CS2103T - Week 4 - Topics

W4.1 - Design : Models

W4.2 - Class / Object Diagrams : Basics

W4.3 - Class Diagrams : Intermediate-Level

W4.4 - Java : JavaFX

W4.5 - Java : varargs

W4.6 - Code Quality : Naming

W4.7 - Code Reviews

W4.8 - Static Analysis

W4.9 - RCS : Managing Pull Requests

W4.10 - Automating the Build Process

W4.1 - Design : Models

W4.1a - Design - Modelling - Introduction - What

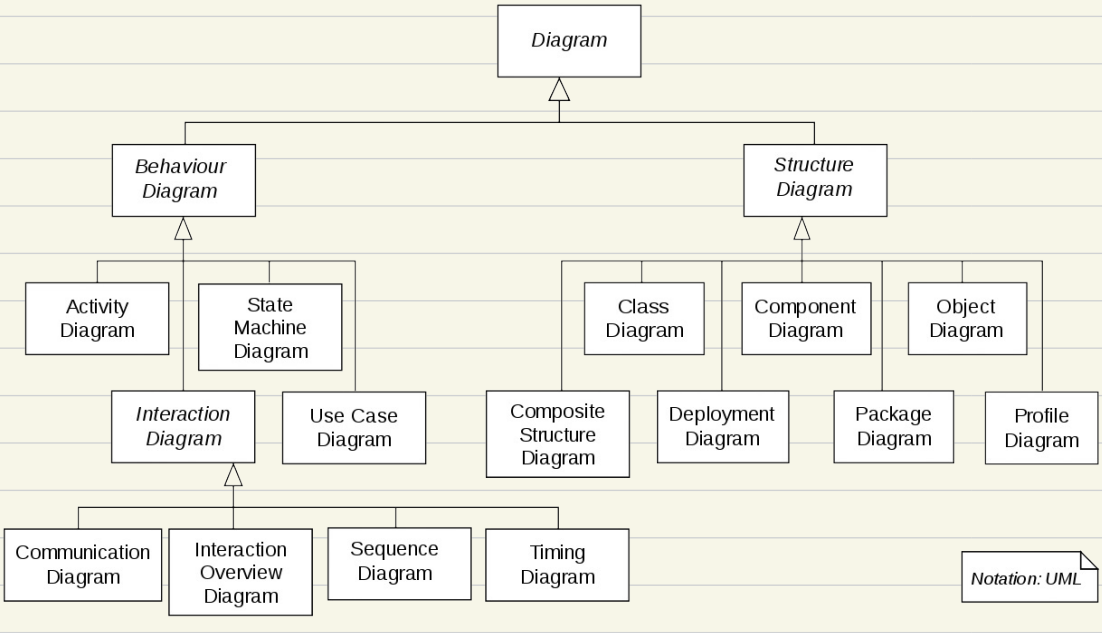
- **Model** : representation of something else
- Form of abstraction
- Captures only a selected aspect (structure)
- Multiple models of the same entity maybe needed to capture it fully

W4.1b - Design - Modelling - Introduction - How

- Usefulness of models in SE:
  1. To analyze a complex entity related to SE
  2. To communicate information among stakeholders
  3. As a blueprint for creating software

W4.1c - Design - Modelling - Introduction - UML models

Unified Modelling Language

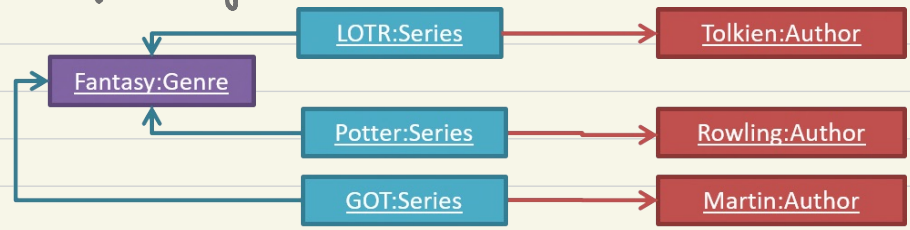


how to interpret

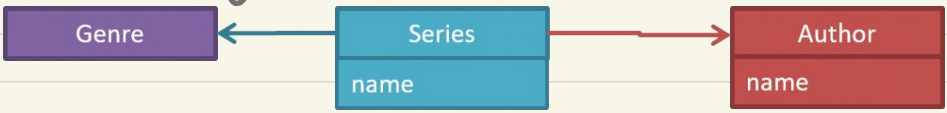
W4.2 - Class / Object Diagrams : Basics

W4.2a - Design - Modelling - Modelling Structure - OO structures

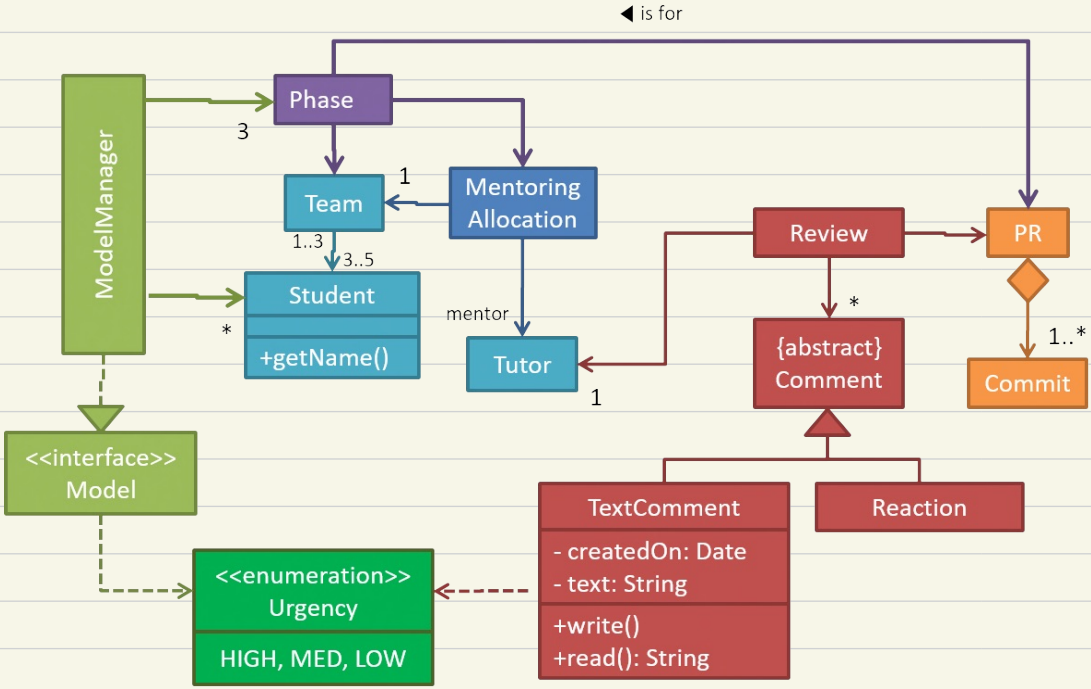
- UML object diagrams:



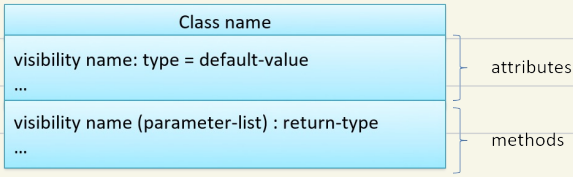
- UML class diagrams:



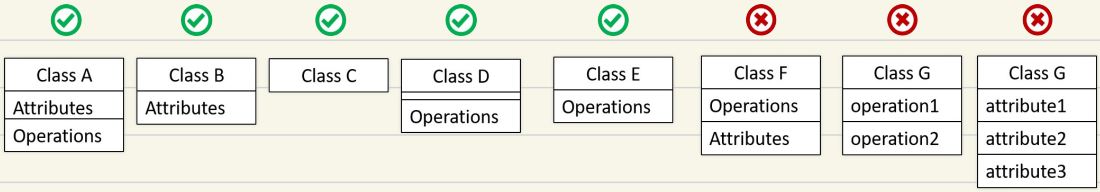
W4.2b - Design - Modelling - Modelling Structure - Class diagrams - basic



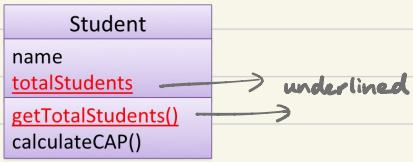
- Classes :



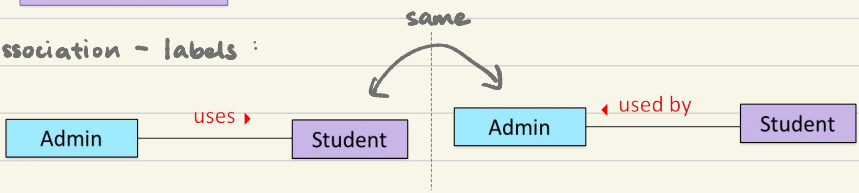
- + : public
- − : private
- # : protected
- ~ : package private



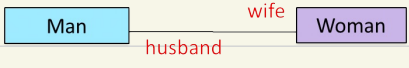
- Class-level members :



- Association - labels :

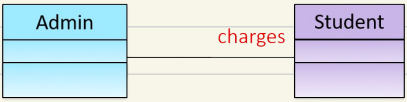


- Association - roles :




```java
class Man{
    Woman wife;
}

class Woman{
    Man husband;
}
```

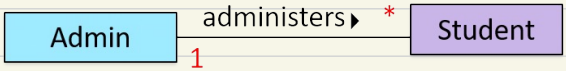



```java
class Admin{
    List<Student> charges;
}
```

aspect of an OOP solution that dictates how many objects take part in each association

- Associations - multiplicity

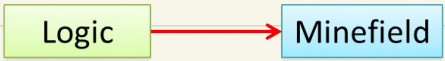
- `0..1` : *optional*, can be linked to 0 or 1 objects
- `1` : *compulsory*, must be linked to one object at all times.
- `*` : can be linked to 0 or more objects.
- `n..m` : the number of linked objects must be `n` to `m` inclusive



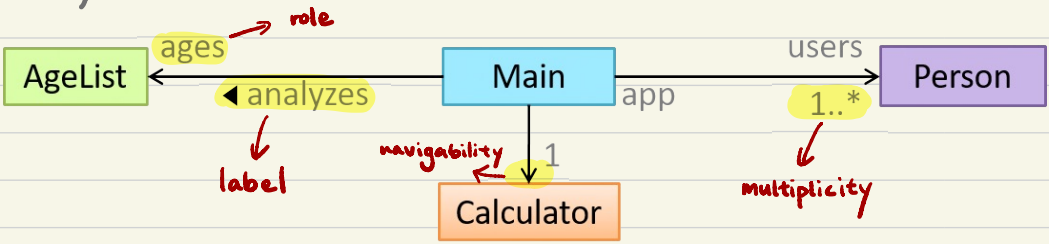
which class in the association knows about the other

- Associations - navigability

logic aware of Minefield

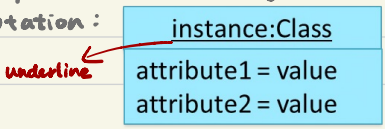


- Summary (example)



W4.2c - Design - Modelling - Modelling Structure - Object diagrams

- Complement class diagrams
- Notation :



- Notes :
  - No compartment for methods
  - Attributes compartment can be omitted
  - Object name can be omitted i.e. :ClassName

W4.2d - Tools - UML - Object versus class diagrams

- Class diagrams vs. object diagrams :

Compared to the notation for a class diagrams, object diagrams differ in the following ways:

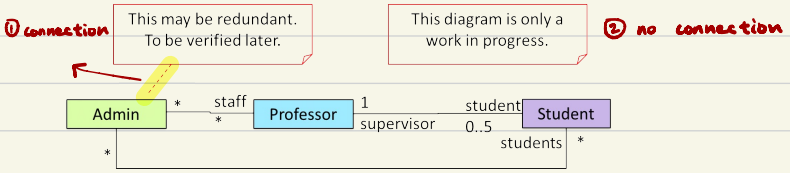
- Shows objects instead of classes:
  - Instance name may be shown
  - There is a `:` before the class name
  - Instance and class names are underlined
- Methods are omitted
- Multiplicities are omitted

Furthermore, **multiple object diagrams can correspond to a single class diagram**.

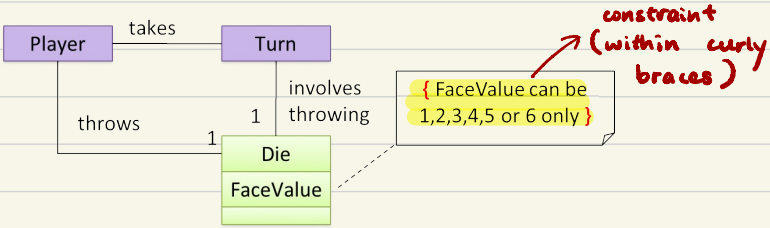
W4.3 - Class Diagrams: Intermediate-Level

W4.3a - Tools - UML - Notes

- UML notes:



W4.3b - Tools - UML - Constraints

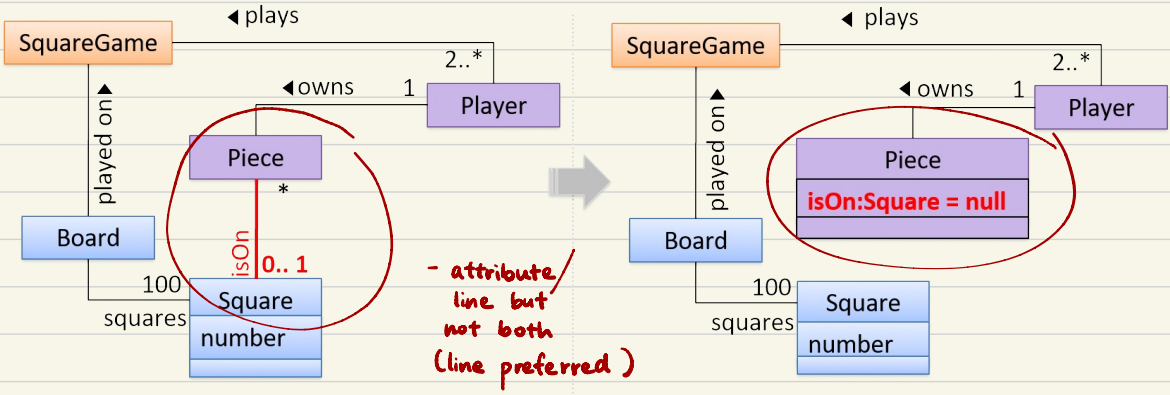


W4.3c - Tools - UML - Class Diagrams - Associations as attributes

An association can be shown as an attribute instead of a line.

Association multiplicities and the default value too can be shown as part of the attribute using the following notation. Both are optional.

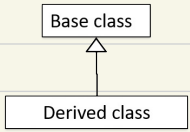
```
name: type [multiplicity] = default value
```



W4.3d - Design - Modelling - Modelling Structure - Class diagrams

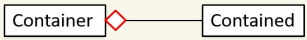
- Other types of associations : — intermediate

- inheritance



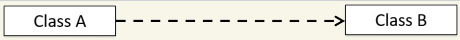
- compositions → strong whole-part relationship

- aggregations → weaker container-contained relationship



- dependencies → a need for one class to depend on another without having a direct association with it

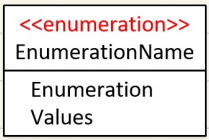
decreasing association strength



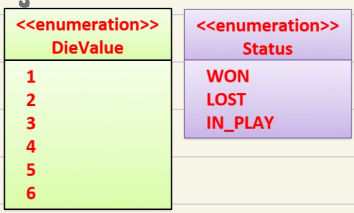
↳ A depends on B

- Other class-like entities :

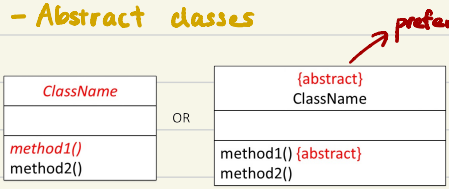
- Enumerations :



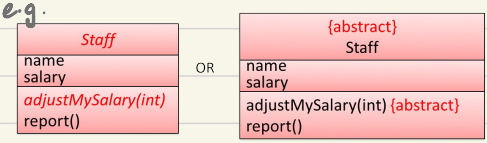
e.g.



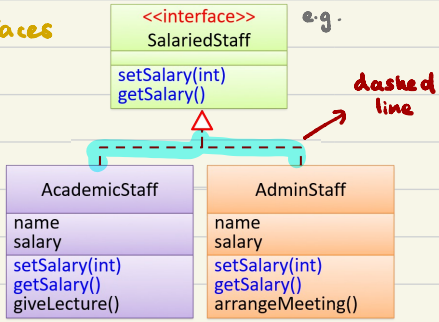
- Abstract classes



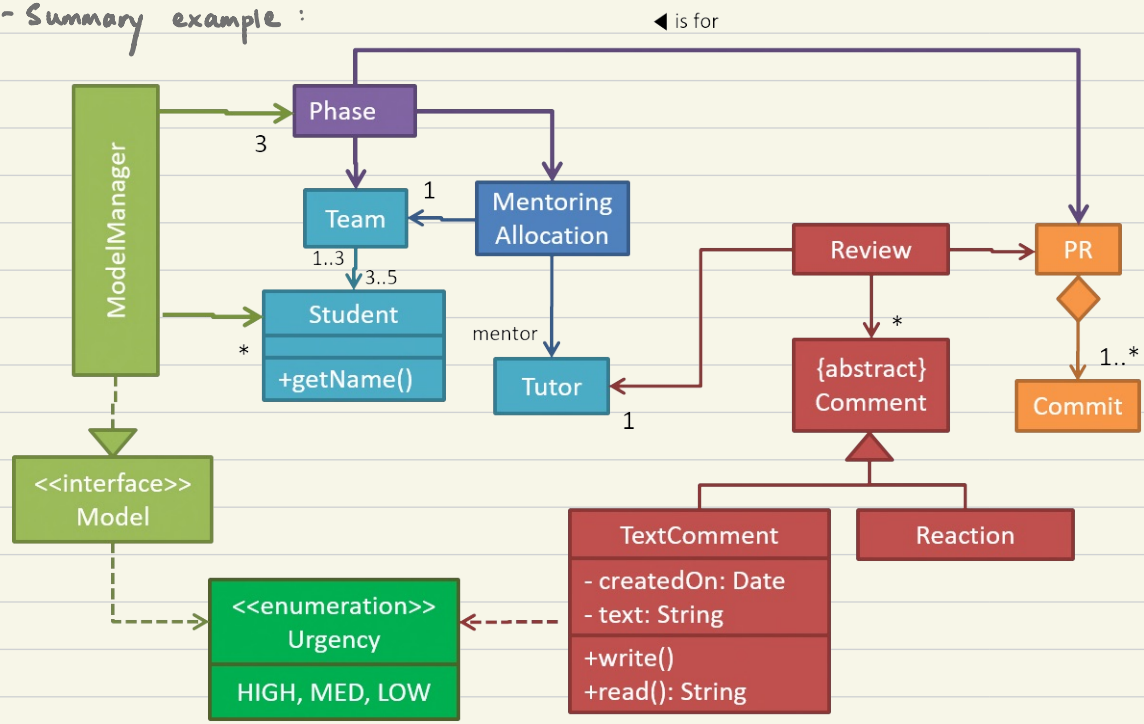
e.g.



- Interfaces

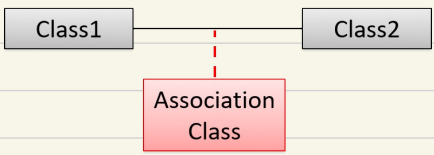


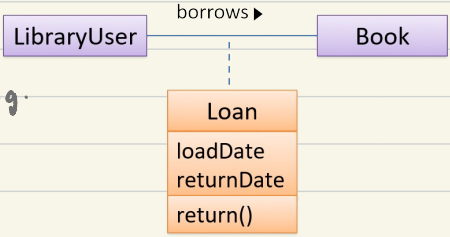
- Summary example :



W4.3e - Paradigms - OOP - Associations - Association classes

An *association class* represents additional information about an association. It is a normal class but plays a special role from a design point of view.





W4.4 - Java : JavaFX

W4.4a - C++ to Java - Miscellaneous Topics - JavaFX

JavaFX is a technology for building Java-based GUIs. Previously it was a part Java itself, but has become a third-party dependency since then. It is now being maintained by OpenJDK.

W4.5 - Java : varargs

W4.5a - C++ to Java - Miscellaneous Topics - Varargs

Variable Arguments (Varargs) is a *syntactic sugar* type feature that allows writing a methods that can take a variable number of arguments.

📦 The `search` method below can be called as `search()`, `search("book")`, `search("book", "paper")`, etc.

```
1 public static void search(String ... keywords){
2    // method body
3 }
```

W4.6 - Code Quality : Naming

W4.6a - Implementation - Code Quality - Naming - Introduction

- Proper naming :
  1. improves readability
  2. reduces bugs caused by ambiguities regarding intent of variable/method

W4.6b - Implementation - Code Quality - Naming - Basic

- Use nouns for things and verbs for actions
- Classes : nouns
- Methods : verbs
- Single-valued vs. multivalued variables

Name for a	👎 Bad	👍 Good
Class	`CheckLimit`	`LimitChecker`
method	`result()`	`calculate()`

```
Person student;
ArrayList<Person> students;
```

W4.6c - Implementation - Code Quality - Naming - Basic

- Use standard words

Use correct spelling in names. Avoid 'texting-style' spelling. **Avoid foreign language words, slang, and names that are only meaningful within specific contexts/times** e.g. terms from private jokes, a TV show currently popular in your country

W4.6d - Implementation - Code Quality - Naming - Intermediate
- Use name to explain

Bad	Good
`processInput()` (what 'process'?)	`removeWhiteSpaceFromInput()`
`flag`	`isValidInput`
`temp`	

Bad	Good
`bySizeOrder()`	`orderBySize()`

Bad	Bad	Good
`value1`, `value2`	`value`, `Value`	`originalValue`, `finalValue`

W4.6e - Implementation - Code Quality - Naming - Intermediate
- Not too long, not too short

While it is preferable not to have lengthy names, names that are 'too short' are even worse. If you must abbreviate or use acronyms, do it consistently. Explain their full meaning at an obvious location.

W4.6f - Implementation - Code Quality - Naming - Intermediate
- Avoid misleading names

Bad	Good	Reason
`phase0`	`phaseZero`	Is that zero or letter O?
`rwrLgtDirn`	`rowerLegitDirection`	Hard to pronounce
`right` `left` `wrong`	`rightDirection` `leftDirection` `wrongResponse`	`right` is for 'correct' or 'opposite of 'left'?
`redBooks` `readBooks`	`redColorBooks` `booksRead`	`red` and `read` (past tense) sounds the same
`FiletMignon`	`egg`	If the requirement is just a name of a food, `egg` is a much easier to type/say choice than `FiletMignon`

W4.7 - Code Reviews

W4.7a - Quality Assurance - Quality Assurance - Code Reviews - What

- **Code review**: the systematic examination code with the intention of finding where the code can be improved

- Forms of reviews: driver writes code, observer reviews
  1. PR reviews
  2. In pair programming
  3. Formal inspections

- the author - the creator of the artifact
- the moderator - the planner and executor of the inspection meeting
- the secretary - the recorder of the findings of the inspection
- the inspector/reviewer - the one who inspects/reviews the artifact.

☺ Pros:

1. detect functionality defects, coding standard violations
2. verify non-code artifacts, incomplete code
3. do not require test drivers / stubs

☹ Cons:

1. manpower-intensive
2. manual process
   ⇒ error-prone

W4.8 - Static Analysis

W4.8a - Quality Assurance - Quality Assurance - Static Analysis - What

- Static analysis: the analysis of code without actually executing the code
- Static:
  - IDEs
  - Linters
- Dynamic:
  - Profilers
  - Code coverage tools

Static analysis of code can find useful information such unused variables, unhandled exceptions, style errors, and statistics. Most modern IDEs come with some inbuilt static analysis capabilities. For example, an IDE can highlight unused variables as you type the code into the editor.

The term *static* in static analysis refers to the fact that the code is analyzed without executing the code. **In contrast, *dynamic analysis* requires the code to be executed to gather additional information about the code** e.g., performance characteristics.

Higher-end static analyzer tools can perform more complex analysis such as locating potential bugs, memory leaks, inefficient code structures etc.

***Linters* are a subset of static analyzers** that specifically aim to locate areas where the code can be made 'cleaner'.

W4.9 - RCS : Managing Pull Requests

W4.9a - Tools - Git and GitHub - Reviewing PRs

- **PR review stage** : dialog between PR author and members of the repo that received the PR, in order to refine and eventually merge the PR

W4.10 - Automating the Build Process

W4.10a - Implementation - Integration - Introduction - What

- **Integration**: combining parts of a software product to form a whole
- One of the most troublesome, rarely goes smoothly

W4.10b - Implementation - Integration - Build Automation - What

- Build scripts

> Some popular build tools relevant to Java developers: Gradle, Maven, Apache Ant, GNU Make
>
> Some other build tools : Grunt (JavaScript), Rake (Ruby)

- Some build tools also serve as **dependency management tools** (update third party libraries that evolve constantly) ← automate

W4.10c - Implementation - Integration - Build Automation - Continuous integration and continuous deployment

- **Continuous integration** (CI): integration, building and testing happens automatically after each code change
- **Continuous deployment** (CD): changes are not only integrated continuously, but also deployed to end-users at the same time