COMP2511

Tutorial 3

Last Week's Tutorial

- Static fields and methods
  - Fields and methods that belong to a class itself, rather than concrete instances
- Inheritance, method overriding and polymorphism
  - 'Is-a' relationships between classes
  - Defining custom behaviour of inherited methods in subclasses
- JavaDoc
  - Code documentation
- Access modifiers
  - Restrictions on the visibility of class fields/methods to other classes

This Week’s Tutorial

- Code review
  - Method overriding
  - Static fields and methods
  - Method hiding (extension material)
- Testing
  - Writing JUnit tests
  - Using VSCode debugging tools
  - Exceptions
- Domain Modelling
  - System design before writing any concrete code
  - Creating UML diagrams

Week 3 Reminders

- Assignment 1 is now out!
  - If you've started, great work!
  - If you haven't started, that's also alright, but please try your hardest to start either today or this weekend!
- If you haven't already, please fill in the Assignment 2/3 Partner Preference form, even if you don't have a specific partner in mind.
  - This form will close on Week 4 Friday, and if you haven't filled it out I'll have to assume that you do not have a partner in mind.

Code Review

src/overriding

src/staticfield

Testing

JUnit

- If you remember Jest testing from COMP1531, you can just think of JUnit as the Jest equivalent for Java.
- Within each test file/class, each of the methods defined in it will be the unit tests.
- Tests are identified using the @Test tag.
- You can perform most logic as normal - conditions are checked using methods that perform specific **asserts**, like *assertEquals*, *assertTrue* and *assertThrow*.

IDE Debugging

- At this point of this course, your main method of debugging has most likely been through using print statements.
- We have access to some more ‘advanced’ debugging techniques that allow us to step through the execution of the program, like GDB for C.
- A **breakpoint** is a specific spot in your code where you would like to intentionally pause execution.
  - In VS Code, they can be set on the *glyph margin*, to the left of each of the line numbers.
  - By running your code in **debug mode**, you can stop execution at breakpoints, allowing you inspect the state of variables and the call stack at that point of time, and also step through the program to analyse the control flow of the program.

Exceptions

- An **exception** is an event that occurs during the execution of a program that disrupts the normal flow of the program's instructions.
  - In simple terms, you can think of exceptions as flags that something has gone wrong, such as dividing by zero or trying to dereference a null object.
  - The process of creating an exception and 'giving' it to the runtime system to deal with it is referred to as **throwing** an exception.
- In Java, exceptions are put into two categories:
  - **Checked exceptions**, which need to be included in a method's signature if it can potentially be thrown and caught in a *try-catch* block (eg. IOException from FileWriter).
  - **Unchecked exceptions**, which do not need to be included in a method's signature or caught in a try-catch block (eg. ArithmeticException, IllegalArgumentException).

Code Example

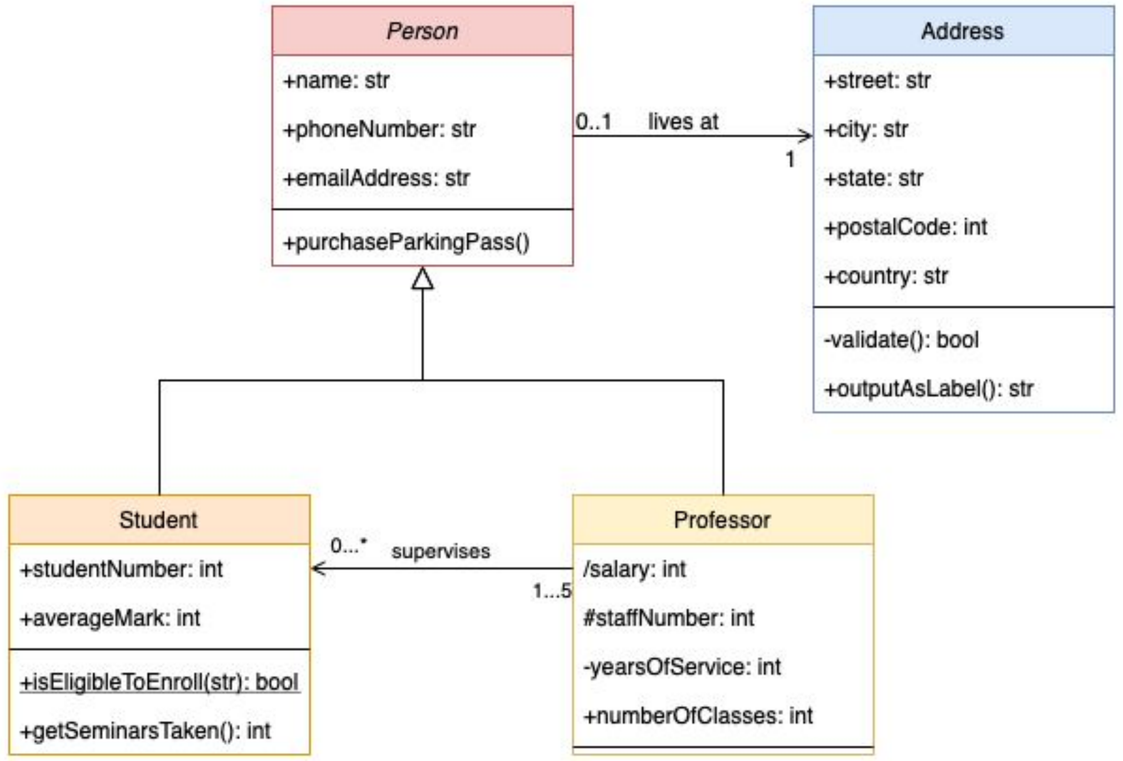
src/wondrous

Domain Modelling

UML Diagrams

- **UML** (Unified Modeling Language) diagrams are commonly used to visually demonstrate the structure of a system in terms of its components (classes) and the relationship between each of its components.
- Cheat sheet for what each of the important components of a UML diagram represent in the repo: https://github.com/jamesandpanda/tutorials/tree/main/COMP2511/24T3/tute03

UML Diagram Example



Live Example

(because demonstrating this most likely makes a lot more sense than trying to explain it through words!!!)

Domain Modelling - Cars

Method Hiding (Extension)

Static Methods

- Can static methods be overridden?
  - Nope!
  - Method overriding is associated with re-defining the functionality of a method defined in an earlier class, but this makes no sense if those methods belong to the earlier class itself, rather than concrete instances…
- Can static methods be inherited?
  - Perhaps surprisingly, yes!
  - This is because you can still call a static method from a concrete instance of the class. This is bad practice however, so you shouldn't really run into this in the wild.
- With this in mind, is there something *similar* to method overriding for static methods?

Method Hiding

- If a subclass defines a static method with the **same method signature** as a static method in the superclass, then the method in the subclass *hides* the one in the superclass.
- If a concrete instance of a class calls a static method, the actual method called depends on its **static type**, or **type of reference**.
    - This is distinct to method *overriding*, where the actual method called is determined by the type of the actual object, and is determined dynamically (i.e. at run-time, rather than at compile-time through static checks). This mechanism is known as **dynamic dispatch**, and is used for all non-static methods in Java.

Code Review

src/hiding