SWEN20003
Object Oriented Software Development

Modelling Classes and Relationships

Shanika Karunasekera
karus@unimelb.edu.au

University of Melbourne

The Road So Far

- Java Foundations
  - A Quick Tour of Java
- Object Oriented Programming Foundations
  - Classes and Objects
  - Arrays and Strings
  - Input and Output
  - *Software Tools and Bagel*
  - Inheritance and Polymorphism
  - Interfaces and Polymorphism

The Road Ahead

- Advanced Object Oriented Programming and Software Design
    - Modelling Classes and Relationships
    - Generics
    - Collections and Maps
    - Exceptions
    - Design Patterns 1 & 2
    - Software Testing and Design
    - Asynchronous Programming
    - Advanced Java Concepts

Lecture Objectives

After this lecture you will be able to:

- Identify **classes** and **relationships** for a given problem specification
- Represent classes and relationships in **UML**
- Develop simple **Class Diagrams**

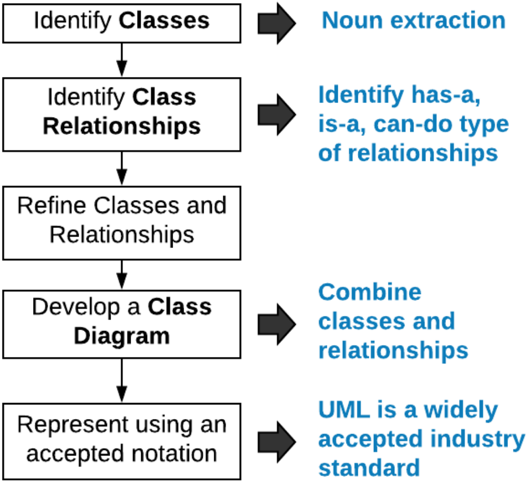
Hints for Project 2; pay attention!

First Steps

Before writing Java code:

Design your system, then **implement** the system (write code) **following your design!**

Design Algorithm



Identify **Classes**
Noun extraction
Identify **Class Relationships**
Identify has-a, is-a, can-do type of relationships
Refine Classes and Relationships
Develop a **Class Diagram**
Combine classes and relationships
Represent using an accepted notation
UML is a widely accepted industry standard

Problem Statement

This game involves a number of characters, like the player, their allies, and their enemies. Some of these characters walk around the world, and some allow the player to interact with them.

There are also other objects in the game like keys, blocks, and power-ups, some of which can be interacted with, some that can be collected/picked up, and some that can be "used" after being picked up.

Some objects in the game have health and can "die", while some are killed or removed instantly. Other objects can attack and defend other characters.

Identifying Classes - Noun Extraction

This game involves a number of characters, like the player, their allies, and their enemies. Some of these characters walk around the world, and some allow the player to interact with them.

There are also other objects in the game like keys, blocks, and power-ups, some of which can be interacted with, some that can be collected/picked up, and some that can be "used" after being picked up.

Some objects in the game have health and can "die", while some are killed or removed instantly. Other objects can attack and defend other characters.

Class Design

How would you design this system?

What relationships would the classes have, if any?

Would there be one overarching superclass, or many?

How do we *represent* our design?

Introduction to UML

Keyword

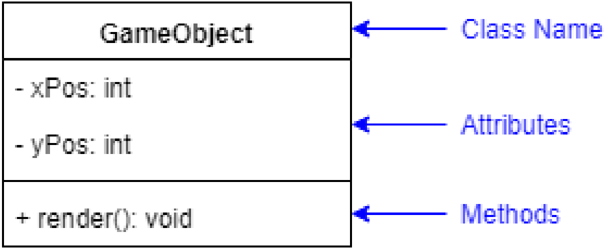
Unified Modeling Language (UML): a graphical modelling language that can be used to represent object oriented analysis, design and implementation.

What you are learning is class modelling; UML is only one notation/tool!

Representing a Class in UML

How would you represent the superclass of the game, `GameObject`?

- How would you represent the class?
- How would you represent its attributes?
- How would you represent it methods?



GameObject

- xPos: int

- yPos: int

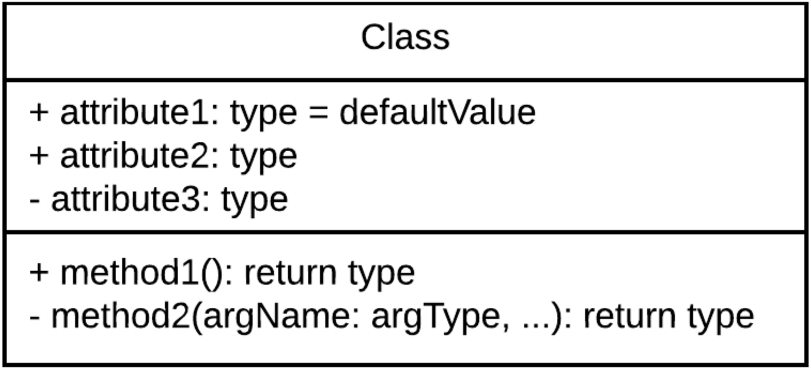
+ render(): void

Class Name

Attributes

Methods

Representing A Class



Class

+ attribute1: type = defaultValue
+ attribute2: type
- attribute3: type

+ method1(): return type
- method2(argName: argType, ...): return type

Representing Class Attributes

Components of an attribute:

- Name (e.g. xPos)
- Data Type (e.g. `: int`)
- Initial Value (e.g. `= 0`)
- Privacy (e.g. `+`)
- Multiplicity

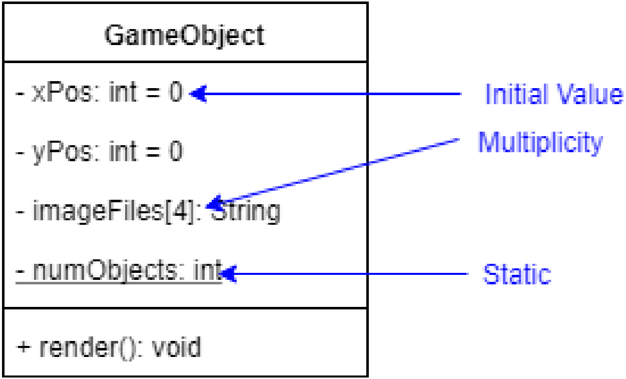
Finite	[10] (array)
Range (Known)	[1..10] (array or list, etc.)
Range (Unknown)	[1..*] (list, etc.)
Range (Zero or More)	[*] (list, etc.)

- Static (e.g. `numMethodCalls : int`)

Representing Class Attributes

Representing access control:

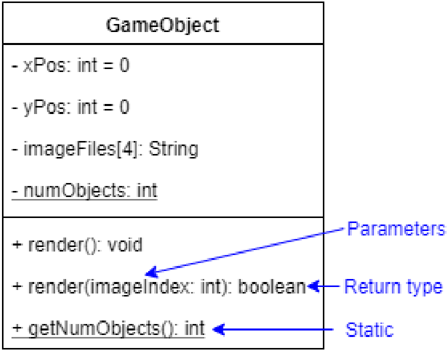
+	Public
∼	Package
#	Protected
−	Private



Methods

Components of a method:

- Name (e.g. render())
- Privacy (e.g. +)
- Return type (e.g. : `void`)
- Parameters (e.g. `name : String`)



GameObject

- xPos: int = 0

- yPos: int = 0

- imageFiles[4]: String

- numObjects: int

+ render(): void

+ render(imageIndex: int): boolean

+ getNumObjects(): int

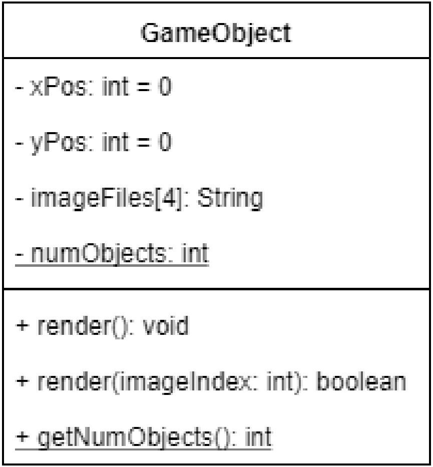
Parameters

Return type

Static

Assess Yourself

Implement the `GameObject` class



GameObject

- xPos: int = 0

- yPos: int = 0

- imageFiles[4]: String

- numObjects: int

+ render(): void

+ render(imageIndex: int): boolean

+ getNumObjects(): int

Assess Yourself

```java
public class GameObject {

    private int xPos = 0;
    private int yPos = 0;
    private String[] imageFiles;
    private static int numObjects;

    public void render() {
        //block of code to execute;
    }

    public boolean render(int imageIndex) {
        //block of code to execute
    }

    public static int getNumObjects() {
        // block of code to execute;
    }
}
```

Representing Class Relationships

Classes may relate to other classes through different types of relationships.

Following are the four common types of relationships between classes:

- Association
- Generalization (Inheritance)
- Realization (Interfaces)
- Dependency

Let's look at how to represent these relationships in UML.

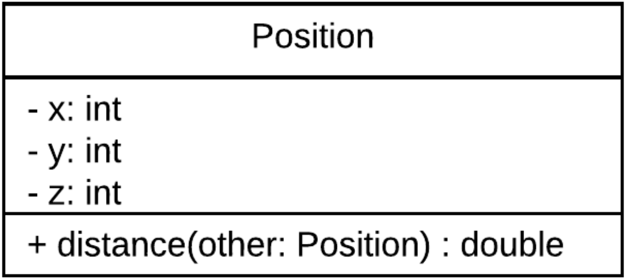
Association

- Represents a **has-a** (containment) relationship between **objects**.
- Allows objects to **delegate** tasks to other objects.
- Shown by a solid line between classes.

During class refinement the design team decides to represent the position of the `GameObject` with a `Position` class (instead of coordinates).

- Is this a good decision?
- How might the class be represented?
- How does it change our existing design?

Association

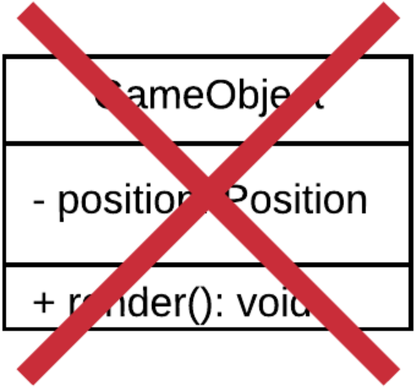


Position

- x: int
- y: int
- z: int

+ distance(other: Position) : double

Association



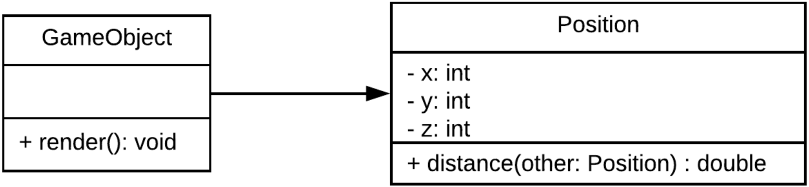
GameObject

- position: Position

+ render(): void

Association

When one class is *contained* by another, we always use an **association**.



Keyword

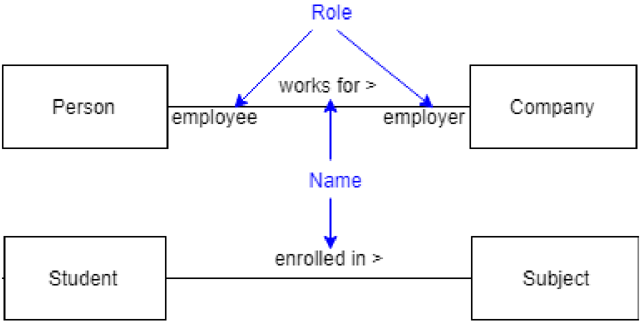
Association: A link indicating one class contains an *attribute* that is itself a class. Does **not** mean one class "uses" another (in a method, or otherwise).

Properties of Association

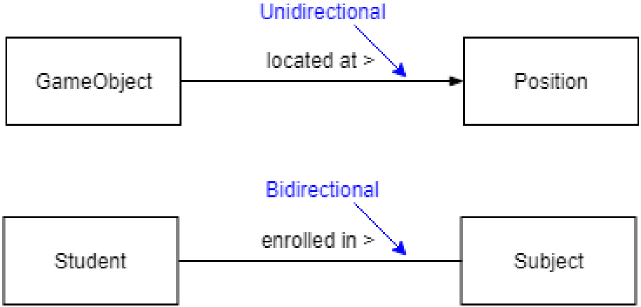
Association links can represent more information:

- Name
- Role labels
- Directionality
  - Unidirectional (hierarchical, ownership)
  - Bidirectional (co-operation)
- Multiplicity
- Type
  - Plain association
  - Aggregation
  - Composition

Properties of Association - Name and Role



Properties of Association - Direction



Properties of Association - Multiplicity

Keyword

Multiplicity: specifies the **number of links** that can exist between **instances (objects)** of the associated classes.

- Indicates how many objects of one class relate to one object of another class
- Can be a single number or a range of numbers
- We can add multiplicity on either end of class relationship by simply indicating it next to the class

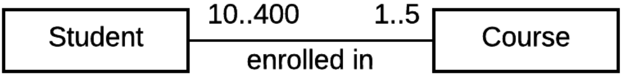
Properties of Association - Multiplicity

- One class can be related to another in the following ways:
  - One-to-one
  - One-to-many
  - One-to-one or more
  - One-to-zero or one
  - One-to-a bounded interval (one-to-two through twenty)
  - One-to-exactly n
  - One-to-a set of choices (one-to-five or eight)
- Multiplicity can be expressed as:
  - Exactly one - 1
  - Zero or one - 0..1
  - Many - 0..* or *
  - One or more - 1..*
  - Exact Number - e.g. 3..4 or 6
  - Or a complex relationship e.g. 0..1, 3..4, 6..* would mean any number of objects other than 2 or 5

Assess Yourself

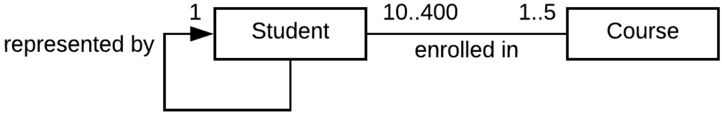
Create a UML representation for the following scenario:

- A `Student` can take up to five `Courses`
- A `Student` has to be enrolled in at least one `Course`
- A `Course` can contain up to 400 `Students`
- A `Course` should have at least 10 `Students`



Self-Association

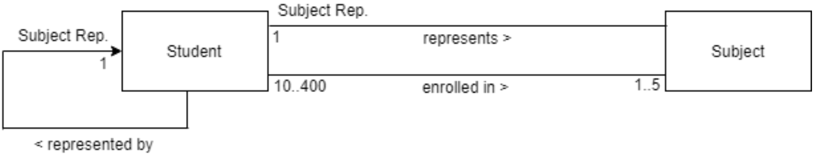
Each `Student` has a student representative they can contact, who is also a student



Multiple Associations

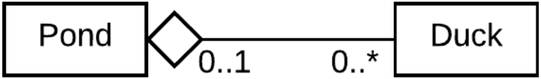
Does the following class relationship enforce a single student rep per-subject?

How about a single student rep per-subject, per-student?



Association - Type (Aggregation)

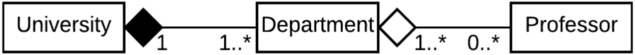
Different form of association, where one class "has" another class, but both exist independently



- If a `GameObject` is destroyed, the `Position` object disappears; dependence
- If the `Pond` object is destroyed, the `Duck` lives on; independence
- Makes sense; a `Duck` can find another `Pond`

Association - Type (Composition)

Different form of association, indicating one class cannot exist without the other; in other words, existing on its own doesn't make sense



- A `Department` is entirely dependent on a `University` to exist
- If the `University` disappears, it makes no sense for a `Department` to exist
- But a `Professor` is just a person; they can find another `University`!

Assess Yourself

What comes next in the design? So far we have:

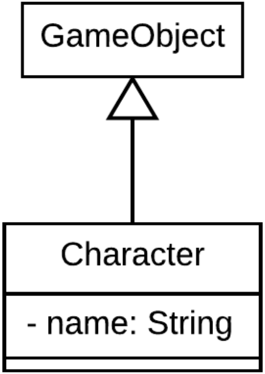
- `GameObject`
- `Position`

Assess Yourself

“This game involves a number of characters, like the player, their allies, and their enemies.”

Sounds like **inheritance**!

Generalization - Inheritance

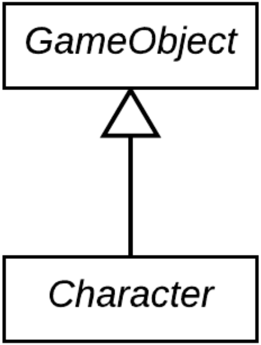


Inheritance

But wait... What happens if I make a `GameObject`? Does that make sense?

Nope!

Abstract Classes



Italicised *methods* or *classes* are **abstract**.

Assess Yourself

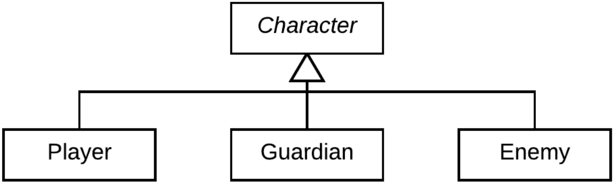
What next? Now we have:

- `GameObject`
- `Position`
- `Character`, which inherits from `GameObject`

How could we add *behaviour*?

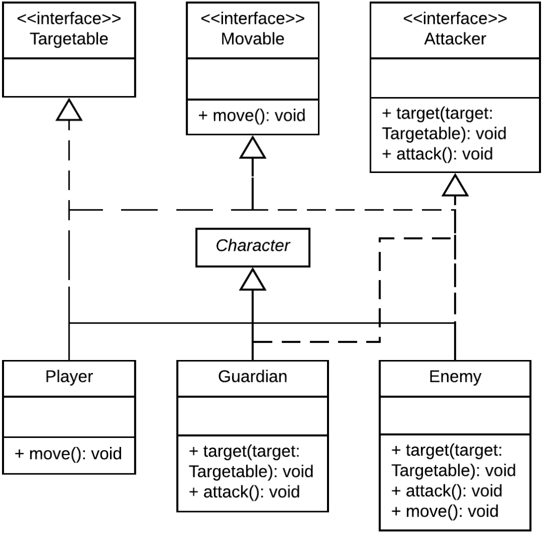
Assess Yourself

Some design help...



Only the `Player` and `Enemy` can move. The `Guardian` and the `Enemy` can attack each other, and the `Enemy` can also attack the `Player`. Both `Characters` can also target the player.

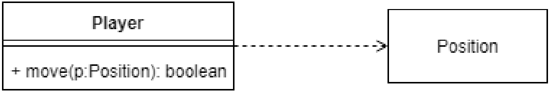
Realization - Interfaces



Dependency

Dependency represents a weak relationship between classes; dependency implies that a change to one class may impact the other class. It is represented by a dotted arrow.

For example, when a method in a class has another class as one of the input parameters, this results in a dependency relationship.

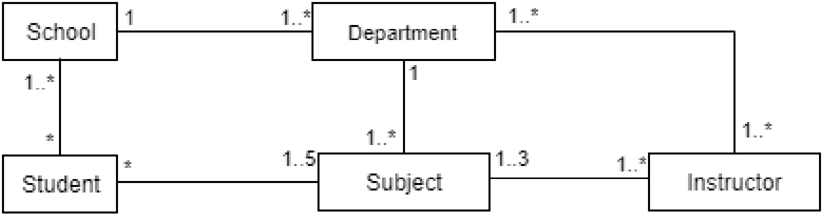


Assess Yourself

Construct a UML diagram for the following description:

- A school has one or more departments.
- A department offers one or more subjects.
- A particular subject will be offered by only one department.
- A department has instructors and instructors can work for one or more departments.
- A student can enrol in up to 5 subjects in a school.
- Instructors can teach up to 3 subjects.
- The same subject can be taught by different instructors.
- Students can be enrolled in more than one school.

Assess Yourself



Assess Yourself

How could your Project 1 submission be better designed?

Project answers don't come that easily...

UML Drawing Tools

- draw.io
- LucidChart (Sign up with your student email for full functionality)
- StarUML
- Eclipse/IDE with plugins (useful for converting UML to code instantly)
- And many more...

draw.io is apparently the tool of choice for SWEN30006, but the choice is yours.

Metrics

A world class chef is working to develop a robotic assistant, and they've hired you to build a simulated kitchen to test it out.

The robot needs to be able to:

- Use normal human tools and utensils (oven, fork)
- Open things (cupboards and containers)
- Prepare ingredients (cutting, dicing, boiling)
- Be operated by a human (for safety reasons)

Create a class diagram for this scenario, including any inheritance or realisation (interface) relationships.