**UML Basics – Syntax**

Box (for each class) is divided into 3 sections

* Class name
* List of main attributes
* List of main methods (no getters/setters)

**UML “Arrows” – 3 main ones…**  
Different arrow symbols for each.

1. Inheritance
   1. Use when one class inherits from another A 🡪 B = A is a child of B
2. Aggregation
   1. Means that B has attributes that are of class A. A 🡪 B = B contains A
   2. Let’s pretend we have a class called Wheels, another class called Car. Inside the Car class, we will store the Wheels class. Therefore, Car contains/stores Wheels attributes.
3. Association
   1. A 🡪 B = A “uses” stuff from B
   2. A calls a method from B, uses a property from B.
4. Composition
   1. A special case of Aggregation, where the thing contained in the class is specific to that class.
   2. Example: Wheels and Car
   3. Car is not really a composition because Wheel can be used for other classes, such as “Truck” “Bike” etc.
   4. However, a class such as Hand 🡪 Human. Human would be a composition (unless design changes later so that Hand is used somewhere else).

**You need to draw these for your assignment / you must submit your UML diagram**

* Best to design first, then code.

**Tips for drawing UML diagrams**

* The aim is to communicate the purpose of your classes and how they work together to solve the larger problem.
* List out the min attributes of the class
* List out the main methods of the class that **define its purpose** (getters/setters are not necessary)

**Programming by contract concepts**

* A method is required to give the correct output **only if** pre-conditions are met.
* CONTRACT: If my pre-condition is met, then it will execute my code.
* **Pre-conditions**: conditions prior to the execution of a method
  + What
  + List down all the trivial, sanity checking, input cleaning stuff
* **Post-conditions**: conditions immediately after the execution of a method
  + What this function does, immediately after the method finishes executing
  + What this function is meant to do
* Implication: no input “cleaning” or “validity checking” required.  
  **DON’T CHECK PRE-CONDITIONS IN YOUR METHODS (i.e IF STATEMENTS ETC.)**

**More concepts**

* **Invariant**: a condition that is always met throughout the life of a class **before and after every method call**
  + I.e. Salary > 0 (can’t ever be lower than 0)
  + A consistent condition throughout the life of a class
* **Covariance/ Contra-variance**: Generality of pre-post conditions (Refer to lec slides)
* **Liskov Substitution Principle**: All your parent methods should work on your children.
  + E.g. Employee has field “name”, therefore Manager should have one too.

**J-Unit testing**

* A special test-class where you can define functions, similar to having a “main” for testing properties.
* Optional feature for assignments

Test {

}

**Declaring pre/post conditions – syntax**

/\*\*

* @param a, first number to input, a > 0
* @param b, second number to input, b < 0
* @return a negative integer

\*\*/

inside your method, press SHIFT+ALT+J to automatically generate this

*public int getNegative(int a, int b) {*

*return a\*b;*

*}*

A function doesn’t have to test every case, just return [missing]