Software Design and UML

Logistics

- Syllabus / Student Info Forms
  – For those not here yesterday
- LDAP database
  – Everyone check e-mail listing?
  – Will send e-mail after this class.

Plan for today

- Building a software system
  – Software Development Cycle
  – Documenting your design using UML

Software Development Cycle

- Process for software development
  – People management
  – Work management
  – Team management
- Caveat: These processes are merely guidelines
  – Your actual mileage may vary!

Software Development Cycle

- Gather Requirements
  – Find out what the user needs
- System Analysis
  – Express these needs formally in system terms
- Design
  – Design a high level solution
- Implementation
  – Turn solution into code
- Testing
  – Verify that the solution works
- Maintenance
  – Iterate the cycle

Software Development Cycle

- Problem Domain
  – Gather Requirements / System Analysis
- Solution domain
  – Design / Implementation
  – Note: no code until implementation!

Software Development Cycle

- Testing
  - Unit testing
  - Integration testing
  - System testing
  - Reviews
    - Requirements / Design / Code

Software Development Cycle

- Maintainance
  - Modifications – iterate over complete cycle
- Note: This is just one methodology for software developments, there are others (e.g. eXtreme Programming).
- Questions?

Unified Modeling Language

- From the UML FAQ:
  - "The Unified Modeling Language is a third-generation method for specifying, visualizing, and documenting the artifacts of an object-oriented system under development."
  - Booch, Jacobson, Rumbaugh (the Three Amigos)
    - All three now work at Rational Software

Unified Modeling Language

- UML is a language for describing models.
  - Describes what a system is supposed to do but not how it should be implement.
  - Analysis and Design NOT Implementation.
  - CASE tools can generate code from well specified designs.

Unified Modeling Language

- Major Components
  - Entities
    - things in your model
  - Relationships
    - associations between things in the model
  - Diagrams
    - Graphical representation of elements and relationships that present different views of the system.
    - Often presented as a graph (shapes connected by arrows).

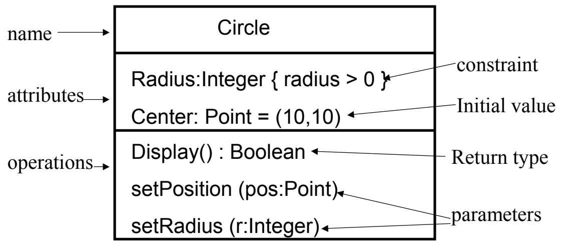
Unified Modeling Language

- UML defines numerous types of diagrams
- In this class we will focus on the following:
  - Class diagrams
    - Illustrates classes/objects and relationships
  - Use Case diagrams
    - Illustrates user interaction (scenerios) with system
  - Sequence Diagrams
    - Illustrates objects interaction over time in realizing a use case.

Class Diagrams

- Classes and Objects
  - All objects have the following:
    - Name – how an object is identified
    - Attributes – defines an object’s state
    - Operations – defines an object’s behavior
  - Classes
    - Categories of objects with the same set of attributes and behavior
    - Objects are instantiations of classes

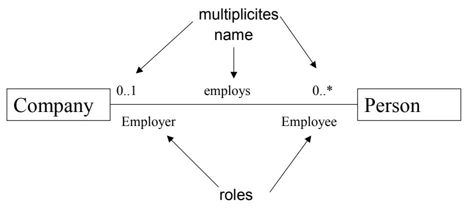
Class Diagrams – Classes



Class Diagrams -- Relationships

- Associations
  - Relationship between different objects of different classes
  - Associations can have the following:
    - Name – identifies the association type
    - Multiplicity – indicates how many objects can participate in the association
    - Roles – Meaning of classes involved
  - Represented by lines connecting associated classes

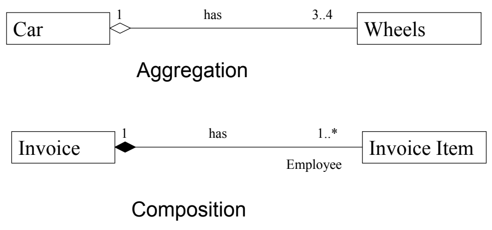
Class Diagrams -- Associations



Class Diagrams -- Relationships

- Aggregation
  - Specifies a “whole”/”part” relationship”
  - has-a relationship
    - Indicated by a line with an unfilled diamond at the end
  - <u>Composition</u> – strong aggregation where the part generally does not exist without the whole.
    - Indicated by a line with a filled diamond at the end

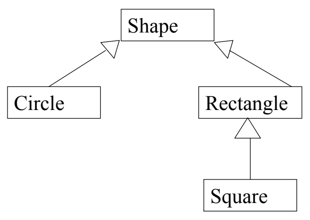
Class Diagrams -- Aggregation



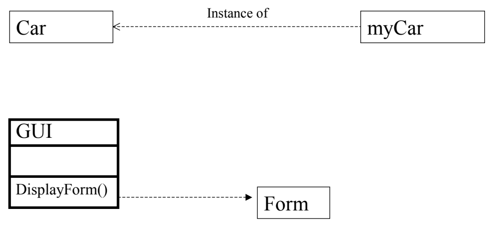
Class Diagrams -- Relationships

- Generalization
  - is-A relationship
  - Indicates inheritance
    - Indicated by a line with an open triangle.
- Dependency
  - Relationship where a change in one element requires a change in the other
    - Instantiation Relationships
    - Temporary associations (operation arguments)
    - Creator / Createe relationship
    - Indicated by a dotted line

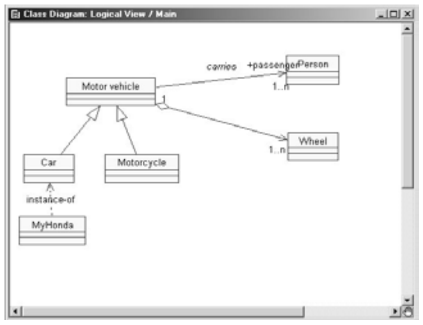
Class Diagrams -- Generalization



Class Diagrams -- Dependency



Class Diagram – Summary



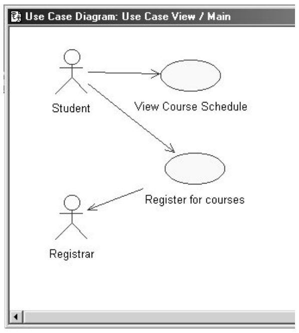
Class Diagrams -- Summary

- Classes / Objects – represented as boxes
  - Name / Attributes / Operations
- Relationships – lines connecting boxes
  - Associations
  - Aggregations / Composition
  - Generalization
  - Dependency
- Questions?

Use Case Diagram

- Use case – Scenario about system use from a external ***user perspective***.
  - Extremely useful tool for requirements gathering and analysis.
  - Use cases are indicated by an oval
  - Actor – Entity located outside of a system that is involved in the interaction with the system in a use case.
  - Actors are indicated by a stick person.

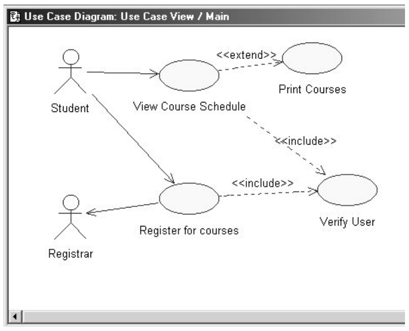
Use Case Diagram



Use Case – Relationships

- Use Cases can have relationships with other use cases
  – Include –use case that is performed during the course of another use case.
  – Extend – Adding extra steps to an already existing use case.

Use Case – Relationships



Programming by Contract

- Introduced by Bertrand Meyer, the creator of Eiffel.
- Creates a contract between the software developer and software user
  – Every feature, or method, starts with a *precondition* that must be satisfied by the consumer of the routine.
  – each feature ends with *postconditions* which the supplier guarantees to be true (if and only if the preconditions were met).
  – each class has an *invariant* which must be satisfied after any changes to the object represented by the class.

Use Case -- Documentation

- To be documented with a use case:
  – Sequence of steps that occur in the scenario
  – Preconditions
  – Postconditions
  – Variations and alternative scenarios

Use Case – Register for courses

- Precondition:
  – Student has been assigned a valid id/password
- Postcondition:
  – Student becomes registered and can attend class.

Use Case – Register for courses

- Sequence of events
  - Student logs into system
  - System extracts student data from DB
  - Based on this data, system presents a menu of courses student can take
  - Student chooses course
  - Notification sent to registrar to add student to course.

Use Case – Register for courses

- Alternative scenarios
  - Student database unavailable
  - Courses cannot be retrieved
  - Course chosen by student is full.
  - Communication to registrar is unavailable.

Use case diagram – Summary

- Use case – Scenario about system use from a external ***user perspective***.
  - Ovals in diagram
- Actor – Entity located outside of a system that is involved in the interaction with the system in a use case.
  - Stick person
- Relationships
  - Extend / Include
- Documentation
- Questions?

Summary

- Software Design and Life Cycle
  - Requirements / Analysis / Design / Implementation / Test / Maintenance
- UML
  - Class Diagrams
  - Use Case Diagrams
  - Sequence Diagrams (next time)