SWEN20003
Object Oriented Software Development
Workshop 7

Eleanor McMurtry

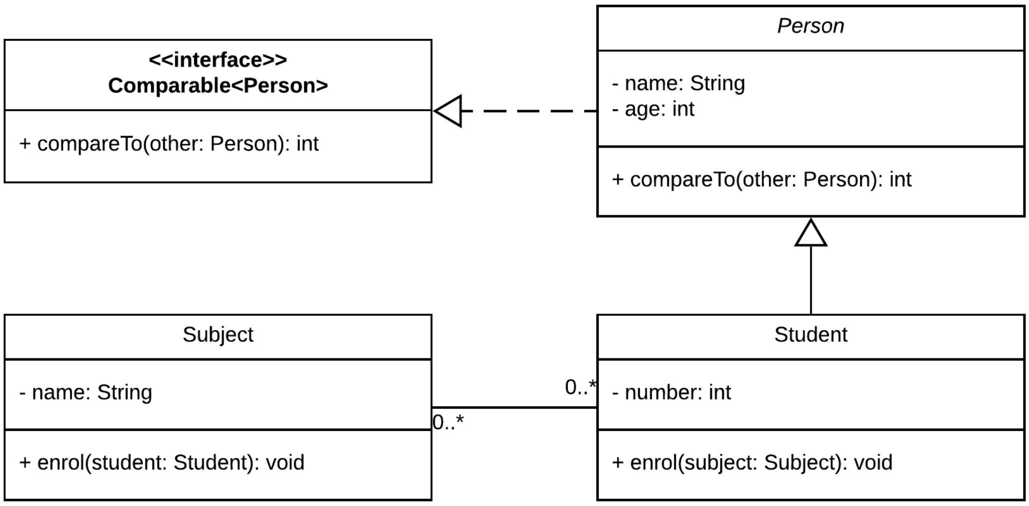
Semester 2, 2020

Workshop

This week, we are learning to use UML class diagrams for designing and communicating our Java programs.

- A UML class diagram consists of classes (each of which have a name, zero or more attributes, and zero or more methods), as well as relationships between classes.
- Privacy of attributes and methods (`+` or) comes **before** their name. Types of attributes and methods come **after** their name.
- An association from class `A` to class `B` means that `A` has an attribute of type `B`. You **must** use associations **instead of** attributes for classes that appear on your diagram. Classes that do not appear on your diagram (such as `String`) **do not need associations**.
- Associations come with a **multiplicity** representing how many instances of the type are stored (for example, in an array or `ArrayList`).
- Inheritance and interfaces are represented with an arrow **from** the subclass (or implementing class) **to** the parent class (or interface).
- Abstract classes and methods are represented using *italics*. Static attributes and methods are represented with an underline.
- Final classes, attributes, and methods are (optionally)represented by adding `<<final>>` before their name.
- Constructors, getters, and setters are not always shown on UML class diagrams.

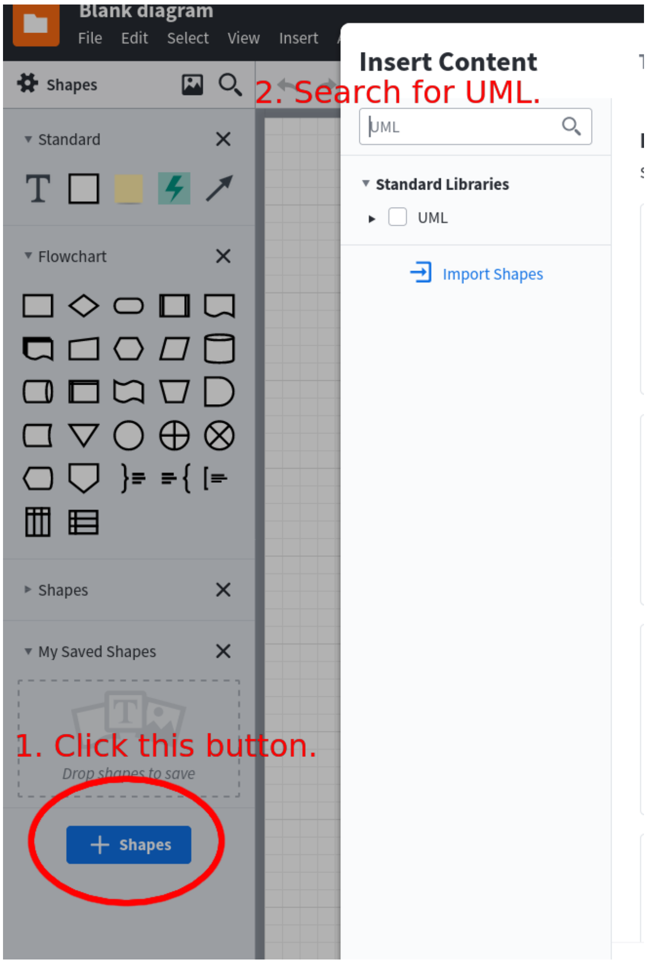
There are many software options for creating UML class diagrams. We recommend https://lucidchart.com; a free account will be enough for this subject. Below is an example diagram showing these concepts.



<<interface>>
Comparable<Person>
+ compareTo(other: Person): int
Person
- name: String
- age: int
+ compareTo(other: Person): int
Subject
- name: String
+ enrol(student: Student): void
Student
- number: int
+ enrol(subject: Subject): void
0..*
0..*

Creating UML with LucidChart

To use UML in LucidChart, follow the below steps. Click `UML` when it appears, then click `Use Selected Shapes`.



2. Search for UML.

1. Click this button.

Questions

1. Implement Java classes following the diagram on the previous page.
2. Create a UML class diagram to represent the classes and interface from Question 1 last week. (The question is reproduced below.)
   (a) Define a `FileWriteable` interface with a method
   `void writeToFile(BufferedWriter writer) throws IOException`
   This method should be used to write some textual representation of the object to the provided `BufferedWriter`. (This is a process called **serialisation**.)
   (b) Define the following classes, and implement the interface for them:
   - `Point`, with attributes `x` and `y`
   - `Student`, with attributes `name` and `id`
   - `Car`, with attributes `model` and `colour`

   class?
   (c) Define a class `Database`. It should store up to 100 `FileWriteable` objects. Objects can be added to and removed from the database.
   (d) Add a method `void writeAll(String filename)` that opens a file called `filename`, and writes all of its objects to that file.
3. Create a UML class diagram representing a design for the following scenario:
   - The game of Monopoly is defined by a *board*, which contains 40 *spaces*, and between 2 to 6 *players*.

   - A space can be either a *property*, *chance*, or *bonus*, and each of the types has a different *action* when a player lands on them.
   - Properties may additionally be *railway stations* or *utilities*, each with a different action when a player lands on them.
   - Players each have a position on the board, an amount of money that they have, a number of properties that they own, and can move along the board.

4. Create a UML class diagram representing a design for the following scenario:
   - We are ambitious Java enthusiasts and are already ready to begin creating our own small ‘graphics’ library. We are designing a system to render simple shapes onto the screen. For now, we are concerned about two types of shapes in particular: **square**s and **triangle**s. A shape has a specific area associated with it, and it can also be rendered to the screen.
   - A shape also has a **colour** associated with it. We will be using the the RGB colour system which specifies a colour through three values: *red*, *green*, *blue*. The red, green, and blue values of a colour must be within the range of 0-255 (inclusive) at all times. If a colour is not specified, a shape's default colour is black (red = 0, green = 0, blue = 0).