An Introduction to the Unified Modeling Language (UML)

UML in One Sentence

The UML is a graphical language for
- visualizing
- specifying
- constructing
- documenting

artifacts of a software-intensive system.

Visualizing

- explicit model facilitates communication
- some structures transcend what can be represented in programming language
- each symbol has well-defined semantics behind it

Specifying

The UML addresses the specification of all important analysis, design, and implementation decisions.

Constructing

- Forward engineering: generation of code from model into programming language
- Reverse engineering: reconstructing model from implementation
- Round-trip engineering: going both ways

Documenting

Artifacts include:
- deliverables, such as requirements documents, functional specifications, and test plans
- materials that are critical in controlling, measuring, and communicating about a system during development and after deployment

UML and Blueprints

The UML provides a standard way to write a system's "blueprints" to account for

- conceptual things (business processes, system functions)
- concrete things (C++/Java classes, database schemas, reusable software components)

Reasons to Model

- to communicate the desired structure and behavior of the system
- to visualize and control the system's architecture
- to better understand the system and expose opportunities for simplification and reuse
- to manage risk

Principles of Modeling

- choice of models to create very influential as far as how to attack problem and shape solution
- every model may be expressed at different levels of precision
- best models connected to reality
- no single model is sufficient

Structural Diagrams

Used to visualize, specify, construct, document static aspects of system

- class diagram
- package diagram [not standard UML]
- object diagram
- component diagram
- deployment diagram

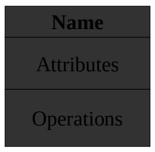
Common Uses of Class Diagrams

- to model vocabulary of the system, in terms of which abstractions are part of the system and which fall outside its boundaries
- to model simple collaborations (societies of elements that work together to provide cooperative behavior)
- to model logical database schema (blueprint for conceptual design of database)

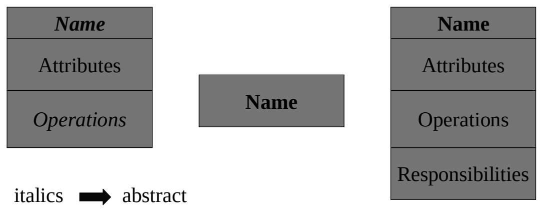
Class

- A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics.
- An attribute is a named property of a class that describes a range of values that instances of the property may hold.
- An operation is a service that can be requested from an object to affect behavior.

Class Notation



Alternative Class Notations



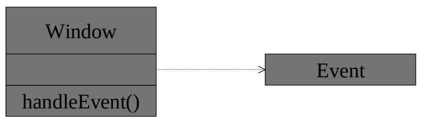
Relationships

connections between classes
- dependency
- generalization
- association

Dependency

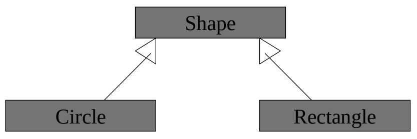
A dependency is a “using” relationship within which the change in the specification of one class may affect another class that uses it.

Example: one class uses another in operation



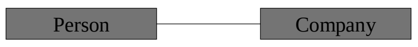
Generalization

A generalization is a “kind of” or “is a” relationship between a general thing (superclass or parent) and a more specific thing (subclass or child).



Association

An association is a structural relationship within which classes or objects are connected to each other. (An association between objects is called a link.)

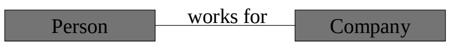


Association Adornments

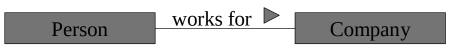
- name
- role
- multiplicity
- aggregation
- composition

Association Name

describes nature of relationship:

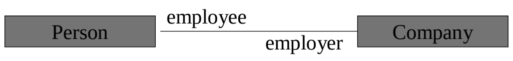


can also show direction to read name:



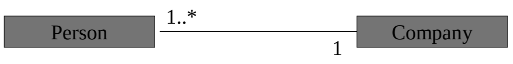
Association Roles

- describe “faces” that classes present to each other within association
- class can play same or different roles within different associations



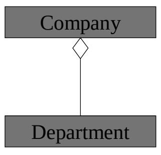
Association Multiplicity

- possible values same as for classes: explicit value, range, or * for “many”
- Example: a Person is employed by one Company; a Company employs one or more Persons



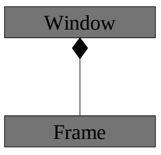
Aggregation

Aggregation is a “whole/part” or “has a” relationship within which one class represents a larger thing that consists of smaller things.



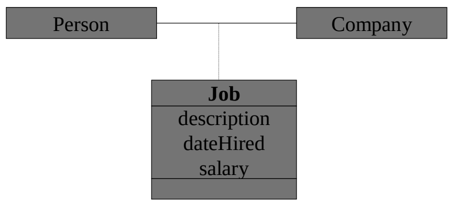
Composition

Composition is a special form of aggregation within which the parts are inseparable from the whole.



Association Classes

An association class has properties of both an association and a class.



Behavioral Diagrams

Used to visualize, specify, construct, document dynamic aspects of system

- use case diagram
- sequence diagram
- collaboration diagram
- statechart diagram
- activity diagram

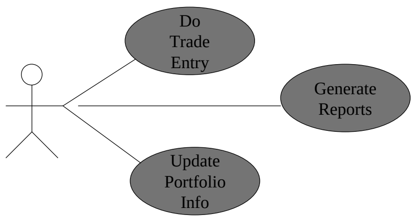
Use Case and Actor

- A use case is a sequence of actions, including variants, that a system performs to yield an observable result of value to an actor.
- An actor is a coherent set of roles that human and/or non-human users of use cases play when interacting with those use cases.

Flows of Events

- The main flow of events (basic course of action) describes the “sunny-day” scenario.
- Each exceptional flow of events (alternate course of action) describes a variant, such as an error condition or an infrequently occurring path.

Simple Use Case Diagram

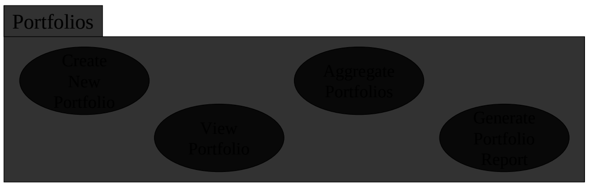


Organizing Use Cases

- packages
- generalization
- include
- extend

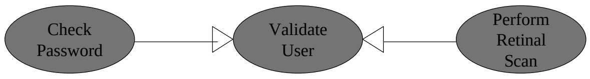
Use Case Packages

Packages of use cases can be very useful in assigning work to sub-teams.



Use Case Generalization

You can generalize use cases just like you generalize classes: the child use case inherits the behavior and meaning of the parent use case, and can add to or override that behavior.


Check Password
Validate User
Perform Retinal Scan