Software Design with UML Class Diagrams

Object Oriented Software Engineering

Lecture 2: Part 2

Dr. Graham McDonald

graham.mcdonald@glasgow.ac.uk

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 <u>not</u> 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?

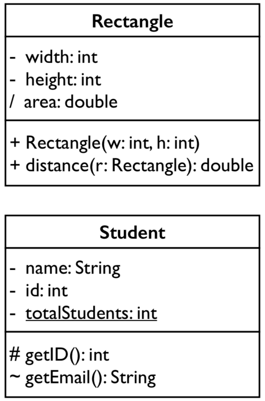
Identify classes and interactions from project requirements:

- **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



Rectangle

- width: int
- height: int
/ area: double

+ Rectangle(w: int, h: int)
+ distance(r: Rectangle): double

Student

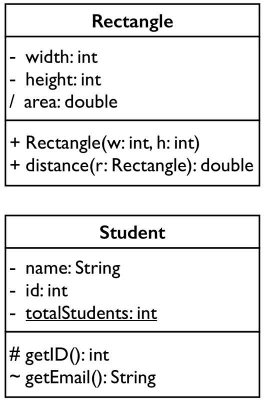
- name: String
- id: int
- totalStudents: int

getID(): int
~ getEmail(): String

Class Attributes (fields, instance variables)

visibility name : type [count] = default_value

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



Rectangle

- width: int
- height: int
/ area: double

+ Rectangle(w: int, h: int)
+ distance(r: Rectangle): double

Student

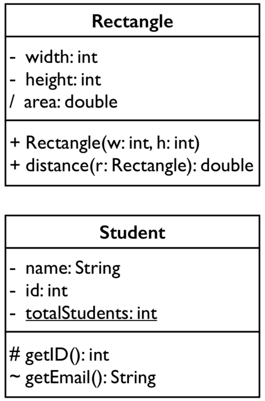
- name: String
- id: int
- totalStudents: int

getID(): int
~ getEmail(): String

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



Rectangle

- width: int
- height: int
/ area: double

+ Rectangle(w: int, h: int)
+ distance(r: Rectangle): double

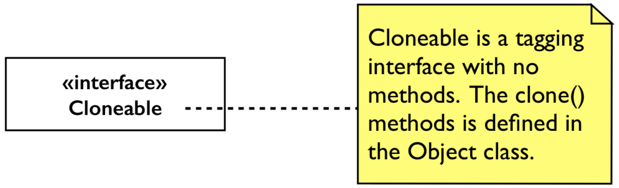
Student

- name: String
- id: int
- totalStudents: int

getID(): int
~ getEmail(): String

Comments

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



«interface»
Cloneable

Cloneable is a tagging interface with no methods. The clone() methods is defined in the Object class.

Software Design with UML Class Diagrams

Object Oriented Software Engineering

Lecture 2: Part 3

Dr. Graham McDonald

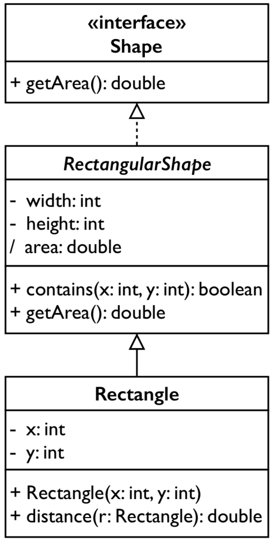
graham.mcdonald@glasgow.ac.uk

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, white arrow
  - **interface**: dashed line, white arrow
- We often omit trivial / obvious generalization relationships, such as drawing the Object class as a parent



«interface»
Shape

+ getArea(): double

RectangularShape

- width: int
- height: int
/ area: double

+ contains(x: int, y: int): boolean
+ getArea(): double

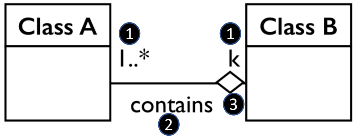
Rectangle

- x: int
- y: int

+ Rectangle(x: int, y: int)
+ distance(r: Rectangle): double

Associational (usage) Relationships

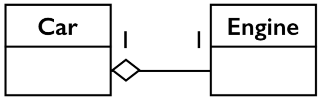
1. **Multiplicity** (how many are used)
   - * (zero or more)
   - 1 (exactly one)
   - 2..4 (between 2 and 4, inclusive)
   - 3..* (3 or more, * may be omitted)



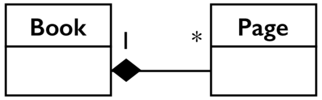
2. **Name** (what relationship the objects have)
3. **Navigability** (direction)

Association Multiplicities

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

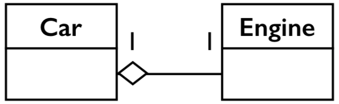


- **One-to-many**
  - Each book has many pages.
  - Each page belongs to exactly one book.

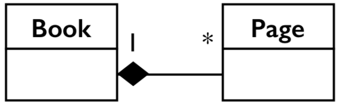


Association Types

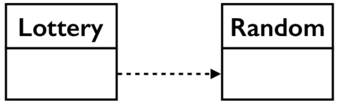
- **Aggregation**: “is part of”
  - symbolized by a clear white diamond



- **Composition**: “part can’t exist on its own”
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond

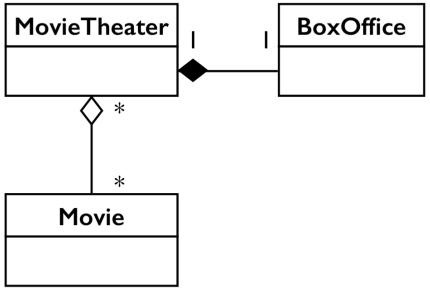


- **Dependency**: “uses temporarily”
  - symbolized by dotted line
  - often is an implementation detail, not an intrinsic part of the object's state

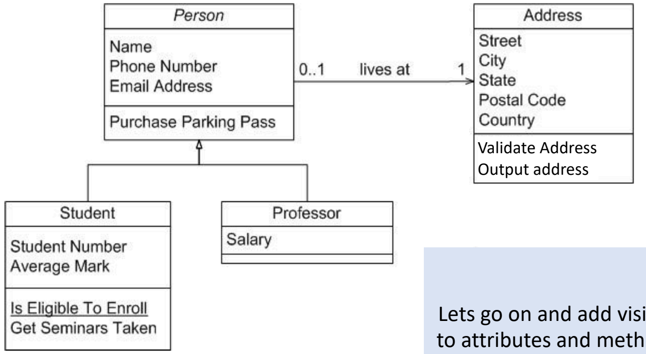


Example: Aggregation/Composition

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



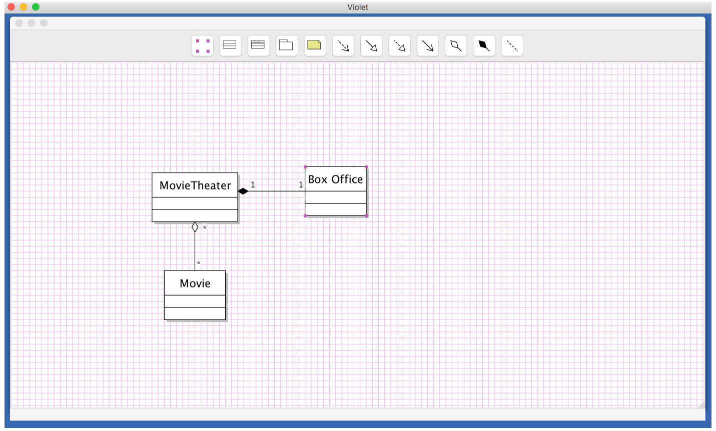
Exercise: Persons



Lets go on and add visibility to attributes and methods…

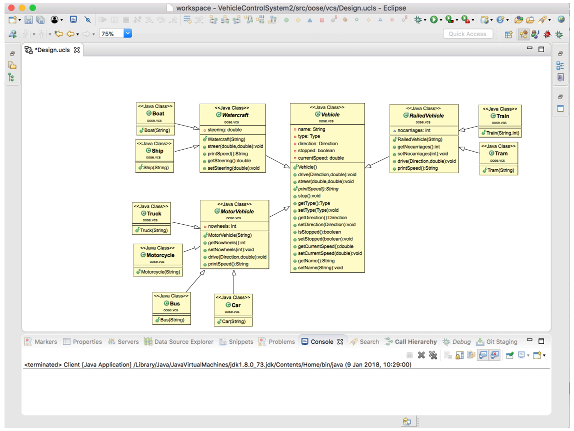
Tools for creating UML Diagrams

- 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


workspace - VehicleControlSystem2/src/oose/vcs/Design.ucls - Eclipse
75%
Quick Access
*Design.ucls
<<Java Class>> Boat oose.vcs Boat(String)
<<Java Class>> Ship oose.vcs Ship(String)
<<Java Class>> Watercraft oose.vcs
steering: double
Watercraft(String)
streer(double,double):void
printSpeed():String
getSteering():double
setSteering(double):void
<<Java Class>> Vehicle oose.vcs
name: String
type: Type
direction: Direction
stopped: boolean
currentSpeed: double
Vehicle()
drive(Direction,double):void
streer(double,double):void
printSpeed():String
stop():void
getType():Type
setType(Type):void
getDirection():Direction
setDirection(Direction):void
isStopped():boolean
setStopped(boolean):void
getCurrentSpeed():double
setCurrentSpeed(double):void
getName():String
setName(String):void
<<Java Class>> RailedVehicle oose.vcs
nocarriages: int
RailedVehicle(String)
getNocarriages():int
setNocarriages(int):void
drive(Direction,double):void
printSpeed():String
<<Java Class>> Train oose.vcs Train(String,int)
<<Java Class>> Tram oose.vcs Tram(String)
<<Java Class>> Truck oose.vcs Truck(String)
<<Java Class>> Motorcycle oose.vcs Motorcycle(String)
<<Java Class>> MotorVehicle oose.vcs
nowheels: int
MotorVehicle(String)
getNowheels():int
setNowheels(int):void
drive(Direction,double):void
printSpeed():String
<<Java Class>> Bus oose.vcs Bus(String)
<<Java Class>> Car oose.vcs Car(String)
Markers
Properties
Servers
Data Source Explorer
Snippets
Problems
Console
Search
Call Hierarchy
Debug
Git Staging

```
<terminated> Client [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_73.jdk/Contents/Home/bin/java (9 Jan 2018, 10:29:00)
```

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.