Metodologias de Desarrollo de Software I

Unified Modeling Language (UML)

Ivar Jacobson
Grady Booch
James Rumbaugh

Agenda - Structural Diagrams

- Class diagram
- Object diagram
- Component diagram
- Deployment diagram
- Composite Structure diagram
- Packages diagram

Class

ClassName

- **A class is a description of a set of objects that share the same:**
  - **attributes**
  - **operations**
  - **relationships**
  - **semantics**
- **A class can be used to represent software things, hardware things, and even things that are purely conceptual**

Class

ClassName

- **Graphically, a class is rendered as a rectangle**
- **A class must have a name that distinguishes it from other classes (*simple name*, *path name*)**
- **The first letter must be capitalize, as Customer or TemperatureSensor**

ClassName

Class Identification

ClassName

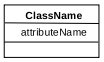
- **Identify candidate classes by picking out all of the noun and noun phrases from a requirement specification of the system.**
- **Discard candidates which are inappropriate for any reason, renaming the remaining classes if necessary.**

Removal of Class

ClassName

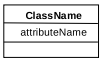
- **Redundant – The same class is given more than one name.**
- **Vague – You are unable to determine the meaning of a noun.**
- **An Event or an Operation – The noun refers to something which is done to, by or in the system.**
- **Meta-language – the noun forms part of the way we define things.**
- **Outside the Scope of the System – The noun is relevant to describing how the system works, but does not refer to something inside the system.**
- **An Attribute – A noun refers to something simple with no interesting behavior, which is an attribute of another class.**

Attributes

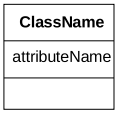


- Named property of a class that describes a range of values that instances of the property may hold
- A class may have any number of attributes or no attributes at all
- An attribute represents some property of the thing you are modeling that is shared by all objects of that class

Attributes



- Graphically, attributes are listed in a compartment just below the class name
- An attribute name is a short noun or noun phrase that represents some property
- The first letter of every word in an attribute name can be capitalized except the first letter, as in name or birthDate

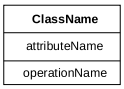


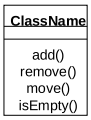
Operations

- An operation is the implementation of a service that can be requested from any object
- A class may have any number of operations or no operations at all

Operations

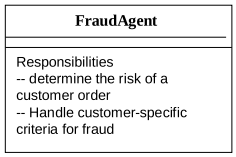
- Graphically, operations are listed in a compartment just below the class attributes
- An operation name is a short verb or verb phrase that represents some behavior
- The first letter of every word in an operation name can be capitalized except the first letter, as move or isEmpty
- An operation can be specified by stating its signature, covering the name, type, and default value of all parameters and a return type



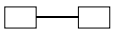


Responsibility

- A responsibility is a contract or an obligation of a class
- CRC like cards and use case-based analysis are helpful to find the responsibilities
- Graphically, responsibilities can be drawn in a separate compartment at the bottom of the class icon in a free-form text. They can be written as a phrase, sentence or a short paragraph



Relationships



- A relationship is a connection among things
- Sequence and/or collaboration diagrams are examined to determine what links between objects need to exist to accomplish the behavior -- if two objects need to "talk" there must be a link between them
- Three types of relationships are:
  - Association: structural relationship among objects
  - Generalization: links generalized classes to their specializations
  - Dependency: represents using relationships among classes (refinement, trace)

Relationships

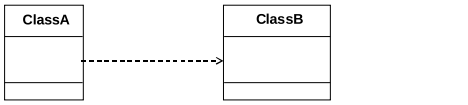
- Class A and class B are associated if:
  - An object of class A sends a message to an object of class B
  - An object of class A creates an object of class B
  - An object of class A has an attribute whose values are objects of class B or collections of objects of class B.
  - An object of class A receives a message with an object of class B as an argument

Relationships - Dependency

- A dependency is a using relationship that states that a change in specification of one thing may affect another thing that uses it, but not necessarily the reverse
- Dependency is used when a thing use another. To show that one class uses another class as an argument in the signature of an operation

Relationships - Dependency

- Graphically, a dependency is shown as a dashed line directed to the thing being depended on
- A dependency can have a name, although names rarely needed

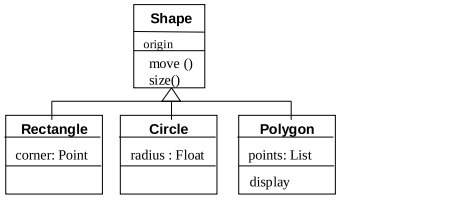


Relationships - Generalization

- A generalization is a relationship between a general thing (superclass) and a more specific kind of that thing (subclass)
- Generalization is used when it is necessary to show parent/child relationships
- A class may have zero, one, or more parents (single or multiple inheritance)
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy
- A class without parents and one or more children is called a root class or a base class
- A class without children is called a leaf class

Relationships - Generalization

- Graphically, a generalization is shown by a triangle in the parent side
- A generalization can have a name, although names are rarely needed

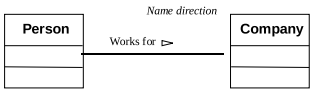


Relationships - Association

- An association is a structural relationship that specifies that objects of one thing are connected to objects of another
- Given an association between two classes it is possible to navigate from an object of one class to an object of the other class, and vice versa
- Association are used when it is necessary to show structural relationships

Relationships - Association

- Graphically, an association is rendered as a solid line connecting the same or different classes
- An association can have four adornments:
  - a name is used to describe the nature of the relationship
  - role
  - multiplicity
  - aggregation

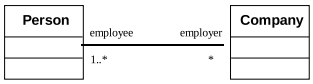


Association- Role

- When a class participates in an association it has a specific role that it plays in that relationship
- A role is just the face the class at the near end of the association presents to the class at the other end of the association
- The same class can play the same or different roles in other associations

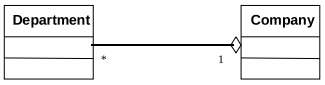
Multiplicity

- The multiplicity defines how many objects may be connected across an instance of an association
- The multiplicity give the lowest and upper limits
  - 0..1 : zero or one instance
  - 1..1: one and only one instance
  - 0..*: zero or more instance
  - 0..1, 3..4, 6..*: any number of objects other than 2 or 5



Relationships - Aggregation

- An aggregation models the whole/part relationship, one class represents a larger thing (whole) which consist of smaller things (parts)
- Aggregation is really just a special kind of association
- Graphically, is specified by adorning a plain association with an open diamond at the whole end

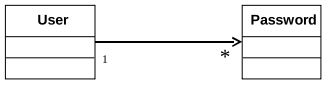


Relationships - Composition

- Composition is a form of aggregation, with a strong ownership and coincident lifetime as part of the whole
- Parts with non-fixed multiplicity may be created after the composite itself, but once created the live and die with it
- The composite must manage the creation and destruction of its parts
- Graphically, is specified by adorning a plain association with a filled diamond at the whole end

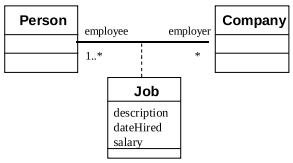
Navigation

- Although associations and aggregations are bi-directional by default, it is often desirable to restrict navigation to one direction
- If navigation is restricted, an arrowhead is added to indicate the direction of the navigation
- Graphically, the navigation is rendered as a redirected arrow



Association Classes

- **Models properties of a relationship**
- **Has both association and class properties**
- **Graphically, association classes are rendered as class symbol attached by a dashed line to an association**



Class Diagrams

- **Class diagrams are used to show the static design view of a system**
- **Involves modeling the vocabulary of the system, modeling collaborations, or modeling schemas**
- **Graphically, a class diagram is a collection of vertices and arcs**
- **Class diagrams commonly contain:**
  - **classes**
  - **interfaces**
  - **collaborations**
  - **dependency, generalization, and association relationships**

Class Diagram

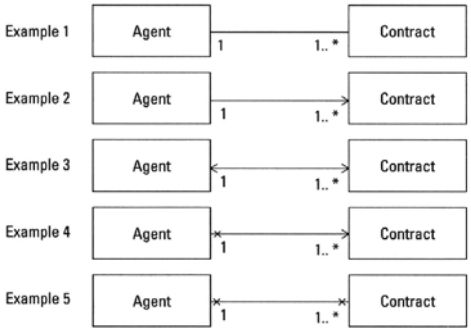
Class Diagram UML 2.0

- Attributes are called properties for all the properties of any diagram

Multiplicity

- If the ordering of the orders in association has meaning, {ordered} to the association end
- To show duplicates, {nonunique}
- To explicitly show the default, {unordered} and {unique}
- To show collection-oriented names, such as {bag} for unordered, nonunique.
- UML 1 allowed discontinuous multiplicities, such as 2, 4 (meaning 2 or 4, as in cars in the days before minivans). Discontinuous multiplicities weren't very common and UML 2 removed them.

Class Diagram UML 2.0 - Navigability



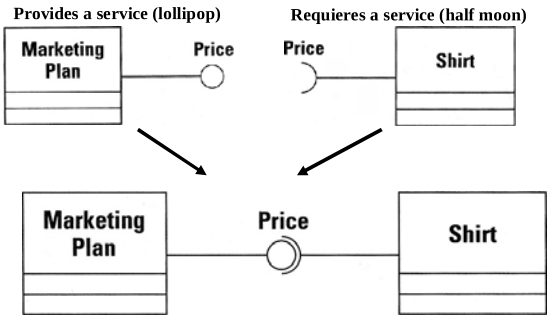
Interface

- **An interface is a collection of operations that are used to specify a service of a class or component**
- **Every interface must have a name**
- **An interface may have any number of operations (showed as a class)**
- **An interface may participate in generalization, association, and dependency relationships**
- **Graphically, an interface is rendered as a circle**

UML 2.0

- **An interface as abstract the class that realize it has to implement the method to acceded to the properties**

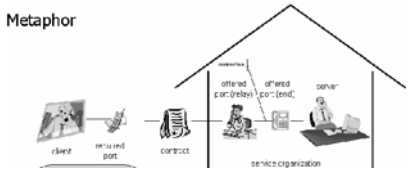
Interface UML 2.0



Ports

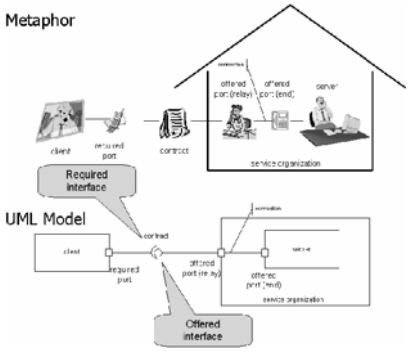
- Ports are instantiable connection points
- Ports may optionally be used in conjunction with structured classes to allow part instances to export out specific services or operations across the enclosing structured class

Ports



A client wants to get a service performed by a server, who is part of a service organization. He calls using his phone (a requiered port) to the secretary who is the only thing within the service organization visible to him. He points out that he has a contract and he can request a service to be performed. The secretary acts as a relay to pass the request onto the server via the end port (terminating at the server)

Ports



Instances

: Instance

- **An instance is a concrete manifestation of an abstraction to which a set of operations can be applied and which has a state that stores the effects of the operations**
- **Instance and object are largely synonymous**
- **Kind of objects:**
  - **Active. Has a execution thread, it can begin an activity**
  - **Passive. Its activities are activated be a stimuli**
  - **Client. Ask for a service**
  - **Server. Answer a service**

Instances

: Instance

- **Graphically, an instance is rendered by underlining its name**

: Frame

Keystrokes : Queue

Instances

: Instance

- Most instances are instances of classes, although there are instances of components, nodes, use cases, and associations
- For example, an instance of
  - a node is typically a computer that physically sits in a room
  - a component takes up some space on the file system
  - a customer record consumes some amount of physical memory
- Every instance must have a name, typically the first letter of all but the first word are capitalized (t or myCustomer)

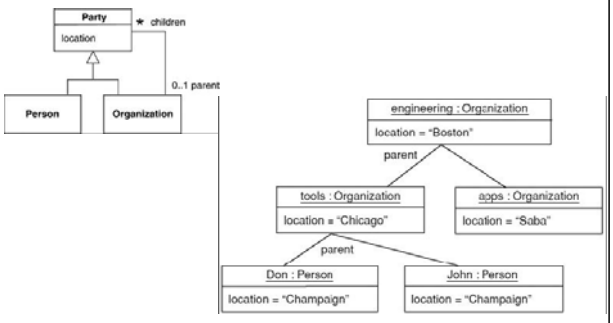
Object Diagrams

- Object diagrams model the instances of things contained in class diagrams
- An object diagram shows a set of objects and their relationship at a point in time
- Object diagrams are used to model the static design view or static process view of a system
- This involves modeling a snapshot of the system at a moment in time and rendering a set of objects, their state, and their relationships

Object Diagrams

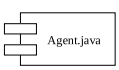
- Graphically, an object diagram is a collection of vertices and arcs
- Commonly an object diagram contains:
  - objects,
  - links,
  - may contain packages

Object Diagrams



Components

- A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces
- Components are used to model physical things that may reside on a node, such as executables, libraries, tables, files, and documents
- A component typically represents the physical packaging of otherwise logical elements, such as classes, interfaces, and collaborations
- Graphically, a component is rendered as a rectangles with tabs



Kinds of Components

- *Deployment component.* Components necessary and sufficient to form an executable system (DLL and EXEs)
- *Work product component.* Theses components are essentially the residue of the development process, consisting of things such as source code files and data files.
- *Execution component.* These components are created of an executing system, such as COM+ object, which is instantiated from a DLL

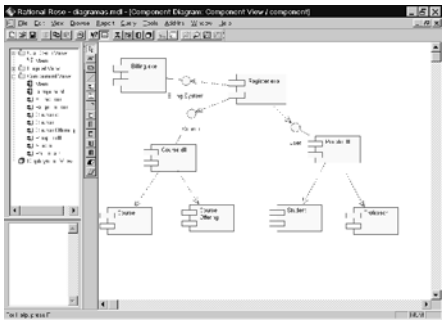
Component Stereotypes

- ***Executable,*** **specifies a component that may be executed on a node**
- ***Library,*** **specifies a static or dynamic object library**
- ***Table,*** **specifies a component that represents a database table**
- ***File,*** **specifies a component that represents a document containing source code or data**
- ***Document,*** **specifies a component that represents a document**

Component Diagrams

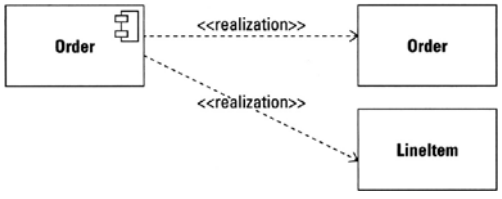
- **Component diagrams are used to model the static implementation view of a system**
- **Involves modeling the physical things that reside on a node, such as executables, libraries, tables, files, and documents**
- **Graphically, a component diagram is a collection of vertices and arcs**
- **Component diagrams contain:**
  - **Components**
  - **Interfaces**
  - **Dependency, generalization, association, and realization relationships**

Component Diagram

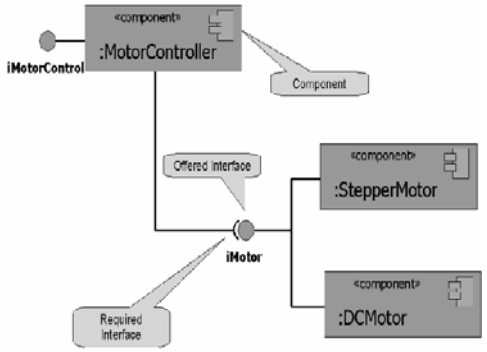


Component Diagram UML 2.0

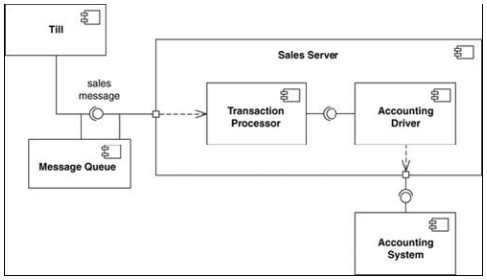
Classes Realization



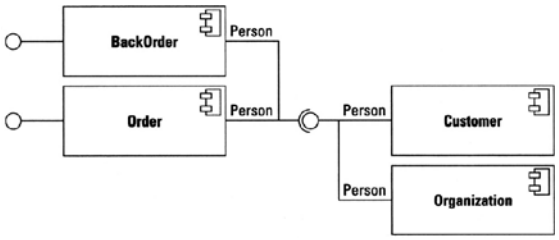
Ports



Component Diagram UML 2.0



Component Diagram UML 2.0

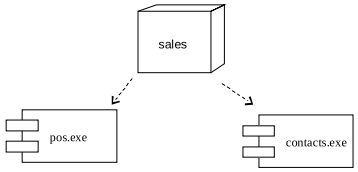


Nodes

- A node is a physical element that exit at run time and represents a computational resource, generally having at least some memory and, often, processing capability
- Nodes are used to model the topology of the hardware on which the system executes
- A node typically represents a processor or a device on which components may be deployed
- Graphically, a node is rendered as a cube

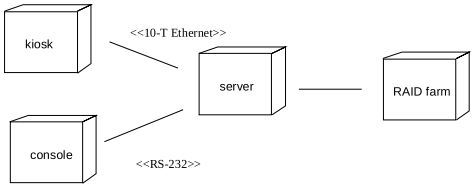
Nodes and Components

- Components are things that participate in the execution of a system, nodes are things that execute components
- Components represent the physical packaging of otherwise logical elements; nodes represent the physical deployment of components



Connections between Nodes

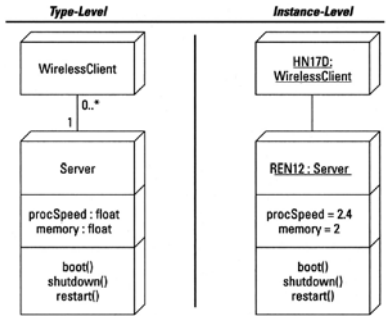
- The most common kind of relationship among nodes is an association, which represents a physical connection among nodes, such as an Ethernet connection, a serial line, or a shared bus



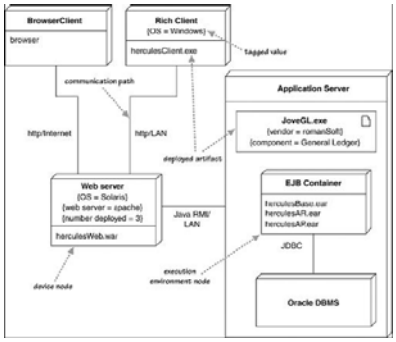
Deployment Diagrams

- The deployment diagram shows the configuration of run-time processing nodes and the components that live on them
- The deployment diagram models the topology of the hardware on which the system executes
- Graphically, a deployment diagram is a collection of vertices and arcs
- Deployment diagrams contain:
  - Nodes
  - Dependency and association relationships

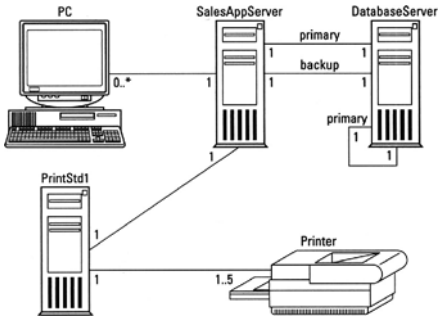
Deployment Diagrams



Deployment Diagrams



Deployment Diagrams



New UML Diagrams

UML 2.0

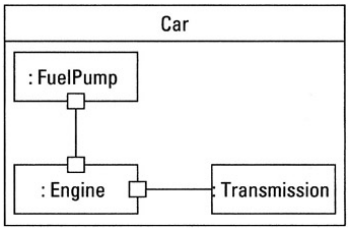
Composite Structure Diagram

Package Diagram

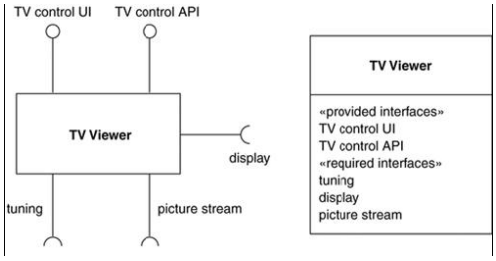
Composite Structures Diagram

- It is possible to decompose a class, component or collaboration into an internal structure
- This allows to take a complex object and break it down into more primitive part objects
- Packages are a compile-time grouping while composite structures show runtime grouping
- They are a natural fit for showing components and how they are broken into parts

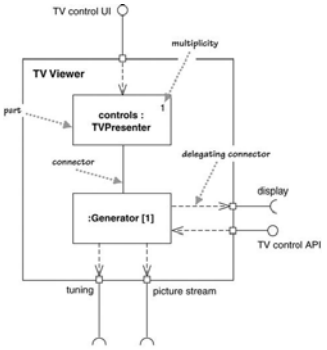
Composite Structures



Two ways of showing a TV viewer and its interfaces



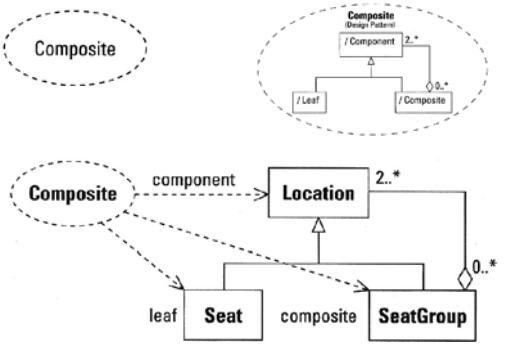
Two ways of showing a TV viewer and its interfaces



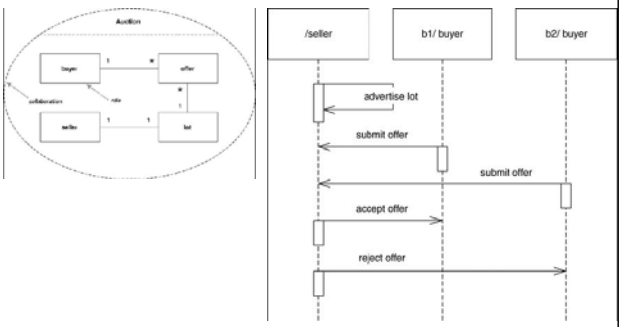
Collaborations

- A collaboration allows to name a conceptual chunk that encompass both static and dynamic aspects
- A collaboration names a society of classes, interfaces, and other elements that work together to provide some cooperative behavior that's bigger that the sum of all its parts
- Collaborations are used to specify the realization of use cases and operations, and to model the architecturally significant mechanisms of the system
- Graphically, a collaboration is rendered as an ellipse with dashed lines

Collaborations - Composite Design Pattern



Collaborations - Auction

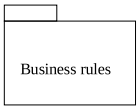


Package Diagram

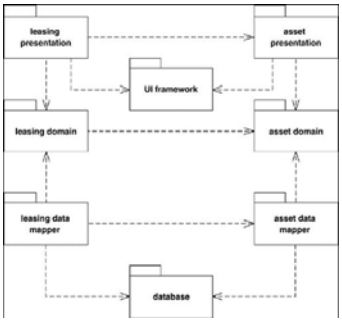
- The package is a general purpose mechanism for organizing modeling elements into groups
- A package is a grouping construct to group elements (classes, use cases, etc) together into higher-level units
- A package can group several diagrams showing different system characteristics
- Some things are visible outside the packages while others are hidden
- Well-designed packages group elements that are semantically close and that tend to change together they have to be loosely coupled and very cohesive
- Each class is a member of a single package. Packages can also be members of other packages
- A package can contain both subpackages and classes.

Packages

- Graphically, a package is rendered as a tabbed folder
- Every package must have a name
- A name is a textual string



Showing Dependencies among Packages



Part VI - Additional Elements and Summary

Common Mechanisms

- Specification
- Adornments
  - Notes
- Common Divisions
- Extensibility mechanisms
  - Stereotypes
  - Tagged Values
  - Constraints

Note

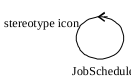
Notes are a mechanism to let designers to capture arbitrary comments and constraints

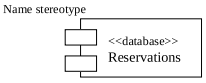
Consider the use of the broken design patterns here

See http://www.gamelan.com for an example of this applet

Stereotypes

- Extension of the vocabulary of the UML allowing the creation of new kinds of building blocks
- Graphically, a stereotype is rendered as a name enclosed by guillemets (<< >>) and placed above the name of another element
- As an option, the stereotype element may be rendered by using a new icon associated with that stereotype

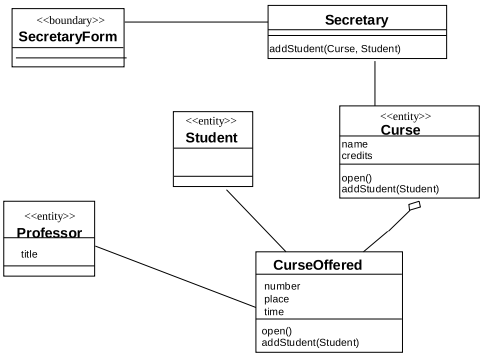




Stereotypes

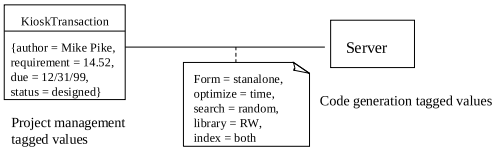
- **Stereotypes can be used to extend the UML notational elements**
- **Stereotypes may be used to classify and extend associations, inheritance relationships, classes, and components**
- **Examples:**
  - **Class stereotypes: boundary, control, entity, utility, exception**
  - **Inheritance stereotypes: uses and extends**
  - **Component stereotypes: subsystem**

Stereotypes



Tagged Value

- **Extension of the properties of a UML element allowing the creation of new information in that element's specification**
- **Graphically, a tagged value is rendered as a string enclosed by brackets and placed bellow the name of another element**



Constraint

- **Extension of the semantics of a UML element allowing the incorporation of new rules or modification of existing ones**
- **Graphically, a constraint is rendered as a string enclosed by brackets and placed near the associated element**