Software Specification & Design

Lecture 1

Instructor

- Keeratipong Ukachoke
- Contact me at kee@glazziq.com

Class Structure

- The class is Software Specification and Design
- Focus on specification 10% design 90%
  (Last year 30% : 70%)
- Reason?

Class Structure

- This is a template for our class
    - Lecture - 85 minutes
    - Break - 10 minutes
    - Small projects - 85 minutes

Books

- Design Patterns - Elements of reusable Object-Oriented Software
  (ISBN: 0201633612)

- Applying UML and Patterns
  (ISBN: 0130925691)

Grading

- 40% Projects and Homeworks
- 30% Midterm
- 30% Final
- Bonus 10% for Participation

Grading

- 90% - 100% : A
- 85% - 89% : B+
- 80% - 84% : B
- 70% - 79% : C
- 60% - 69% : D

Policies

- Only one rule. No cheaters.

Questions

Today topics

- Lecture - Introduction
  - software process
  - software specification
  - software design
- Lecture - Software Design Patterns
  - Singleton
  - Observer
  - Strategy
- In class project

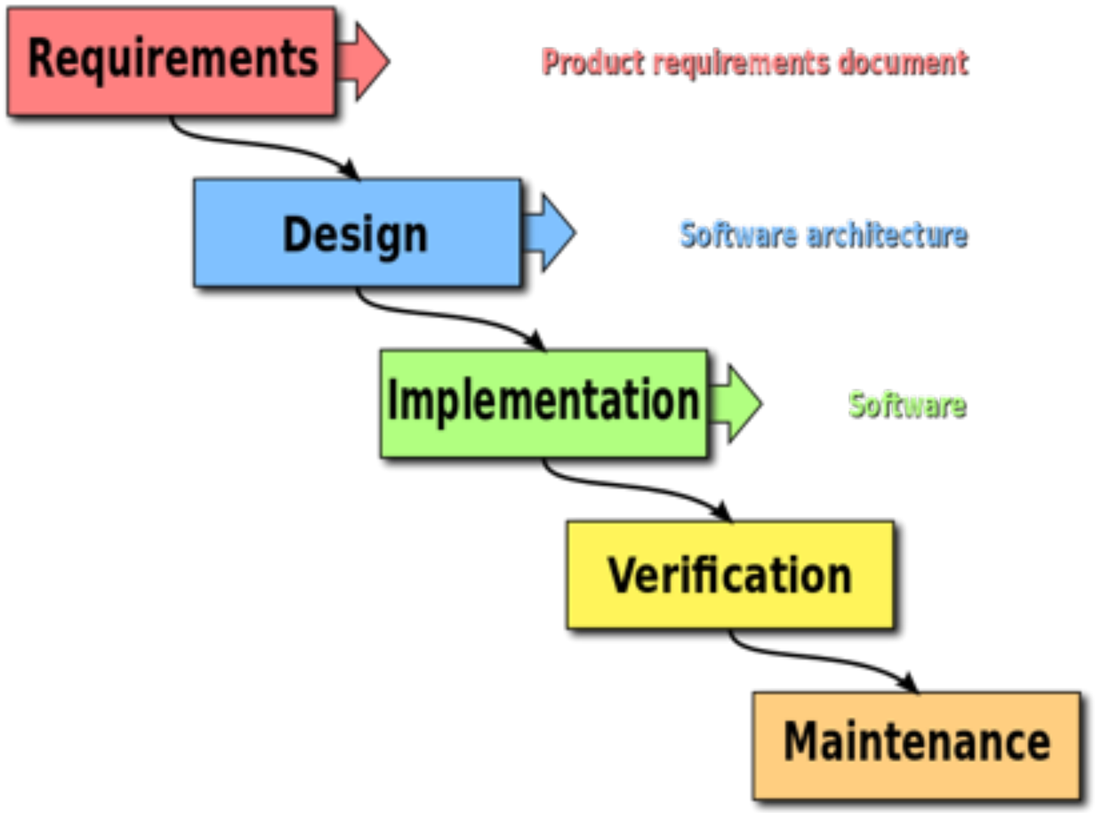
Intro. Software Process

- Aka Software development process
- What is process? Do we need it?
- What are examples of popular processes?

Waterfall Model

- Sequential
- Construction industries
- Define most of requirements at the beginning
- Advantages and disadvantages?

Waterfall Model



http://en.wikipedia.org/wiki/Waterfall_model

Waterfall Model

- Problems
  - Users don’t know what they actually want.
  - Too late to go back
  - Requirements change ~ 25% - 50%
  - The bigger project, the more change

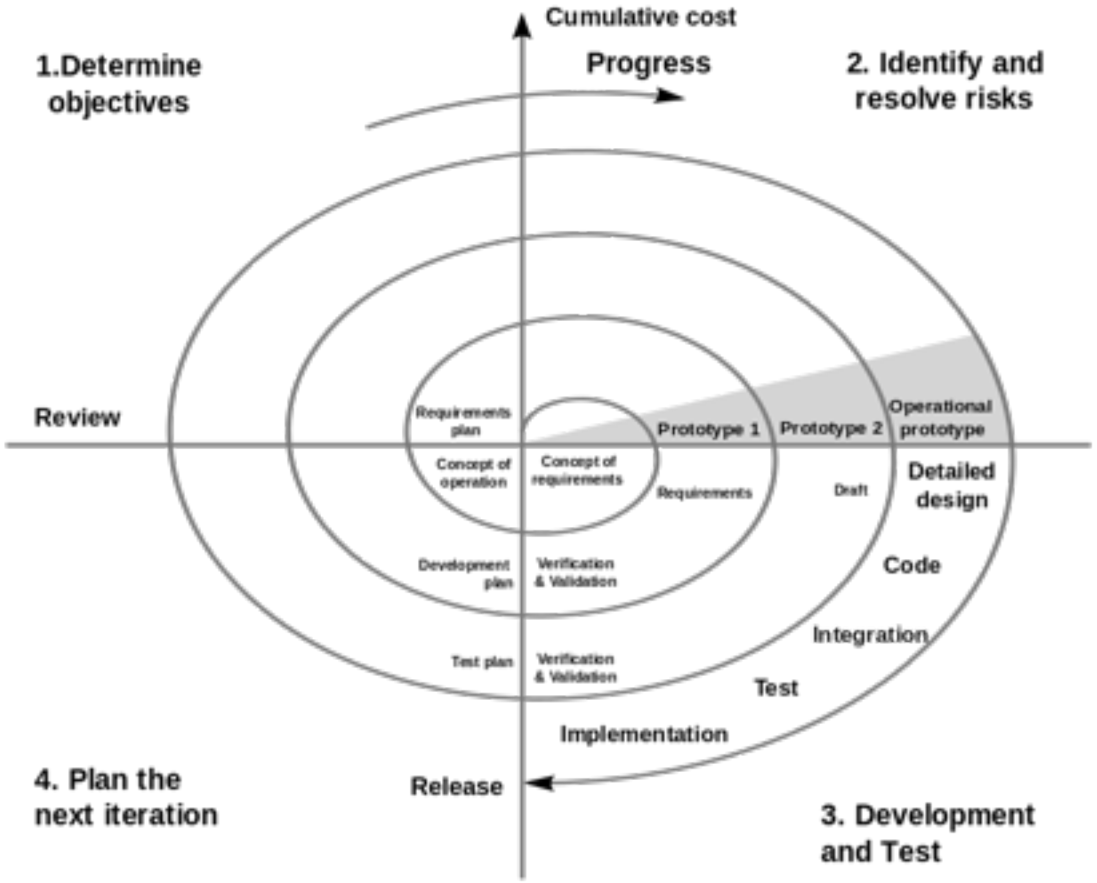
Waterfall Model

- So should we use it?
    - If the requirements are well known, clear and fixed. (Not likely, but possible)
    - You have enough expertise
    - Fix contracts/deliver date/budget

Iterative and Evolutionary Development

- Iterations
- Each can be thought as a mini project
- The system grows over time
- Iterative and Incremental development
  (The names gave different meanings for different people)

Iterative and Evolutionary Development



https://en.wikipedia.org/wiki/File:Spiral_model_(Boehm,_1988).svg

Iterative and Evolutionary Development

- Nature
  - Embrace change
  - Early iterations are far from the true path of the system
  - In late iterations, significant change is rare
    (But can occur)

Iterative and Evolutionary Development

- Benefits
    - Less project failure
    - Early visible progress
    - Early feedback
    - Reduce complexity

Intro. Software Specification

- What is software specification?
- SRS
- Functional vs Non-function
- Use cases

Use Cases

- Quick review, from Dicegame

- Use case [**Play a dice game]**

- A player requests to roll two dice. System presents results. If the sum of faces is 7, player wins, otherwise, player loses.

What are use cases

- Text stories
- Discover and record requirements
- 3 types, brief, casual, fully dressed

Brief use case example

- POS - Process Sale :
  - A customer arrives at a checkout with items to purchase.
  - The cashier uses the POS system to record each purchased item.
  - The system presents a running total and line-item details.
  - The customer enters payment information
  - The system validates and records.
  - The system updates inventory.
  - The customer receives a recipe from the system and then leaves with the item.

Use case - Actors and Scenarios

- Actors
    - A sale person
    - A customer
    - Computer system
    - An organization
- Scenario
    - The scenario of successfully purchasing items with cash
    - The scenario of failing to purchase because of a credit payment denial

Why use cases?

- Simple for normal people (non-tech)
- Have clear goal
- Can scale up and down in term of complexity
- Can be used as a central mechanism in requirements management

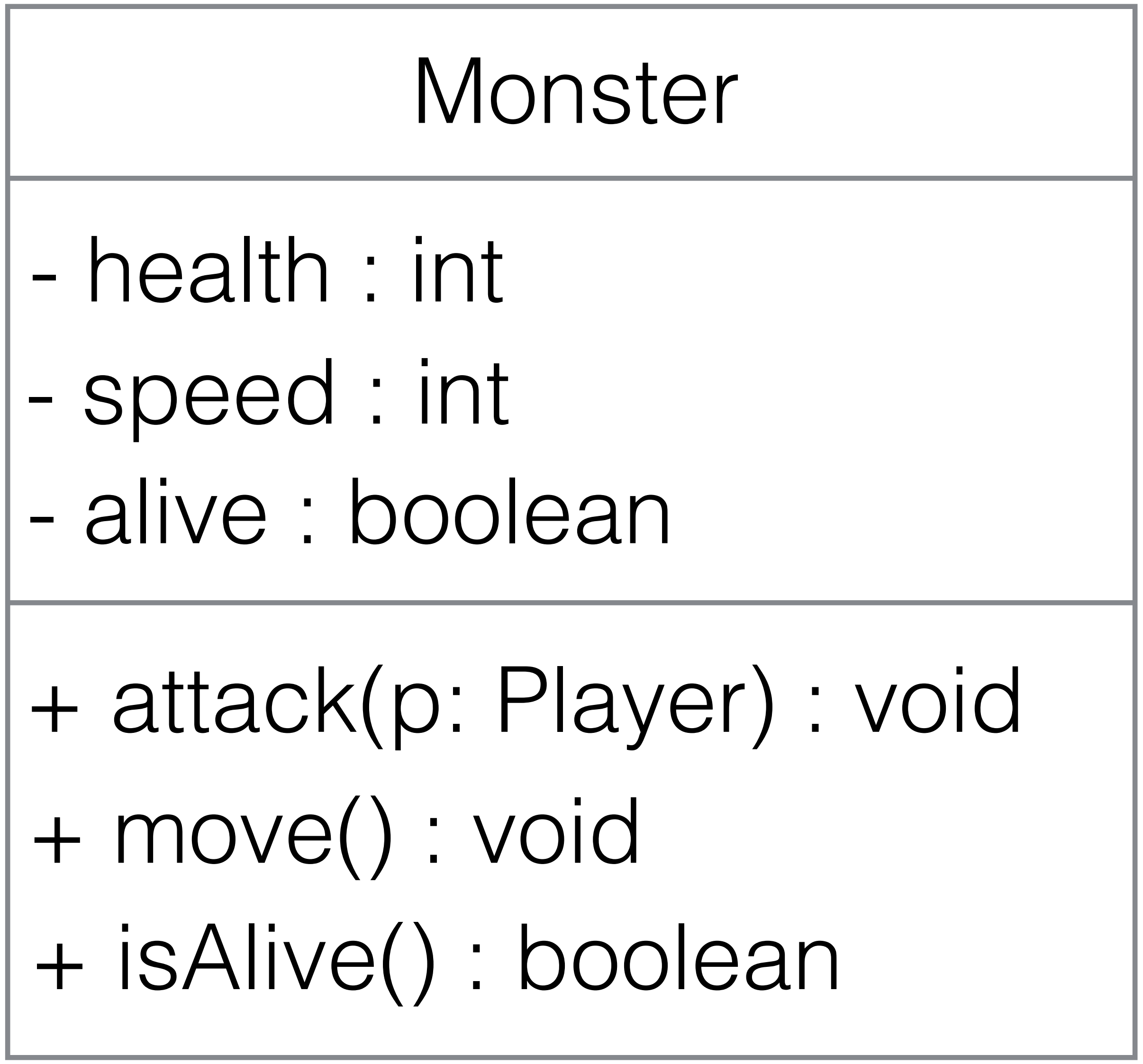
Intro. Software Design

- There are many layers in software design
- From architectural level to implementation level
- Let’s see examples

Java & OOP Review

- Before moving on, we will review about
  - Different types of classes in Java
  - Objects and there default methods
  - Inheritance
  - Interface
  - Common classes in Java such as List, Set, Map

UML Class Diagram



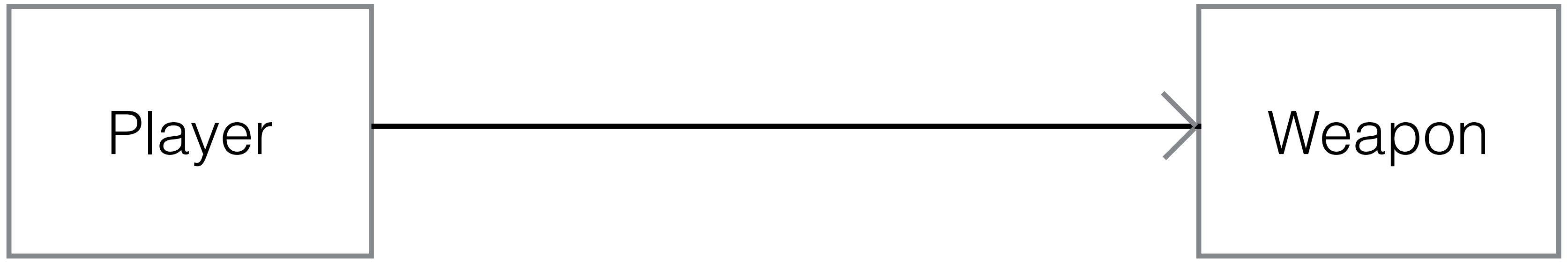
Monster

- health : int
- speed : int
- alive : boolean

+ attack(p: Player) : void
+ move() : void
+ isAlive() : boolean

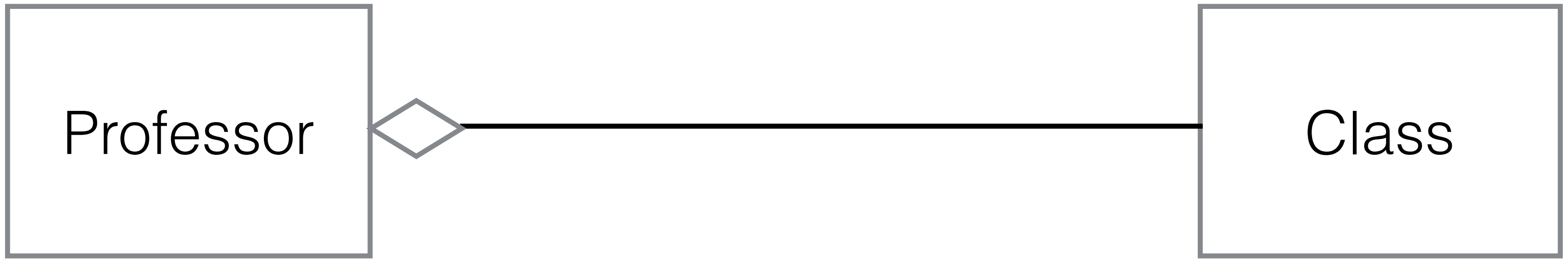
UML Class Diagram

- Direct Association



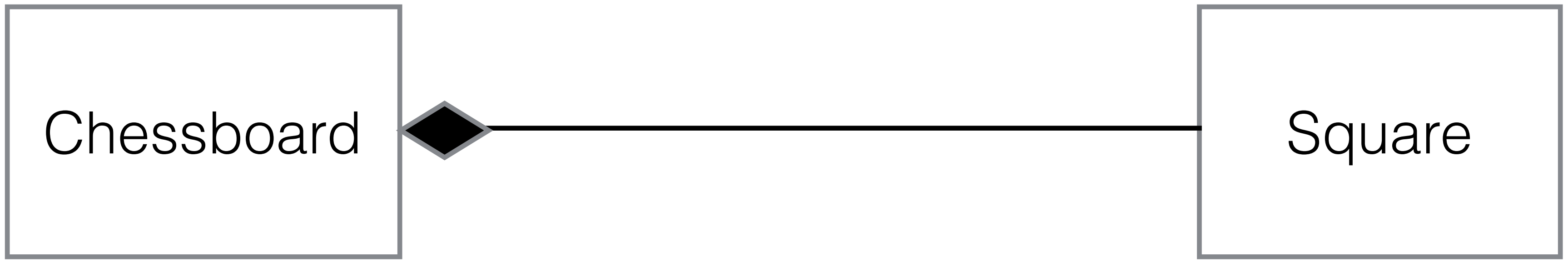
UML Class Diagram

- Aggregation



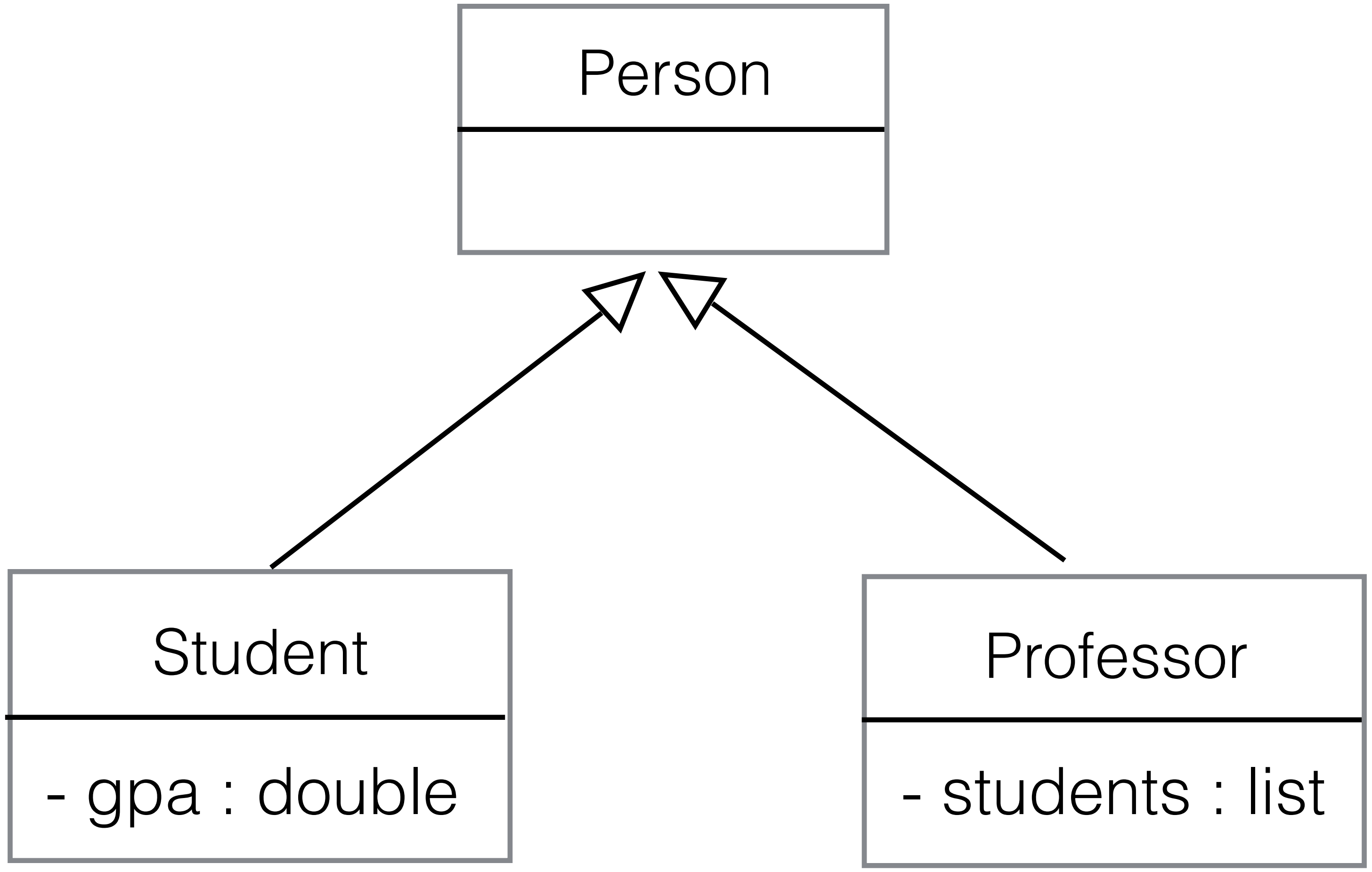
UML Class Diagram

- Composition



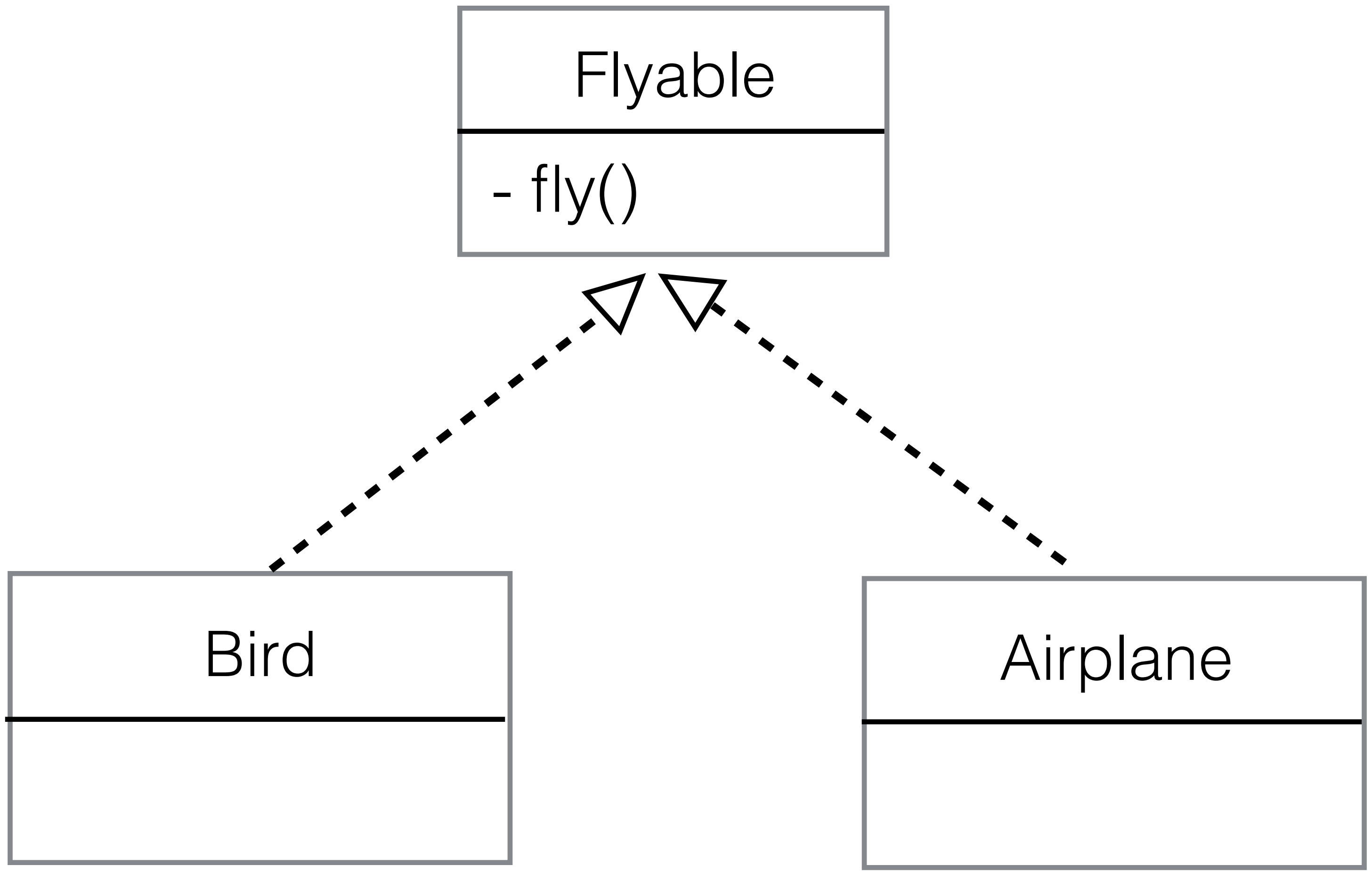
UML Class Diagram

- Generalization



UML Class Diagram

- Realization



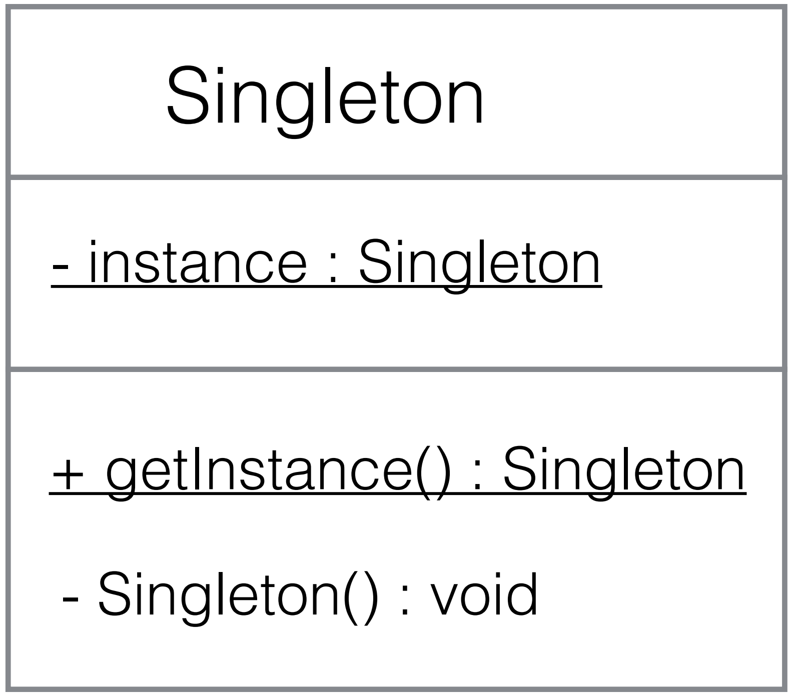
Intro. Design Pattern

- Reusable solution for common problem in software design
- Language independent
- Can be classified in to many categories

Our first pattern today - Singleton

- The problem
    - Exactly one object is needed
    - How to be sure that there will always be only one instance of a class?

Singleton



Singleton

- instance : Singleton

+ getInstance() : Singleton

- Singleton() : void