understanding of the system.
- Models permit us to **specify** the structure or behavior of a system. A model allows us to document system behavior and structure before coding the system.
- Models give us a template that guides us in **constructing** a system. A model is an invaluable tool during construction. It serves as a road map for a developer. Have you experienced a situation where a developer coded incorrect behavior because there was confusion over the wording in a requirements document? Modeling helps alleviate that situation.
- Models **document** the decisions that have been made. Models are valuable tools in the long term because they give “hard” information on design decisions. You don’t need to rely on someone’s memory.

What Is an Object?

What Is an Object?

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

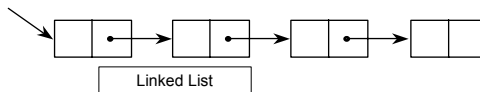
- Physical entity



- Conceptual entity



- Software entity



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

8

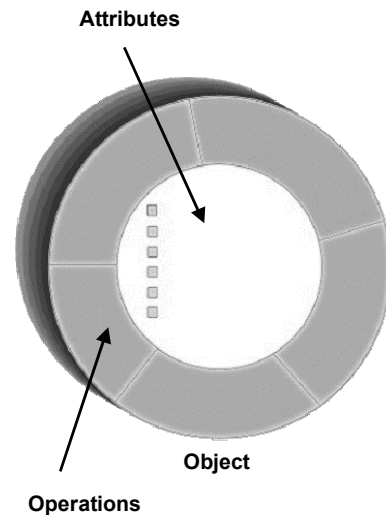
Rational
the software development company

- Objects allow the software developer to represent real-world concepts in the 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.
- Objects can even represent software entities like a linked list.

A More Formal Definition

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.



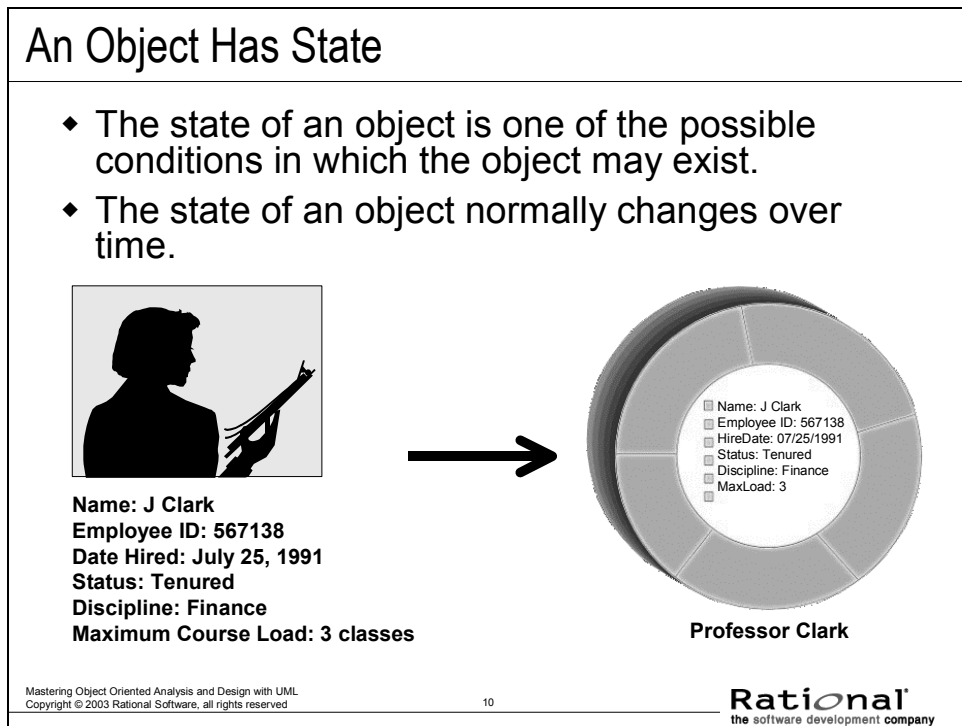
Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

9

Rational
the software development company

- 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 represent an object's state, and operations represent the behavior of the object.
- Object state and behavior are discussed on the next few slides.

An Object Has State



- The **state** of an object is one of the possible conditions that an object may exist in, and it 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.

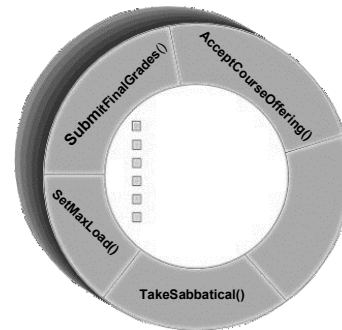
An Object Has Behavior

An Object Has Behavior

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



Professor Clark's behavior
 Submit Final Grades
 Accept Course Offering
 Take Sabbatical
 Maximum Course Load: 3 classes



Professor Clark

Mastering Object Oriented Analysis and Design with UML
 Copyright © 2003 Rational Software, all rights reserved

11

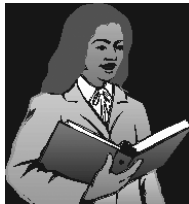
Rational
the software development company

- 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.

An Object Has Identity

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**

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

12

Rational
the software development company

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 their own unique **identities**.

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.

Representing Objects in the UML

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

Unnamed Object

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

13

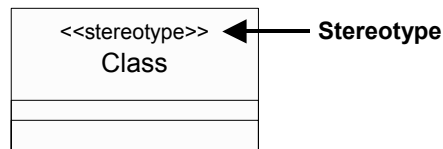
Rational
the software development company

- An object is represented as a rectangle.
- Within the rectangle, underlined, is the name of the class. To name an object, add its name before the colon.
- To keep an object unnamed (anonymous), do not include a name.

What Are Stereotypes?

What Are Stereotypes?

- ♦ Stereotypes define a new model element in terms of another model element.
- ♦ Sometimes you need to introduce new things that speak the language of your domain and look like primitive building blocks.



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

14

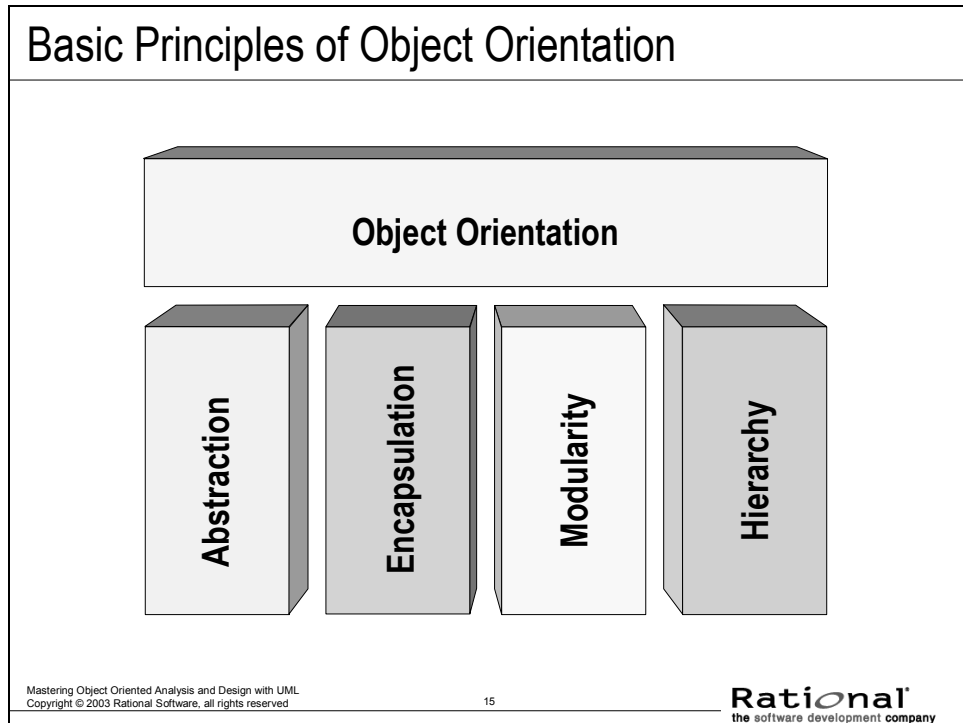
Rational
the software development company

A **stereotype** can be defined as:

An extension of the basic UML notation that allows you to define a new modeling element based on an existing modeling element.

- The new element may contain additional semantics but still applies in all cases where the original element is used. In this way, the number of unique UML symbols is reduced, simplifying the overall notation.
- The name of a stereotype is shown in guillemets (<< >>).
- A unique icon may be defined for the stereotype, and the new element may be modeled using the defined icon or the original icon with the stereotype name displayed, or both.
- Stereotypes can be applied to all modeling elements, including classes, relationships, components, and so on.
- Each UML element can only have one stereotype.
- Stereotype uses include modifying code generation behavior and using a different or domain-specific icon or color where an extension is needed or helpful to make a model more clear or useful.

Basic Principles of Object Orientation



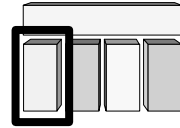
There are four basic principles of object orientation. They are:

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

What Is Abstraction?

What Is Abstraction?

- ♦ The essential characteristics of an entity that distinguish 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



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

16

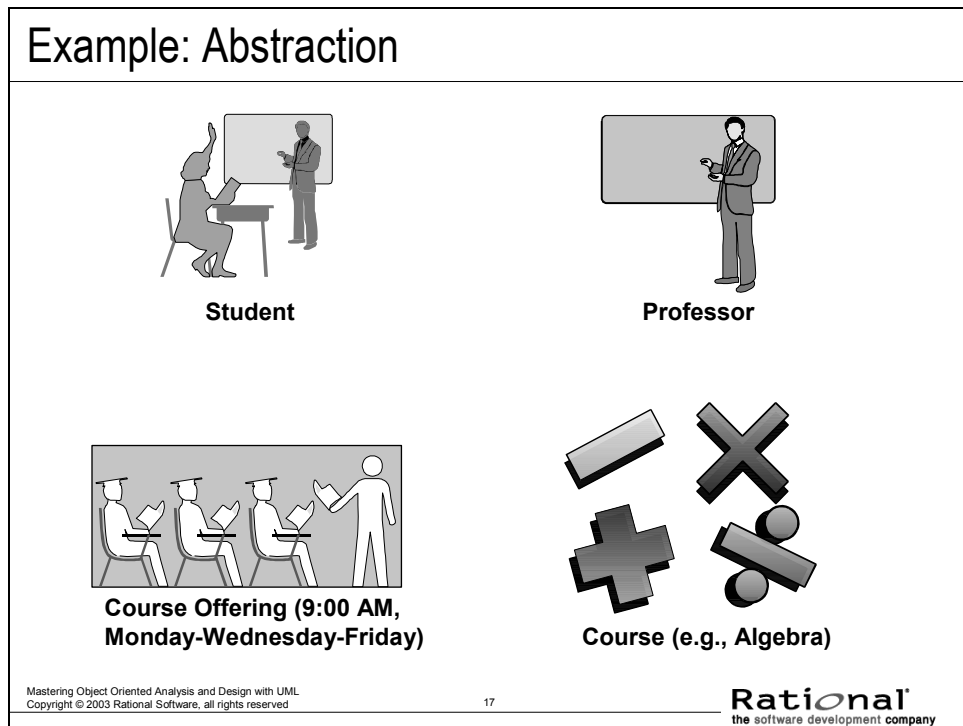
Rational
the software development company

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. It is 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 distinguish 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).

Example: Abstraction



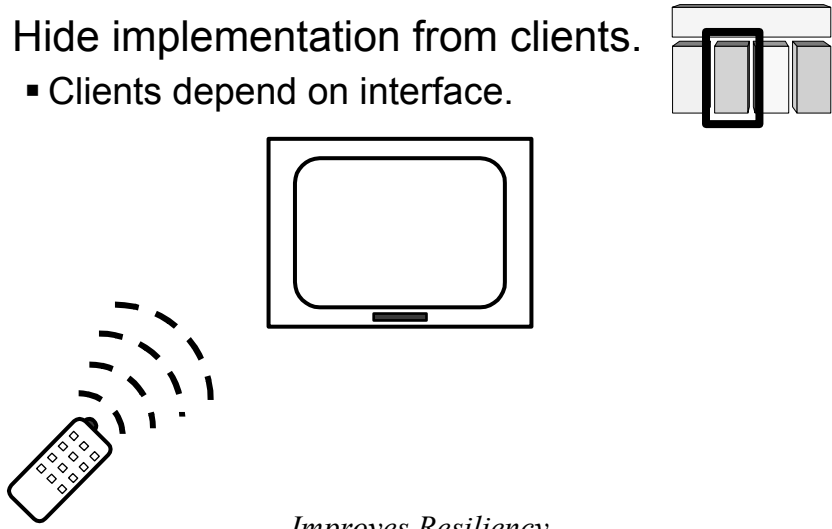
The following are examples of abstraction.

- A student is a person enrolled in classes at 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 days of the week and times.

What Is Encapsulation?

What Is Encapsulation?

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



Improves Resiliency

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

18

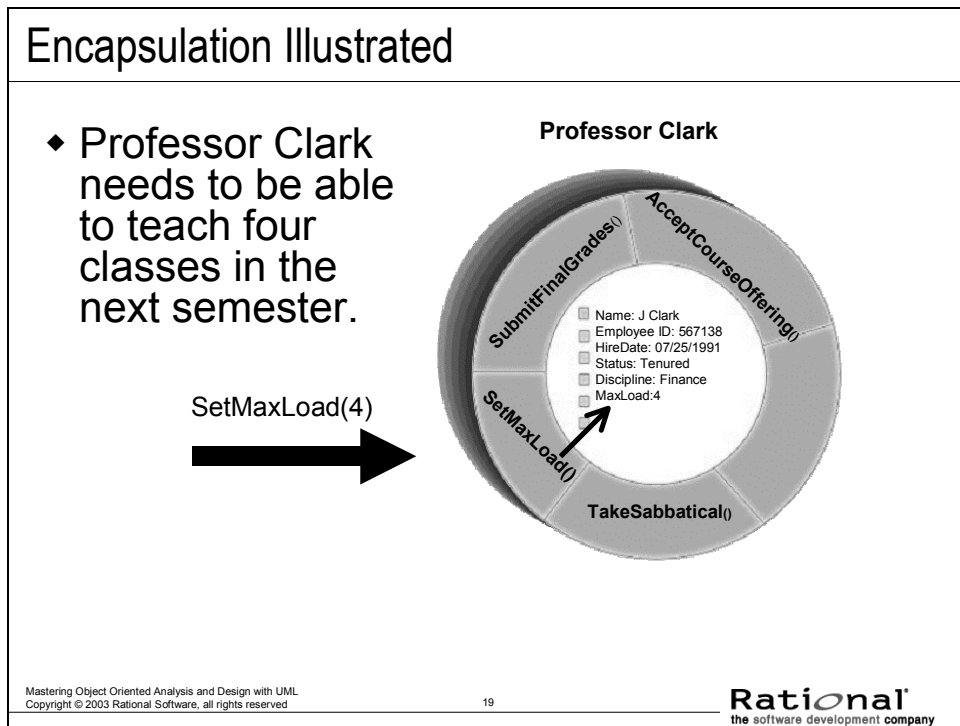
Rational
the software development company

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 fulfills the interface.
- Encapsulation eliminates direct dependencies on the implementation (clients depend on/use the interface). Thus, it is possible to change the implementation without updating the clients as long as the interface is unchanged.
- Clients will not be affected by changes in implementation. This reduces the “ripple effect,” which happens when 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.

Encapsulation Illustrated

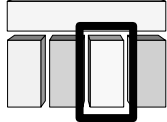
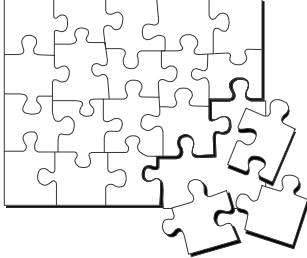


- The key to encapsulation is an object's **message interface**. The object interface ensures that all communication with the object takes place through a set of predefined operations. Data inside the object is only accessible by the object's operations. No other object can reach inside of the object 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 will make 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 of variables that are used to define the maximum course load may be increased, but that does not affect the requesting object. The requesting object depends on the operation interface for the Professor Clark object.

What Is Modularity?

What Is Modularity?

- ♦ Modularity is the breaking up of something complex into manageable pieces.
- ♦ Modularity helps people to understand complex systems.



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

20

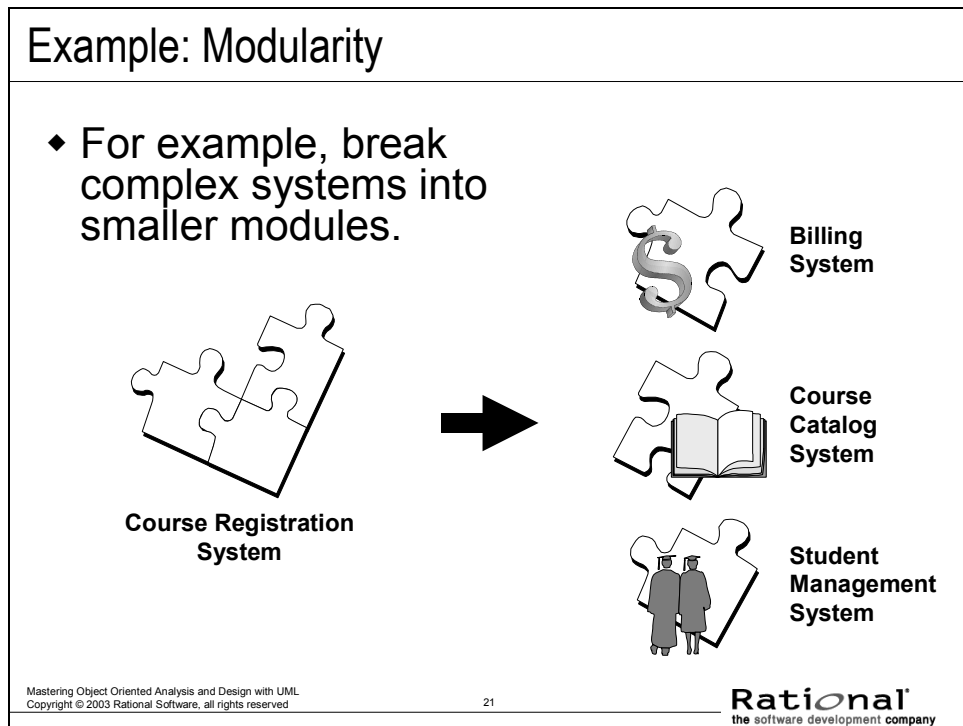
Rational
the software development company

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.

Example: Modularity

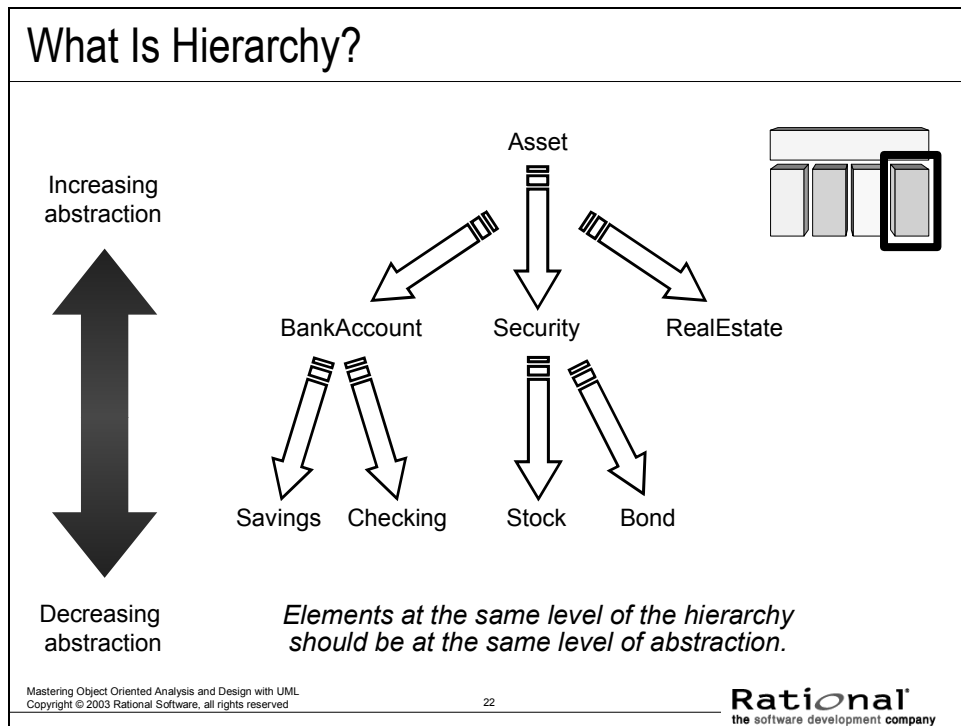


Often, the system under development is too complex to understand. To further understanding, the system is broken into smaller blocks that are each maintained independently. Breaking down a system in this way is called **modularity**. It is critical for understanding a complex system.

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

- The Billing System
- Course Catalog System
- Student Management System

What Is Hierarchy?



Hierarchy can be defined as:

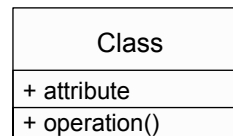
Any ranking or ordering of abstractions into a tree-like structure. Kinds: Aggregation hierarchy, class hierarchy, containment hierarchy, inheritance hierarchy, partition hierarchy, specialization hierarchy, type hierarchy. (*Dictionary of Object Technology*, Firesmith, Eykholt, 1995)

- Hierarchy organizes items in a particular order or rank (for example, complexity and responsibility). 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 not an organizational chart or 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 in a periodic table.

What Is a Class?

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.



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

23

Rational
the software development company

A **class** can be defined as:

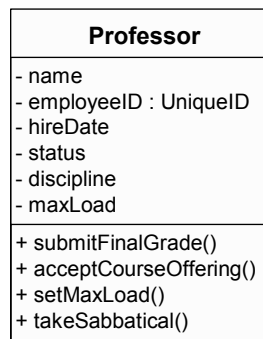
A description of a set of objects that share the same attributes, operations, relationships, and semantics. (*The Unified Modeling Language User Guide*, Booch, 1999)

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

Representing Classes in the UML

Representing Classes in the UML

- ♦ A class is represented using a rectangle with compartments.



Professor J Clark

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

24

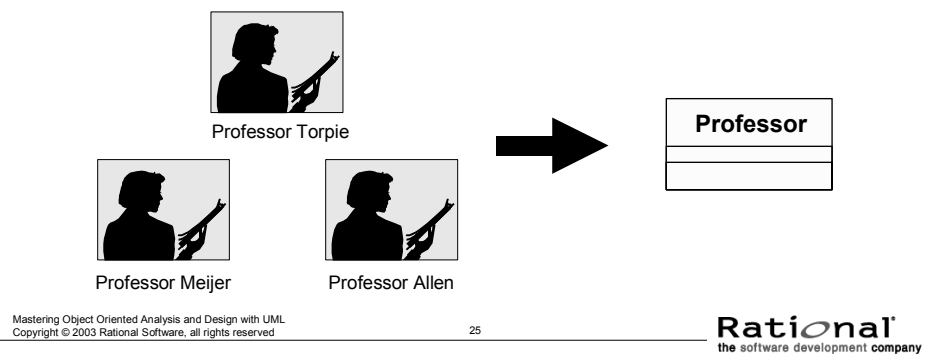
Rational
the software development company

- 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 Relationship Between Classes and Objects

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.

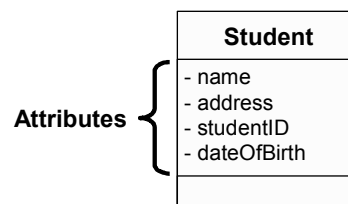


- A class is a description of a set of objects that share the same responsibilities, relationships, operations, attributes, and semantics.
- An object is defined by a class. A class defines a template for the structure and behavior of all its objects. The objects created from a class are also called the **instances** of the class.
- The class is the static description; the object is a run-time instance of that class.
- Since we model from real-world objects, software objects are based on the real-world objects, but they exist only in the context of the system.
- Starting with real-world objects, abstract out what you do not care about. Then, take these abstractions and categorize, or *classify* them, 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 we originally started with.
- Some classes/objects may be defined that do not represent real-world objects. They are there to support the design and are "software only."

What Is an Attribute?

What Is an Attribute?

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



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

26

Rational
the software development company

An **attribute** can be defined as:

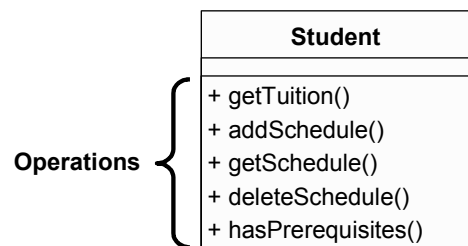
A named property of a class that describes the range of values that instances of the property may hold. (*The Unified Modeling Language User Guide*, Booch, 1999)

- A class may have any number of attributes or no attributes at all. At any time, an object of a class will have specific values for every one of its class' attributes.
- An attribute defined by a class represents a named property of the class or its objects. An attribute has a type that 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. Primitive types can be specific for a certain programming language.

What Is an Operation?

What Is an Operation?

- ♦ An operation is the implementation of a service that can be requested from any object of the class to affect behavior.
- ♦ A class may have any number of operations or none at all.



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

27

Rational
the software development company

An **operation** can be defined as:

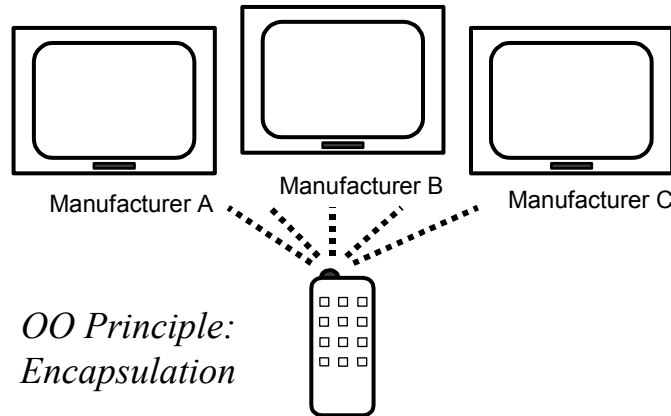
The implementation of a service that can be requested from any object of the class to affect behavior. (*The Unified Modeling Language User Guide*, Booch, 1999)

- The operations in a class describe what the class can do.
- An operation can be either a command or a question. Only a command can change the state of the object; a question should never change it.
- 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.

What Is Polymorphism?

What Is Polymorphism?

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



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

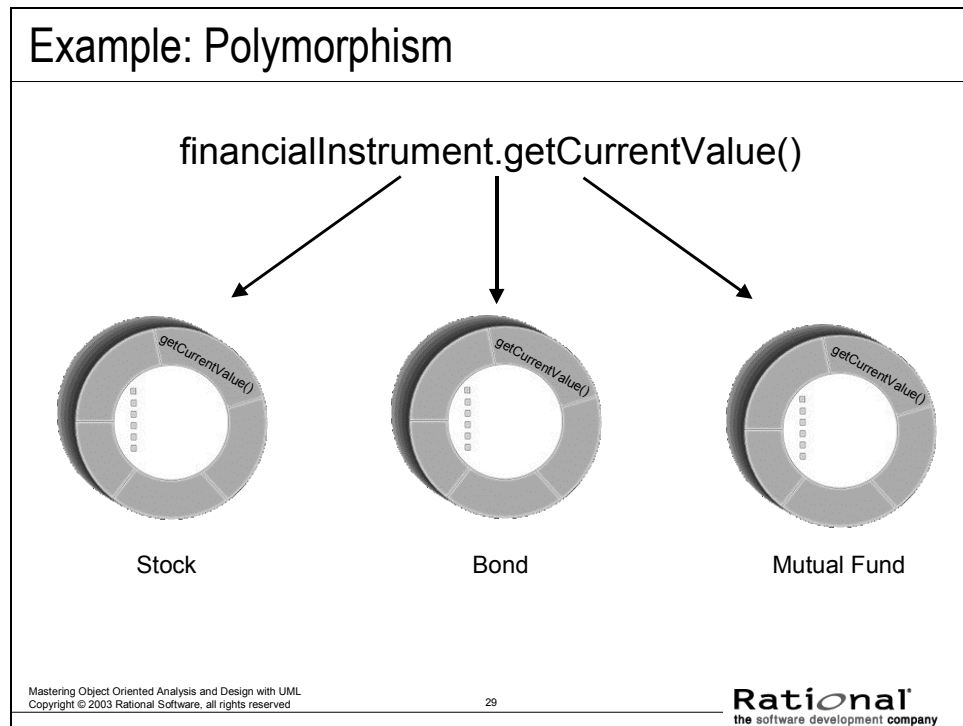
28

Rational
the software development company

The Greek term *polymorphos* means “having many forms.” There may be one or many implementations of a given interface. Every implementation of an interface must fulfill the requirements of that interface. In some cases, the implementation can perform more than the basic interface requirements.

For example, the same remote can be used to control any type of television (implementation) that supports the specific interface that the remote was designed to be used with.

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 time to maturity and interest rates. A mutual fund needs to look up the closing price for the day from the fund management company.

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

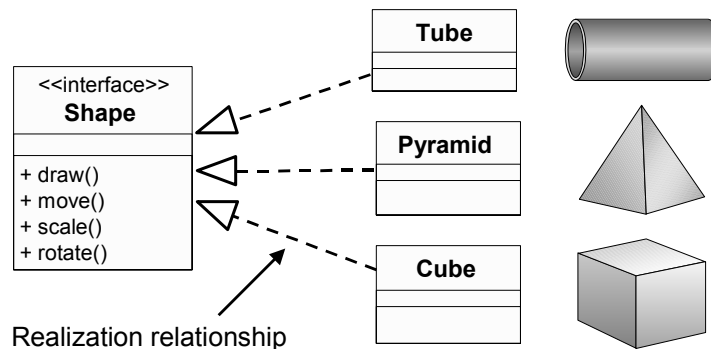
```
IF financialInstrument = Stock THEN
    calcStockValue()
IF financialInstrument = Bond THEN
    calcBondValue()
IF financialInstrument = MutualFund THEN
    calcMutualFundValue()
```

With object technology, each financial instrument can be represented by a class, and each class will 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.

What Is an Interface?

What Is an Interface?

- ♦ Interfaces formalize polymorphism
- ♦ Interfaces support “plug-and-play” architectures



(stay tuned for realization relationships)

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

30

Rational
the software development company

From the *UML User's Guide* (Booch, 1999): An interface is “a collection of operations that are used to specify a service of a class or a component.”

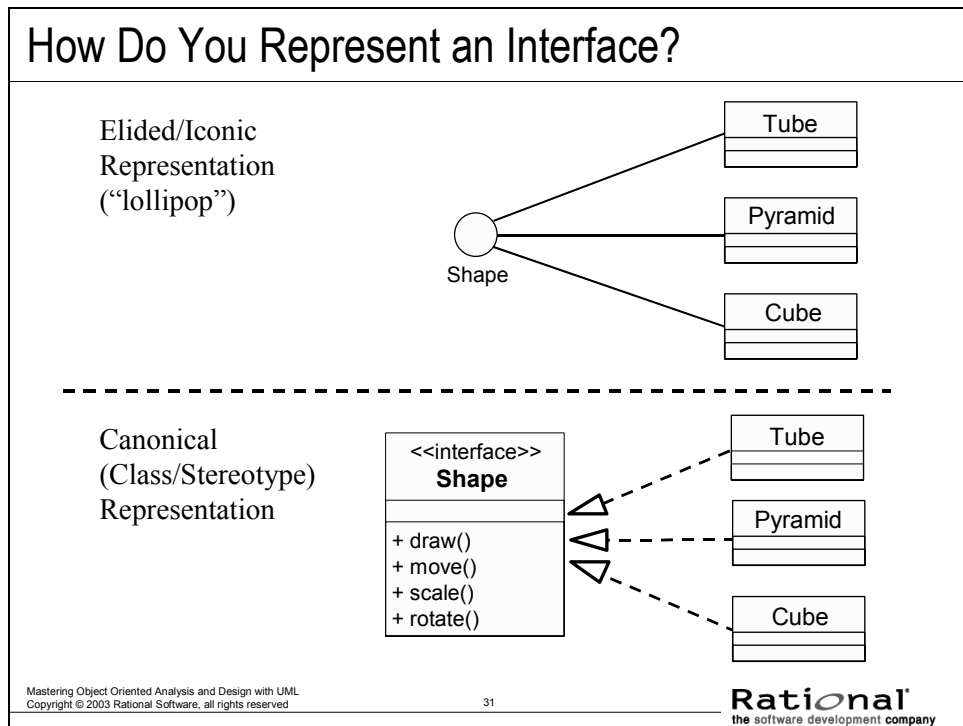
Interfaces formalize polymorphism. They allow us to define polymorphism in a declarative way, unrelated to implementation. Two elements are polymorphic with respect to a set of behaviors if they realize the same interfaces. In other words, if two objects use the same behaviors to get different, but similar results, they are considered to be polymorphic. A cube and a pyramid can both be drawn, moved, scaled, and rotated, but they look very different.

You have probably heard that polymorphism is one of the big benefits of object orientation, but without interfaces there is no way to enforce it, verify it, or even express it except in informal or language-specific ways. Formalization of interfaces strips away the mystery of polymorphism and gives us a good way to describe, in precise terms, what polymorphism is all about. Interfaces are testable, verifiable, and precise.

Interfaces are the key to the “plug-and-play” ability of an architecture: Any classifiers (for example, classes, subsystems, components) that realize the same interfaces may be substituted for one another in the system, thereby supporting the changing of implementations without affecting clients.

Realization relationships are discussed later in this module.

How Do You Represent an Interface?



The lollipop notation is best used when you only need to denote the existence of an interface. If you need to see the details of the interface (for example, the operations), then the class/stereotype representation is more appropriate.

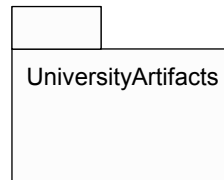
From *The Random House Collegiate Dictionary*:

- Elide: to pass over; omit; ignore.
- Canonical: authorized; recognized; accepted.

What Is a Package?

What Is a Package?

- ♦ A package is a general-purpose mechanism for organizing elements into groups.
- ♦ It is 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.

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

32

Rational
the software development company

A **package** can be defined as:

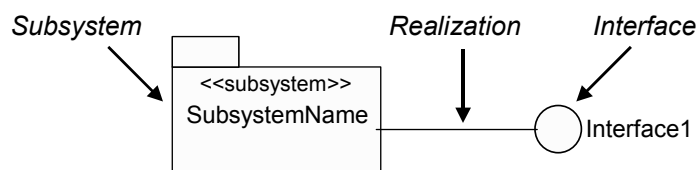
A general purpose mechanism for organizing elements into groups. (*The Unified Modeling Language User Guide*, Booch, 1999)

- Models can contain hundreds and even thousands of model elements. The sheer number of these elements can quickly become overwhelming. Therefore, it is 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.

What Is a Subsystem?

What Is a Subsystem?

- ♦ A combination of a package (can contain other model elements) and a class (has behavior)
- ♦ Realizes one or more interfaces which define its behavior



OO Principles: Encapsulation and Modularity

(stay tuned for realization relationships)

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

33

Rational
the software development company

A subsystem is a model element that has the semantics of a package, such that it can contain other model elements, and a class, such that it has behavior. (The behavior of the subsystem is provided by classes or other subsystems it contains.) A subsystem realizes one or more interfaces, which define the behavior it can perform.

From the *UML User's Guide* (Booch, 1999): A subsystem is “a grouping of elements of which some constitute a specification of the behavior offered by the other contained elements.”

A subsystem may be represented as a stereotyped package.

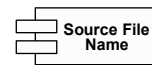
The realization relationship will be discussed later in this module.

What Is a Component?

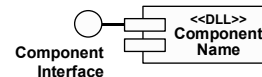
What Is a Component?

- ♦ A non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture
- ♦ A component may be

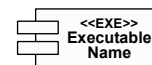
- A source code component



- A run time component



- An executable component



OO Principle: Encapsulation

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

34

Rational
the software development company

From the *UML User's Guide* (Booch, 1999): A **component** is “a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.”

Software components include:

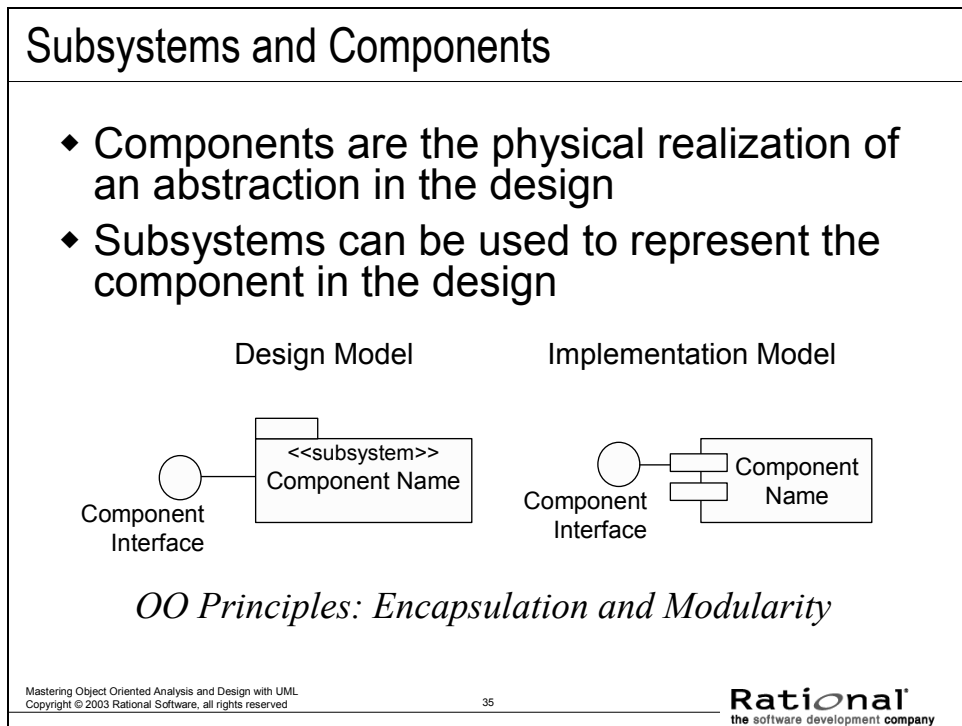
- Source code components (for example, .h, .cpp files, shell scripts, data files),
- Binary code components. Examples include: Java Beans, ActiveX controls, COM objects (DLL's and OCX's from VB), CORBA objects), and
- Executable components (.exe's).

Stereotypes (with alternate icons) may be used to define these specific kinds of components.

Component-based development is the creation and deployment of software-intensive systems assembled from components. Component software development should not be confused with object-oriented programming (OOP). OOP is a way to build object-based software components. Component-based development (CBD) is a technology that allows you to combine object-based components. OOP is concerned with creating objects, CBD is concerned with making objects work together.

A component conforms to and provides the physical realization of a set of interfaces. Components are implementation things. They are the physical realization of an abstraction in your design. Interfaces were discussed earlier in this module.

Subsystems and Components



A component conforms to and provides the physical realization of a set of interfaces. Components are implementation things. They are the physical realization of an abstraction in your design. A subsystem can be used to represent the component in the design.

A subsystem is the design representation of a component. They both encapsulate a set of replaceable behaviors behind one or more interfaces. Subsystems and interfaces will be discussed later in the course.

Components UML 1.4 and Beyond

Components UML 1.4 and Beyond

- ♦ UML 1.4 introduces concept of Artifacts:
 - An Artifact represents a physical piece of information that is used or produced by a software development process. Examples of Artifacts include models, source files, scripts, and binary executable files.
- ♦ To distinguish between artifacts in general, and the artifacts that make up the implementation, we introduce a new term:
 - Implementation Element - the physical parts (UML artifacts) that make up an implementation, including software code files (source, binary or executable), and data files.

Components UML 1.4 and Beyond (cont.)

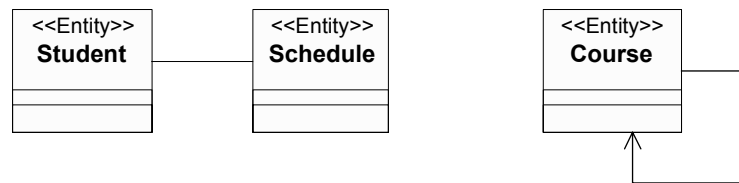
Components UML 1.4 and Beyond (cont.)

- ◆ Component becomes similar to subsystem:
 - can group classes to define a larger granularity units of a system
 - can separate the visible interfaces from internal implementation
 - can have instances that execute at run-time
- ◆ The distinction between "component" and "artifact" is new in UML 1.4.
 - Many tools, profiles, and examples continue to use "component" to represent implementation elements.

What Is an Association?

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



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

38

Rational
the software development company

An **association** can be defined as:

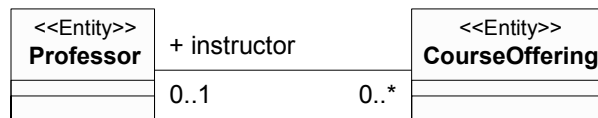
The semantic relationship between two or more classifiers that specifies connections among their instances. In other words, an association is a structural relationship that specifies that objects (instances of classes) are connected to other objects.

- The way that we show 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 in 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 course class demonstrates how a course object can be related to another course object.

What Is 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.



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

39

Rational
the software development company

Multiplicity can be defined as:

The number of instances of one class that relate 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 can have either zero or one Professor object related to it. Conversely, a Professor object can have zero or more Course Offering objects related to it.
- Multiplicity must be defined on both ends of the association.

Multiplicity Indicators

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

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

40

Rational
the software development company

- 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, followed by another 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.
- An optional scalar role has the multiplicity 0..1.

What Is Aggregation?

What Is Aggregation?

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



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

41

Rational
the software development company

An **aggregation** can be defined as:

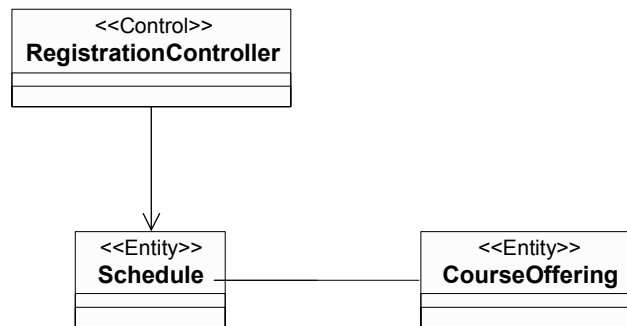
A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.

- Aggregation is used to model relationships between model elements. There are many examples of aggregation: 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 the 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 where there is a strong relationship between two classes. Therefore, the same instance can participate in two different aggregations.

What Is Navigability?

What Is Navigability?

- ♦ Indicates that it is possible to navigate from a associating class to the target class using the association



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

42

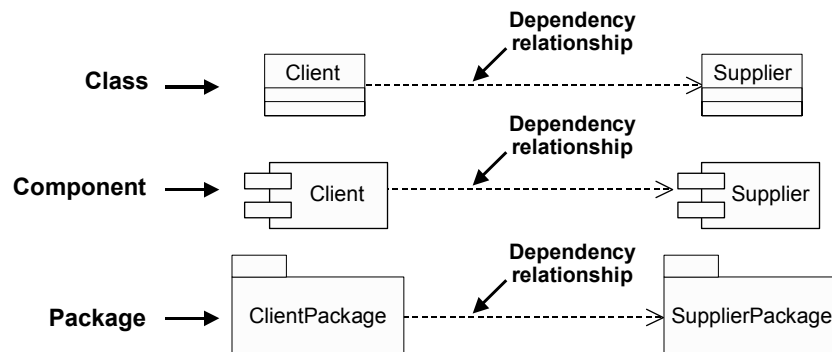
Rational
the software development company

- The **navigability** property on a role indicates that it is possible to navigate from a associating class to the target class using the association. This may be implemented in a number of ways: by direct object references, associative arrays, hash-tables, or any other implementation technique that allows one object to reference another.
- Navigability is indicated by an open arrow placed on the target end of the association line next to the target class (the one being navigated to). The default value of the navigability property is true (associations are bi-directional by default).
- In the course registration example, the association between the Schedule and the Course Offering is navigable in both directions. That is, a Schedule must know the Course Offering assigned to the Schedule, and the Course Offering must know the Schedules it has been placed in.
- When no arrowheads are shown, the association is assumed to be navigable in both directions.
- In the case of the association between Schedule and Registration Controller, the Registration Controller must know its Schedules, but the Schedules have no knowledge of the Registration Controllers (or other classes). As a result, the navigability property of the Registration Controller end of the association is turned off.

Relationships: Dependency

Relationships: Dependency

- ♦ A relationship between two model elements where a change in one may cause a change in the other
- ♦ Non-structural, “using” relationship



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

43

Rational
the software development company

A **dependency** relationship is a weaker form of relationship showing a relationship between a client and a supplier where the client does not have semantic knowledge of the supplier.

A dependency relationship denotes a semantic relationship between model elements, where a change in the supplier may cause a change in the client.

What Is Generalization?

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

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

44

Rational
the software development company

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)

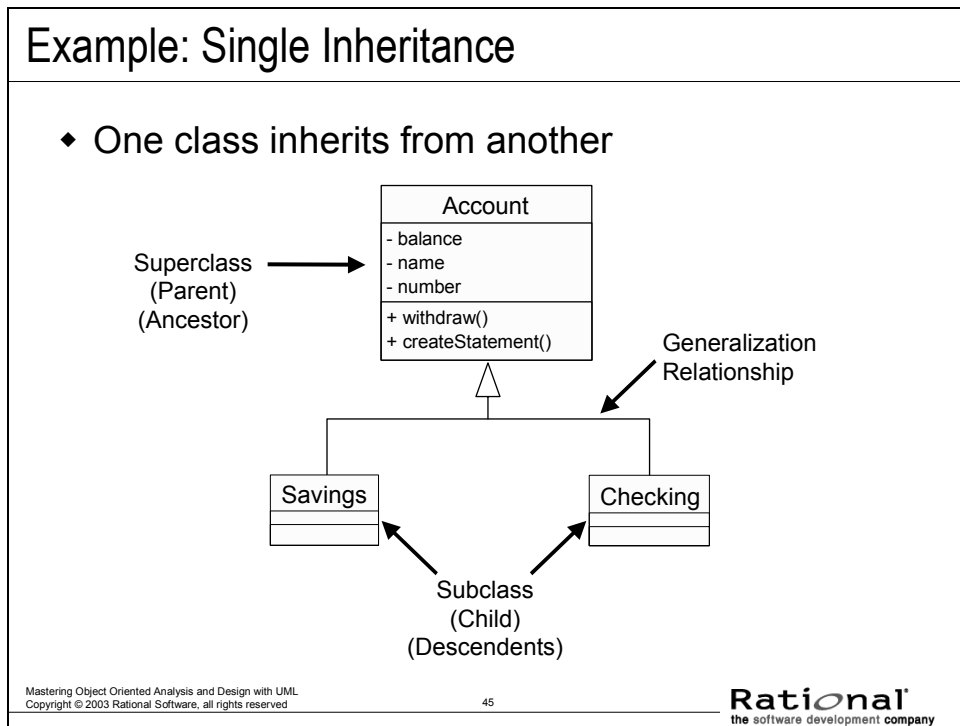
- The subclass may be used where the superclass is used, but not vice versa.
- The child inherits from the parent.
- Generalization is transitive. You can always test your generalization by applying the “is a kind of” rule. You should always be able to say that your specialized class “is a kind of” the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

Example: Single Inheritance

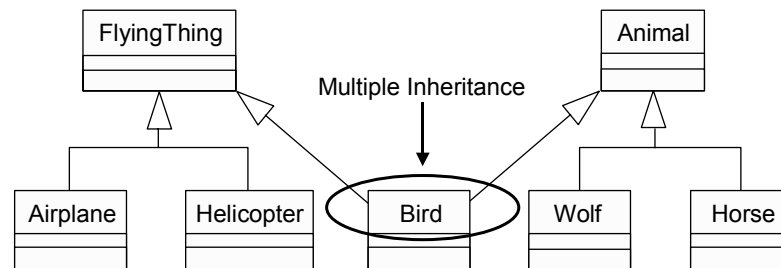


- The generalization is drawn from the subclass class to the superclass/parent class.
- The terms “ancestor” and “descendent” can be used instead of “superclass” and “subclass.”

Example: Multiple Inheritance

Example: Multiple Inheritance

- ♦ A class can inherit from several other classes.



Use multiple inheritance only when needed and always with caution!

Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

46

Rational
the software development company

- **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 will probably need to be adjusted in design and implementation.
- Generally, a class inherits from only one class.

What Gets Inherited?

What Gets 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

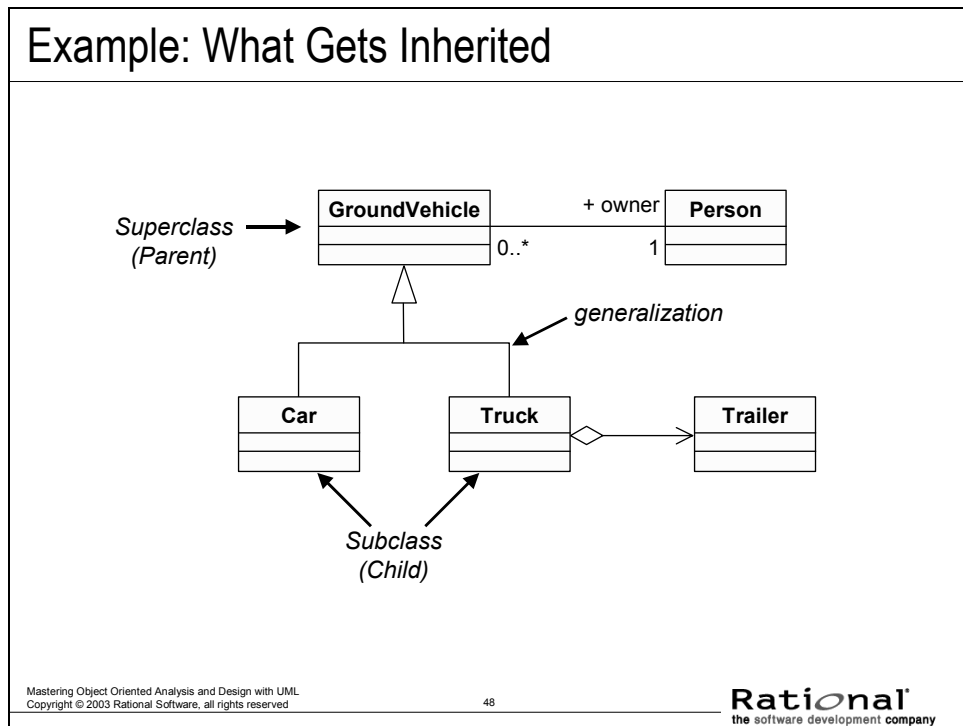
Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

47

Rational
the software development company

Generalization is much more than just finding common attribute, operations, and relationships. It should be more about the responsibilities and essence of the classes.

Example: What Gets Inherited



In the above example, a truck is defined as a vehicle with tonnage and a car is a vehicle with a size (for example, compact or mid-size).

Specifically, attributes:

- A GroundVehicle has two attributes: weight and licenseNumber
- A Car has three attributes: licenseNumber, weight, and size
- A Truck has three attributes: licenseNumber, weight, and tonnage

Operations:

- A GroundVehicle has one operation: register()
- A Car has one operation: register()
- A Truck has two operations: register() and getTax()

Relationships:

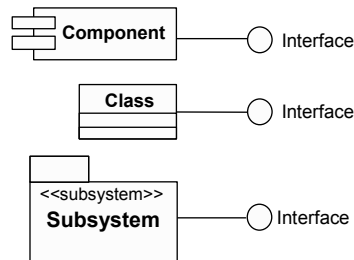
- A Car is related to a Person (the owner)
- A Truck is related to a Person (the owner)
- A Truck also has a Trailer and is related to a Person (the owner)

What Is Realization?

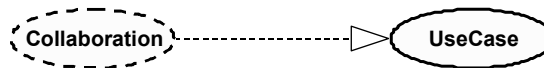
What Is Realization?

- ♦ One classifier serves as the contract that the other classifier agrees to carry out, found between:

- Interfaces and the classifiers that realize them



- Use cases and the collaborations that realize them



Mastering Object Oriented Analysis and Design with UML
Copyright © 2003 Rational Software, all rights reserved

49

Rational
the software development company

Realization is a semantic relationship between two classifiers. One classifier serves as the contract that the other classifier agrees to carry out.

The realizes relationship is a combination of a dependency and a generalization. It is not true generalization, as only the “contract” (that is to say, operation signature) is “inherited.” This “mix” is represented in its UML form, which is a combination of dependency and generalization.

The realizes relationship may be modeled as a dashed line with a hollow arrowhead pointing at the contract classifier (canonical form), or when combined with an interface, as a “lollipop” (elided form).

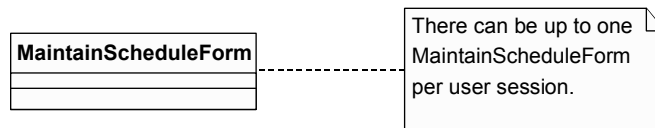
Again, from *The Random House Collegiate Dictionary*:

- Elide: to pass over; omit; ignore.
- Canonical: authorized; recognized; accepted.

What Are Notes?

What Are Notes?

- ♦ A comment that can be added to include more information on the diagram
- ♦ May be added to any UML element
- ♦ A “dog eared” rectangle
- ♦ May be anchored to an element with a dashed line



Review

Review: Concepts of Object Orientation

- ♦ What are the four basic principles of object orientation? Provide a brief description of each.
- ♦ What is an object and what is a class? What is the difference between the two?
- ♦ What is an attribute?
- ♦ What is an operation?
- ♦ What is polymorphism? What is an interface?



Review: Concepts of Object Orientation (cont.)

Review: Concepts of Object Orientation (cont.)

- ♦ What is a package?
- ♦ What is a subsystem? How does it relate to a package? How does it relate to a class?
- ♦ Name the four basic UML relationships and describe each.
- ♦ Describe the strengths of object orientation
- ♦ What are stereotypes?

