Menti Q&A QR

Tools for Software Design

Blair Archibald

Admin

- Lab 0 Solution is now on Moodle
- Labs now features attendance monitoring
  - Make sure to get signed in by a Lab Assistant
  - Don't worry if you need to miss a single lab
    - But *continuous* absence might affect Visa's etc
- Q&A From Last weeks lecture inside previous slides

Recap

You should know:

- Challenges of SE:
  - Scale (handled by abstraction into components)
  - Change (handled by careful design of components)
- Coupling (between components) and Cohesion (within components)
- Given code you should be able:
  - Identify possible design issues
  - Suggest and implement improvements to the design
  - Utilise "advanced" OOP features (interfaces/abstract classes/visibility modifiers...)

Overview

We will explore:

- The Unified Modelling Language (UML)
- UML Class Diagrams:
  - Designing classes
  - Designing their relationships
- UML Sequence Diagrams
  - How classes interact

UML: Unified Modelling Language

- Historically a large push to use model driven engineering for SE:
  - Use notations to design the system:
    - State changes inside objects
    - Message exchange diagrams
    - Use-case models
  - Analyse these before deployment, e.g. to check it meets requirements
  - Implement: potentially letting the model write code
- Contrast with other engineering models: architectural drawings, stress simulations, construction blueprints, isometric plans, etc

UML: Unified Modelling Language

- UML took lots of different modelling tools and tried to combine the best bits
  - Became a standard for modelling
  - Around 14 diagram/Modelling tool types in total
    - Class and Sequence Diagrams most common
- We use UML to mean UML2 here
- I'll draw diagrams with PlantUML[1] which is (mostly) standard UML
  - Text based format that generates diagrams
  - Useful tool to know for project reports etc

[1] https://plantuml.com/

UML: Unified Modelling Language

UML Models are one way to **document** a system

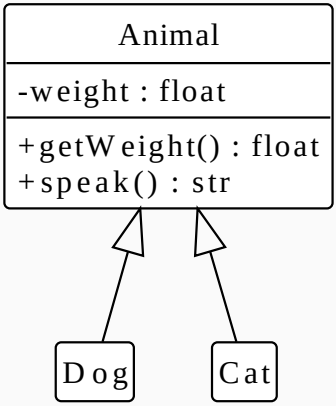
Caveat

- UML has fallen out of favour and is not used as much in practice as a key tool
  - Partly because the "design-up-front" is being replaced by Agile
- But:
  - Having a system model is still useful: you don't build a house without a blueprint
  - Used in almost all SE papers/books
  - Most SE know them, so it's a common design language for teams
  - Graphical modelling tools are making a comeback ("nocode")

See: https://buttondown.email/hillelwayne/archive/why-uml-really-died/

Class Diagrams (Structural)

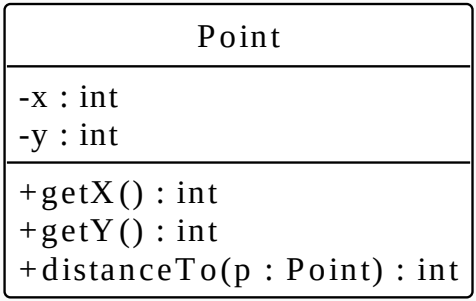
- Class diagrams show
  - Classes that make up the system incl key variables/methods
  - Relationships between the classes (if it's messy you might have bad coupling!)



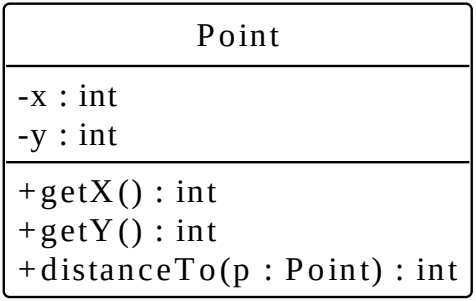
Single Classes

- Classes have three elements:
  - A name
  - Instance Variables
  - Methods

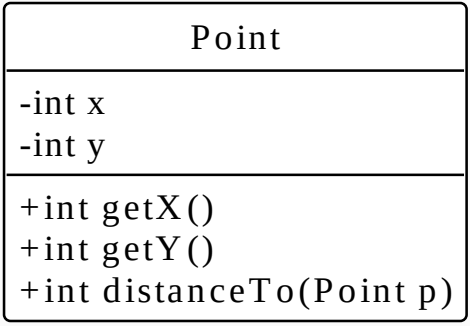
Visibility Symbol	Meaning
-	Private (class only)
~	Package Private
#	Protected (class + children)
+	Public (anyone)



Specifying Methods/Data



Technically should use the `x : int` type notation

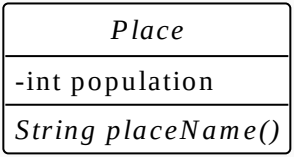


No one will blame you for this style though!

Programmers (including you!) should be able to read both notations

Abstract Classes in UML

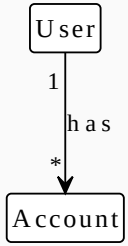
- Same as a class, but with the name in *italics*
- Abstract methods might be italics
  - But not really defined in the spec!

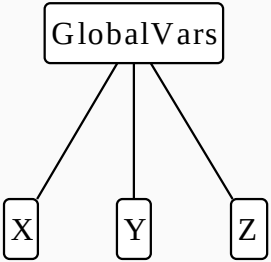


Class Relationships

Reminder: Relationships/Interaction between components is the key idea

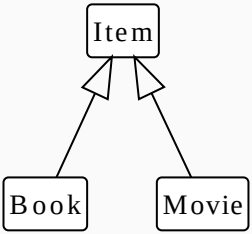
Associations





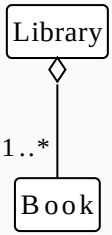
- Denotes some relationship between classes
  - Relationship can be broad: “uses” “depends on” “owns” “has”
  - Can be named (for clarity)
  - *Can* be directed (arrow heads are **solid**)
  - Might be given cardinality labels (0..1, 1..max_accounts)

Extensions (Inheritance)



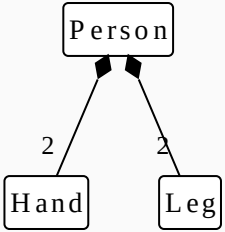
- Clear arrowheads means “is a”
  - I.e. One class **inherits** from another
  - Arrow points to the parent
    - Book *is a* Item

Aggregation (Built From)



- Clear diamond means “part of”
  - A Book *is part* of a library component
  - Alternatively: a Library *has a* (collection of) book
  - Often used for Collections of things
- Weak dependency:
  - A book is still a book without a library
  - Contrast to an account without a user for example

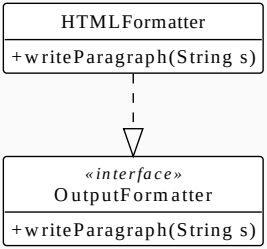
Composition (Strongly Built From)

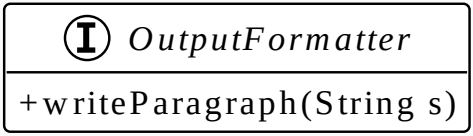


- Solid diamond means “is part of and depends on”
- Strong dependency:
  - If we remove the person, we also remove the hands/legs

Interfaces in UML

- Like a class, but has the text `<<interface>>`
- Inheriting from an interface uses inheritance link with dashed lines.
  - "Realising an interface"





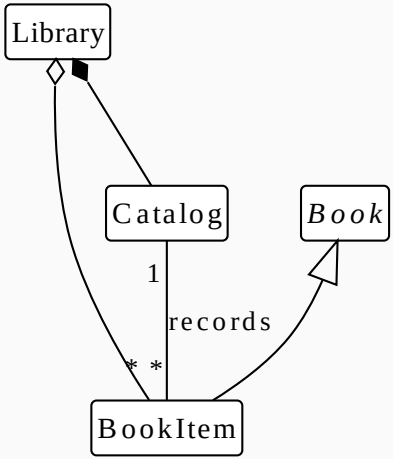
Tools like plantuml might custom symbols for interfaces

Designing at different levels of abstraction

- You might design with less information at different times:
  - Just classes and relationships
  - Specification of interfaces
  - Implementation specific info

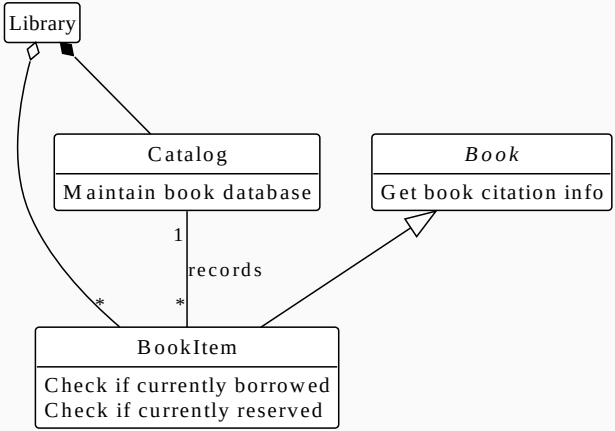
Designing at different levels of abstraction

High-Level Domain model



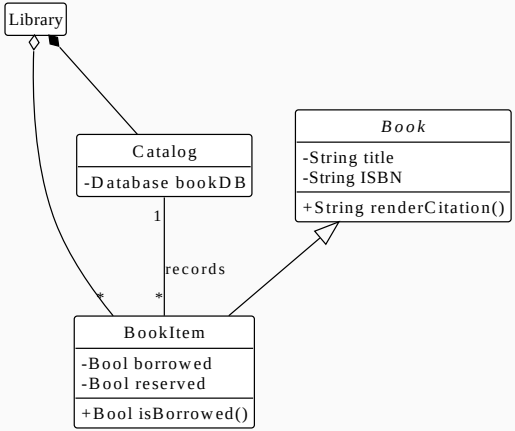
Designing at different levels of abstraction

Domain model with Feature Info



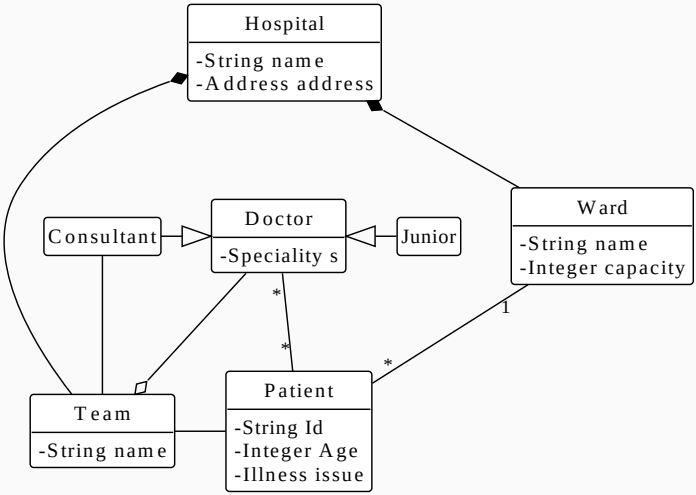
Designing at different levels of abstraction

Domain model with Implementation Info



Lecture Q&A: menti.com/alo6c2asngoh

Example: Hospital



Class Diagram Recap

- Class Diagrams show:
  - Elements of a class (the data + API)
  - Relationships between classes
    - Associations
    - Extensions
    - Aggregations/Compositions
- Use to design, "did we capture relevant entities?"
- Use to check designs: e.g. Lots of relations $\implies$ lots of coupling

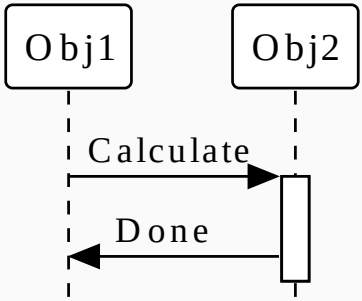
Sequence Diagrams

- Class diagrams are static: tell you structure
- Sequence diagrams document temporal/dynamic relations between objects
  - Describe protocols (Order methods are called in)

UML Sequence Diagrams

- Classes/Objects/Components go along the top
- Time runs *down* the diagram
  - Solid rectangles means "doing work"
- *Control* is passed using an arrow
  - A method call
- Usually implicit that the left-most object starts with control

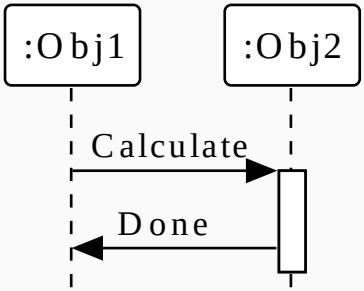
- Useful for complex protocols with lots of objects



Aside: Classes Vs Objects

- In Sequence Diagrams *technically* objects communicate
  - Not classes (since classes are scheme so don't exist)

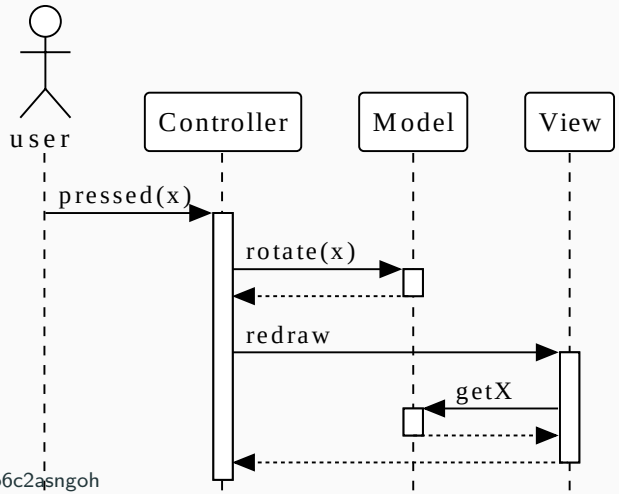
- Notation:
  - Person $\implies$ Class Person
  - o:Person $\implies$ Object called "o" of Type Class Person
  - :Person $\implies$ Any Object of Type Class Person



In practice almost no one makes the distinction!

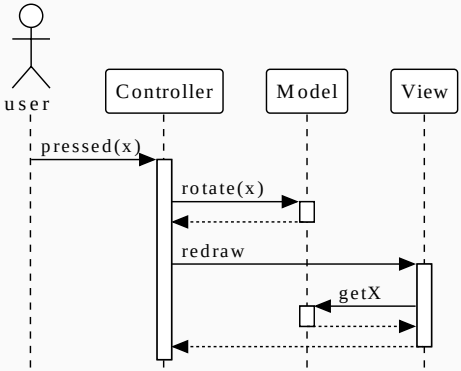
Example: Sequence Diagram for Model-View-Controller

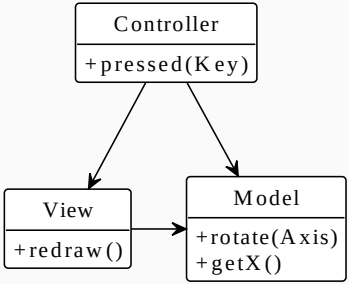
Use case: User presses X key to rotate some diagram around x-axis



Behaviour and Structure (Sequence + Class Diagrams)

Sequence diagrams can be reflected in Class Diagrams (and vice-versa)





Different diagrams to describe the **same** system: **control/behaviour** vs **structure**

Sequence Diagrams from User Stories

- Sequence diagrams might describe complex internal algs
  - How layouts work etc
- Can also describe how **user-stories** will be/are implemented
  - What objects are involved
  - How information flows between objects
- Can use this to help guide design
  - Comparing alternatives etc

Worked Example

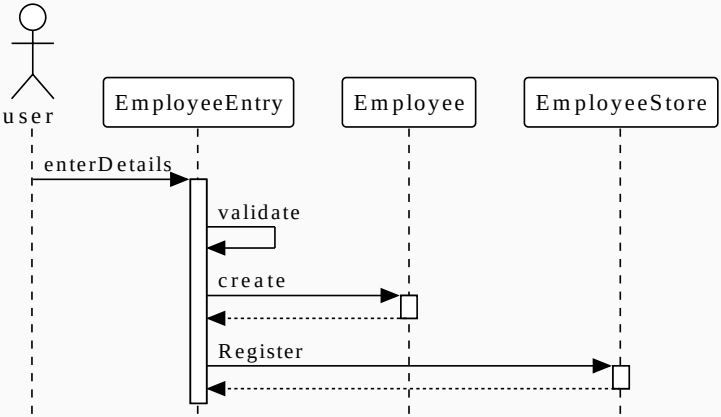
```
When a new employees details are entered they are validated, and
assuming valid, the employee should be added to the employee list.
```

Worked Example

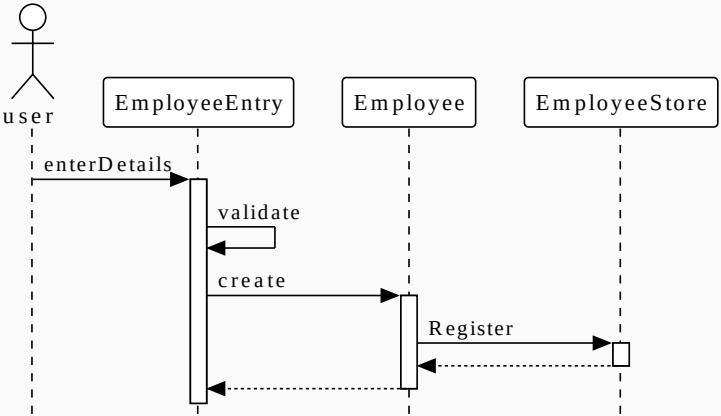
```
When a new employees details are entered they are validated, and
assuming valid, the employee should be added to the employee list.
```

- Step 1: Identify participants
  - User
  - Employee
  - EmployeeEntry
  - EmployeeStore
- Step 2: Identify information/control flow
  - User: provides an entry
  - Employee: created from a (valid) entry
  - EmployeeEntry: filled with user data
  - EmployeeStore: stores a ready created employee

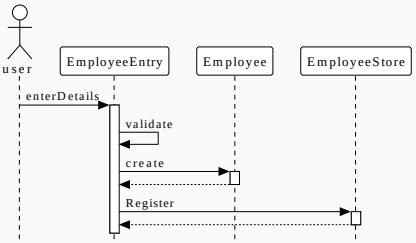
Worked Example: Possible Designs

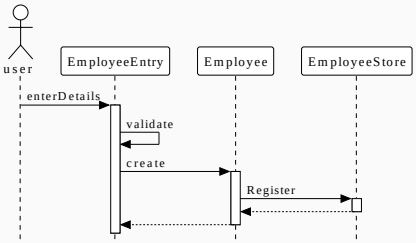


Worked Example: Possible Designs



Worked Example: Possible Designs

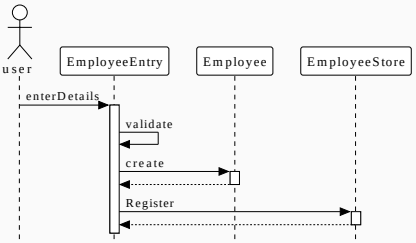


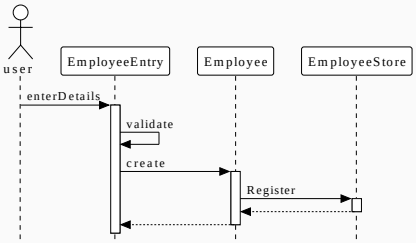


Can then ask: which design is *better*[2]?

[2] "Better" is subjective, but we can decide which *we* are happier with

Worked Example: Possible Designs





Can then ask: which design is *better*[2]?

- Second design implies `Employee` knows how it is stored
  - What if we want to change to a `AllUserStore` later?
- *May* want to move `validate` logic to validator class

[2] "Better" is subjective, but we can decide which *we* are happier with

Recap

- UML is a way to **document/model** software systems
- Class diagrams show *relationships* between classes
  - associations, inheritance, aggregation, composition
- Sequence diagrams show specific *interactions* between objects
- You should be able to:
  - Draw Class/Sequence Diagrams from code or a specification
  - Identify and explain errors in a diagram
  - Use diagrams to critically reflect on the design of a system

Useful Links

- https://www.uml-diagrams.org/class-reference.html