Software Design with UML Class Diagrams

Lecture 2

Outline

- What is UML?
- What is a UML class diagram?
  - What kind of information goes into it?
  - How do I create it?
  - When should I create it?
- Examples
- Tools

Software Design

- **Design**: specifying the **structure** of a software system and its **functions (behaviour)**
  - Its an opportunity to get insights on design alternatives to make appropriate design choice
  - You can also evaluate the extent to which the system complies with end user expectations

Software Design

- A transition from "what" the system must do, to "how" the system will do it
  - What classes will we need to implement a system that meets our requirements?
  - What fields and methods will each class have?
  - How will the classes interact with each other?

- The outcome of a software design activity is a **domain model**

Object-Oriented Analysis

Domain Model:

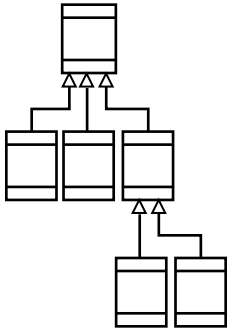
- A conceptual model of the domain that incorporates both behavior and data.
  - A formal/semiformal/informal representation of a domain with concepts, roles, datatypes, individuals, and associated rules of interaction.

What is UML

- Diagrammatic representations of an OO system
  - Programming languages are not abstract enough for OO design
  - UML is an open standard; lots of companies use it

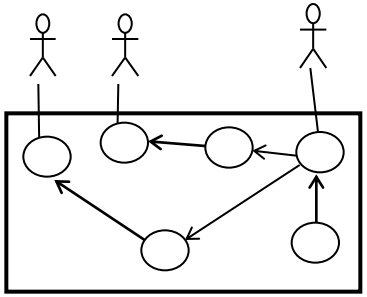
- What is legal UML?
  - A *descriptive* language: rigid formal syntax (like programming)
  - A *prescriptive* language: shaped by usage and convention
  - It's okay to omit things from UML diagrams if they are not needed

UML Modelling Notations



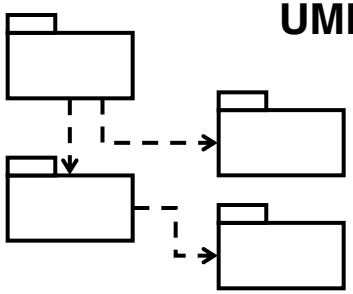
UML Class Diagrams

information structure
relationships between data items modular structure for the system



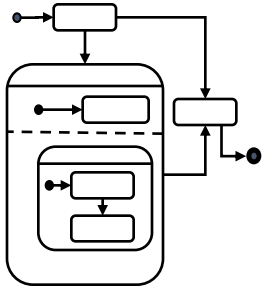
Use Cases

user's view Lists functions visual overview of the main requirements



UML Package Diagrams

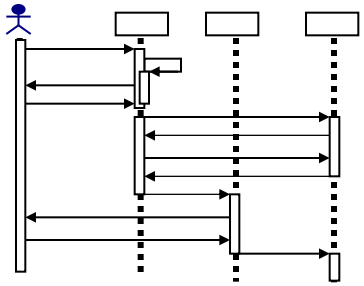
Overall architecture
Dependencies between components



(UML) Statecharts

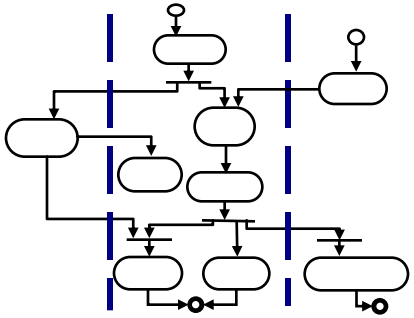
responses to events

dynamic behavior event ordering, reachability, deadlock, etc



UML Sequence Diagrams

individual scenario
interactions between users and system
Sequence of messages



Activity diagrams

business processes;
concurrency and synchronization;
dependencies between tasks;

Class Diagrams

UML class diagrams

- What is a UML class diagram?
  - A diagram of the classes in an OO system, their fields and methods, and connections between the classes that interact or inherit from each other

- Things not represented in a UML class diagram:
  - details of how the classes interact with each other
  - algorithmic details; how a particular behavior is implemented

How do we design classes?

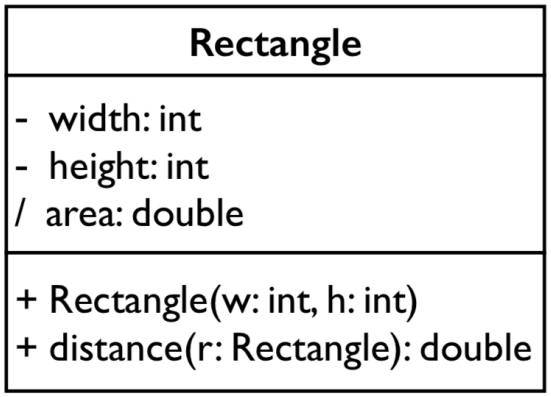
Identify classes and interactions from project requirements:

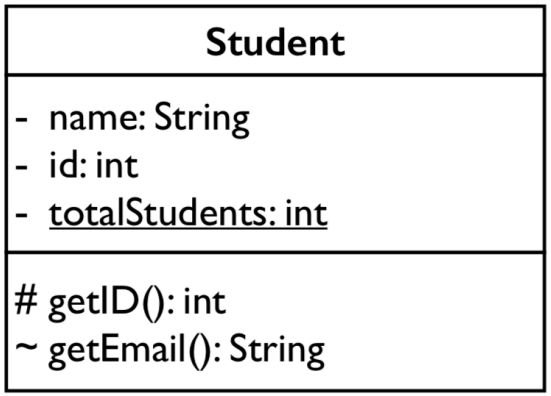
- Nouns are potential classes, objects, and fields
- Verbs are potential methods or responsibilities of a class
- Relationships between nouns are potential interactions (containment, generalization, dependence, etc.)

- Which nouns in your project should be classes?
- Which ones are fields?
- What verbs should be methods?
- What are potential interactions between your classes?

Diagram of a class

- class name in top of box
  - write <<interface>> on top of interfaces' names
  - use *italics* for an *abstract class* name
- attributes (optional)
  - should include all fields of the object
- operations / methods (optional)
  - may omit trivial (get/set) methods
    - but don't omit any methods from an interface!
  - should not include inherited methods

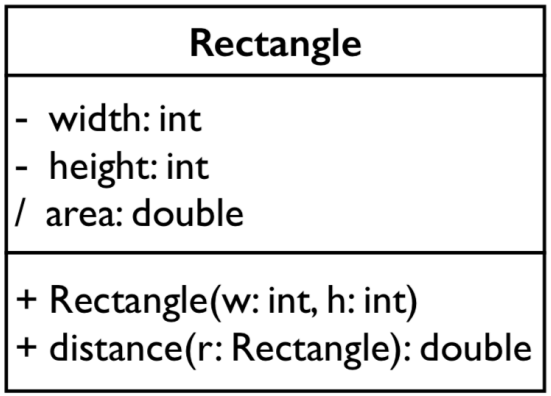


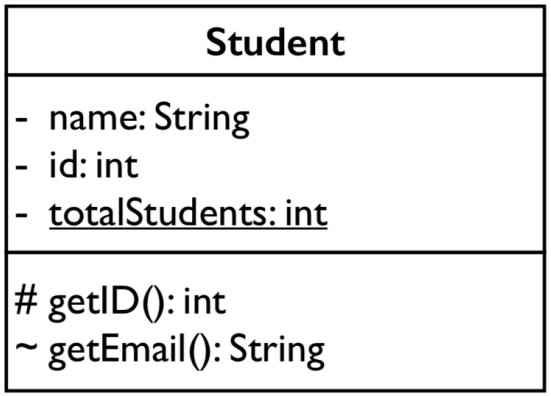


Class attributes (fields, instance variables)

visibility name : type [count] = default_value

- visibility:
  - \+ public
  - # protected
  - \- Private
  - ~ package (default)
  - / derived
- underline <u>static attributes</u>
- **derived attribute**: not stored, but can be computed from other attribute values

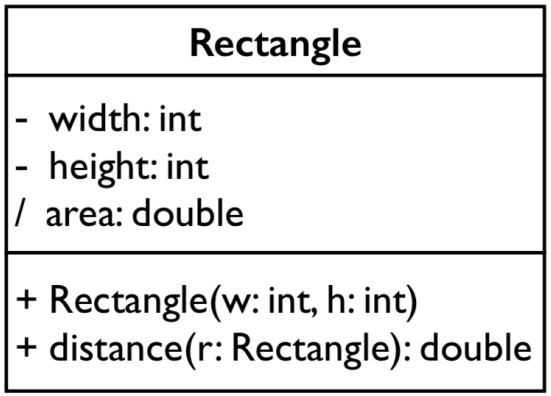


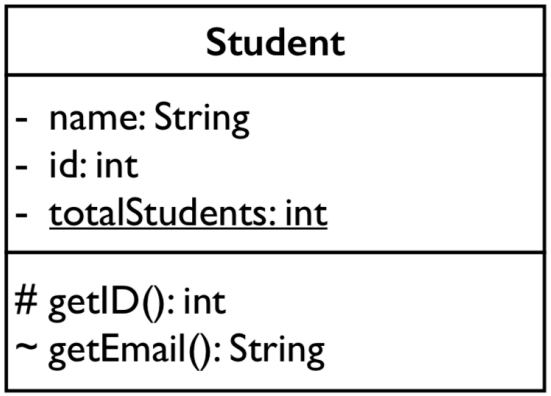


Class operations / methods

visibility name(parameters) : return_type

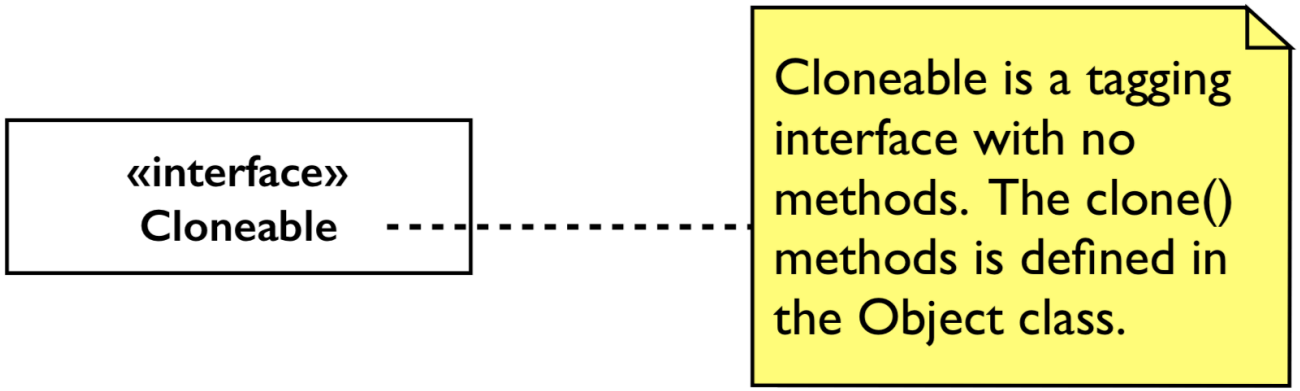
- visibility:
  - \+ public
  - # protected
  - \- Private
  - ~ package (default)
- underline static methods
- parameters listed as name : type
- omit return_type on constructors and when return type is void





Comments

- Represented as a folded note, attached to the appropriate class/method/etc by a dashed line



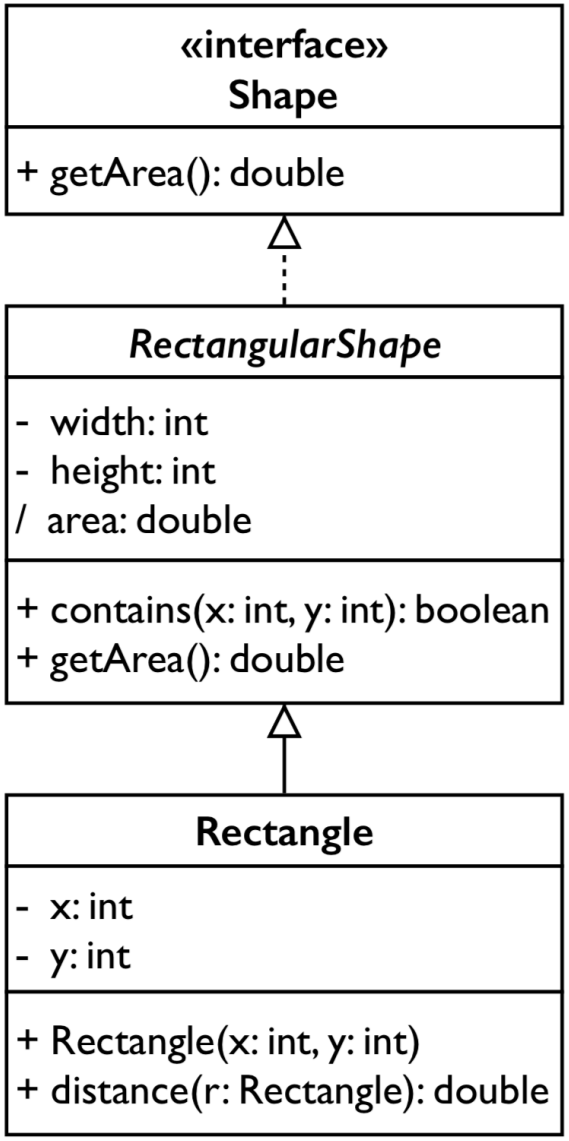
Relationships between classes

- Generalization: an inheritance relationship
  - inheritance between classes
  - interface implementation
- Association: a usage relationship
  - dependency
  - aggregation
  - composition

Generalization relationships

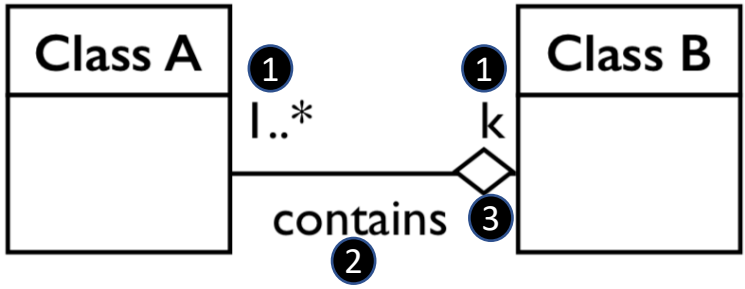
- Hierarchies drawn top-down with arrows point upward to parent.
- Line/arrow styles indicate if parent is a(n):
  - class: solid line, black arrow
  - abstract class: solid line, white arrow
  - interface: dashed line, white arrow

- Often omit trivial / obvious generalization relationships, such as drawing the Object class as a parent



Associational (usage) relationships

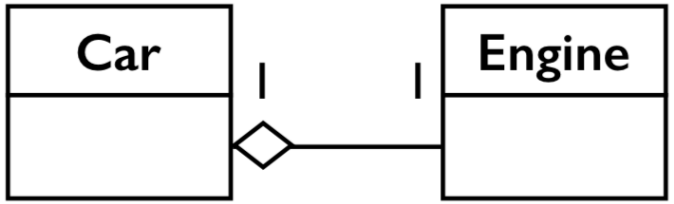
1. Multiplicity (how many are used)
   - * (zero or more)
   - 1 (exactly one)
   - 2..4 (between 2 and 4, inclusive)
   - 3..* (3 or more, * may be omitted)



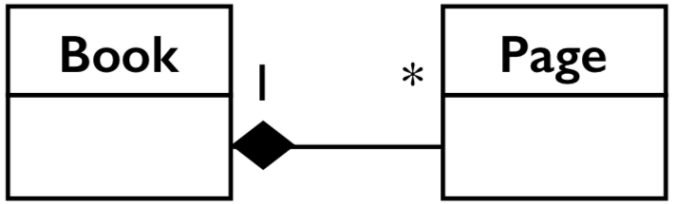
2. Name (what relationship the objects have)
3. Navigability (direction)

Association multiplicities

- One-to-one
  - Each car has exactly one engine.
  - Each engine belongs to exactly one car.



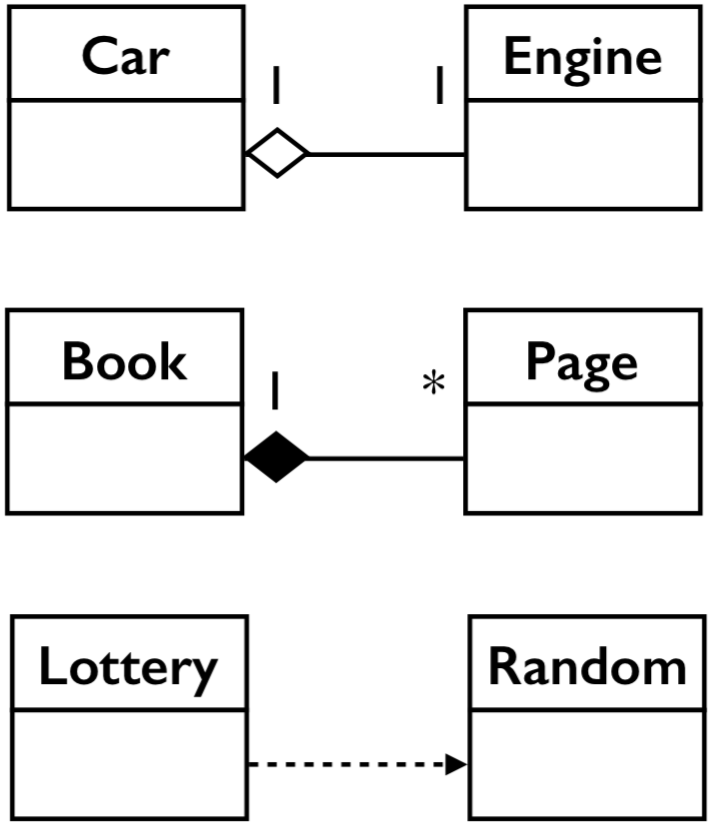
- One-to-many
  - Each book has many pages.
  - Each page belongs to exactly one book.



Association types

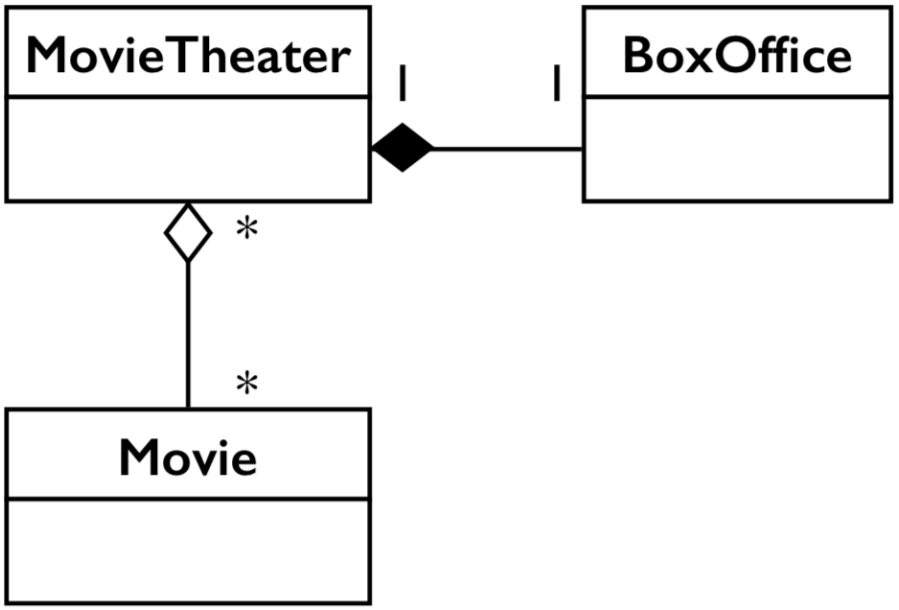
- **Aggregation**: "is part of"
  - symbolized by a clear white diamond

- Composition: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond
- Dependency: "uses temporarily"
  - symbolized by dotted line
  - often is an implementation detail, not an intrinsic part of the object's state

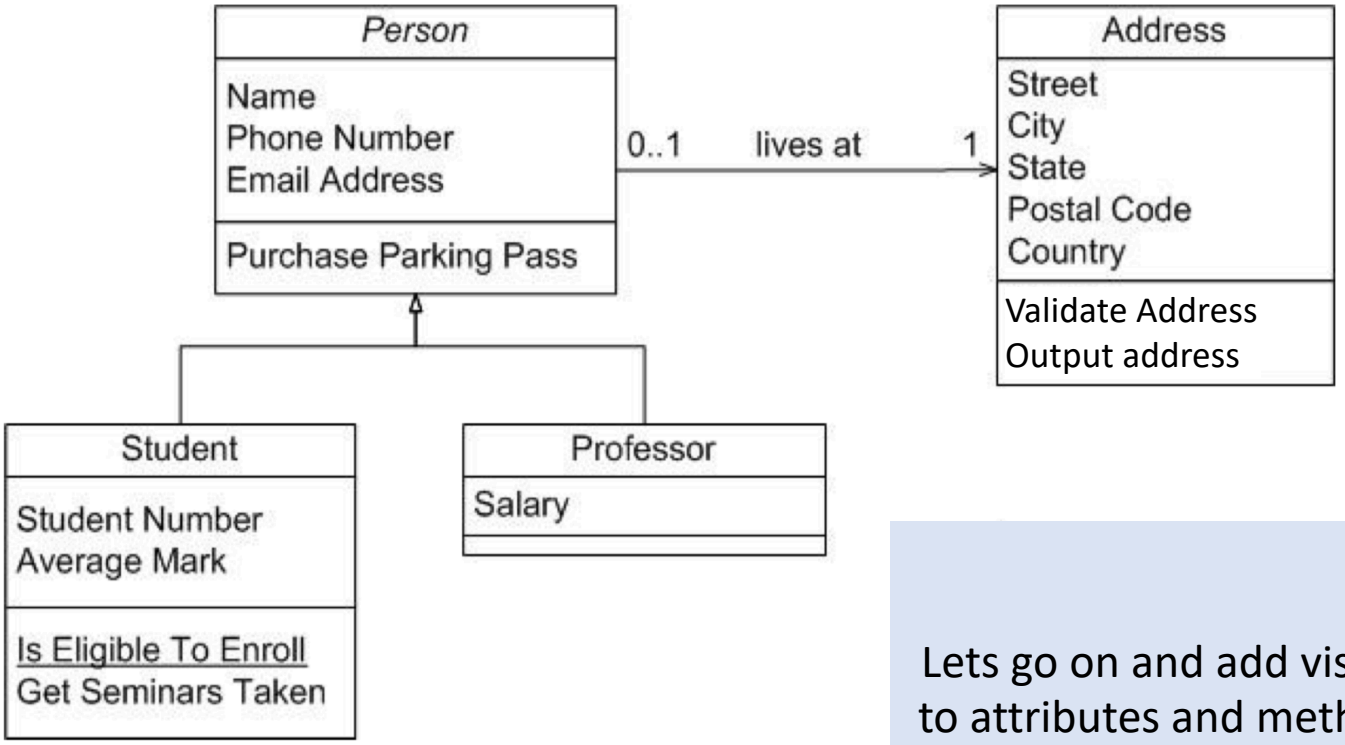


Example: Aggregation/composition

- If the movie theater goes away
  - so does the box office: composition
  - but movies may still exist: aggregation



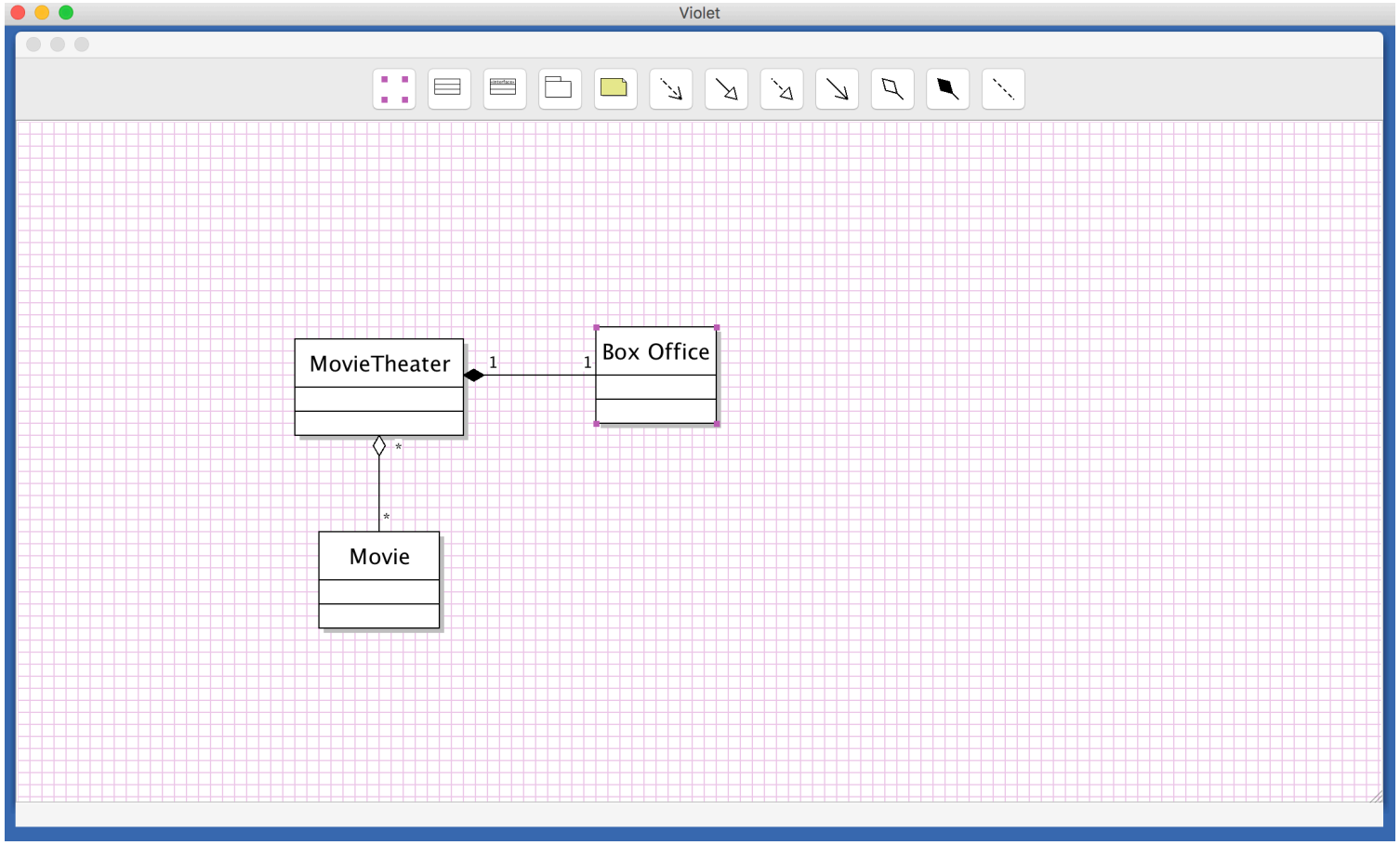
Exercise: Persons



Lets go on and add visibility to attributes and methods…

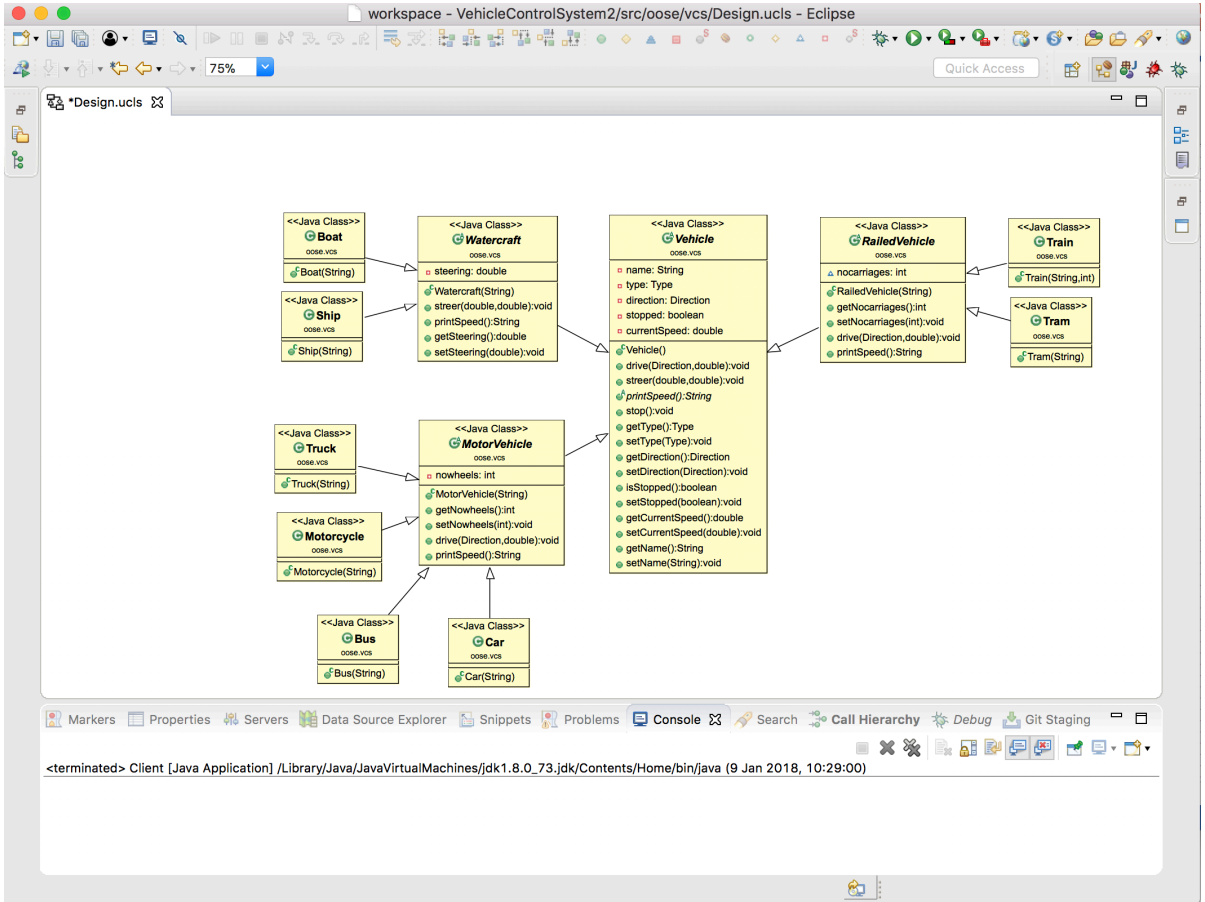
Tools for creating UML Diagrams

- Violet (free) http://horstmann.com/violet/



Tools for creating UML Diagrams

- ObjectAid UML Explorer (free) -Works as eclipse plugin
- http://www.objectaid.com/class-diagram



What Class diagrams are great for

- discovering related data and attributes
- getting a quick picture of the important entities in a system
- seeing whether you have too few/many classes
- seeing whether the relationships between objects are too complex, too many in number, simple enough, etc.
- spotting dependencies between one class/object and another

What Class diagrams are NOT great for

- discovering algorithmic (not data-driven) behavior
- finding the flow of steps for objects to solve a given problem
- understanding the app's overall control flow (event-driven? web-based? sequential? etc.)

Summary

- A design specifies the structure of how a software system will be written and function.
- UML is a language for describing various aspects of software designs.
- UML class diagrams present a static view of the system, displaying classes and relationships between them.