Lecture 2.1 — OO

Some Caveats

Object Oriented "OO" is *a* way of thinking about programs.
It is *not*:

- ...the only/best approach for all uses. (We do focus on it in this course).
- ...entirely limited to OO languages. (But is easier in them).
- ...a way to avoid basic programming skills. (decomposition, functions, repetition, ...).

Perspective 1 — Modelling

In programming we often treat design as a modelling exercise.

- DEM: Particles, bonds
- University: Faculties, Schools, Staff, Students, Courses, Tutes

So

1. Represent problem as a set of interacting entities.
2. Represent interactions as "messages" between entities.
3. ?
4. Profit

It would be nice to be able to talk about entities both as groups and subgroups. "Exchange students are like Undergrad students but with the following differences."

Perspective 2 — Practical Programming

Suppose we want to represent a student ($:=\{$ID, name(s), program, ...eye colour(?) ...lift strength(?)$\}$) in our program. Doesn't really fit any of the standard types. We need a compound type. Benefits:

- Simpler — one var is easier than twenty.
- Abstraction — Not all code needs to know what's in a student (or that it has changed).

Note: not necessarily an explicit part of the model. The grouping may be done out of convienience (especially in Java).

Types

Types are defined by:

- State — values it can take.
- Behaviour — what can be done with those values.

For example:

Type	Values	Operations
int	1, 2, -50, . . .	$+, -, *, /, \%$
boolean	true, false	&&, ||, !
float	1.0, 3.8, . . .	$+, -, *, /,$ sqrt

A type can be viewed as either (or both):

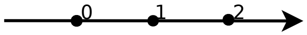
- A description of the values and operations.
- The set of all possible values of that type. In OO terms each value is an object (or "instance of the class"). The class is the set of all objects of that type.

OO languages bundle behaviour into classes along with state.

- Referred to as *member functions* or *methods*.
- So data and the code which interacts with it are “together” (*encapsulation*).

Note: the variables which store state are referred to as *member variables* or *fields*.

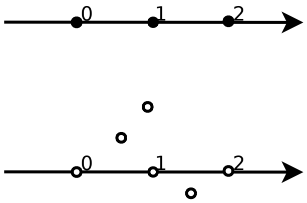
Complex example — don't panic



`Complex.java`

Constructors, return types, `new`, `.` for member access. `this` to refer to the object you are in.

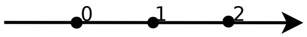
Complex example — don't panic

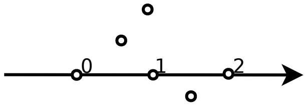


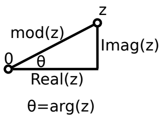
`Complex.java`

Constructors, return types, `new`, `.` for member access. `this` to refer to the object you are in.

Complex example — don't panic







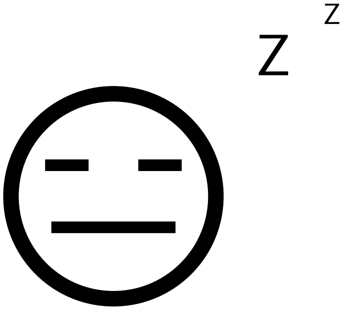
`Complex.java`
Constructors, return types, `new`, `.` for member access. `this` to refer to the object you are in.

Constructors

`Cons.java`

- Classes can have multiple constructors.
- If none is explicitly declared, Java will make a default one.
- You can not call constructors which don't exist (or that you don't have access to).

Constructors are how you ensure that all objects start in a valid state.


Z
z

Invariants and “interfaces”

An invariant is a condition which “always” holds. eg A student’s DOB can’t be in the future. The methods of a class should be written to preserve invariants. The methods provide an “interface”.
But:
student1.dob=new Date(2258, 4, 15); // ???[1]
Soln:

- Constructors to make sure objects start valid.
- All methods[2] to ensure they only move from one valid state to another valid state.
- *Information hiding* — block/obstruct direct access to member variables, to prevent bypassing the above.

[1]Yes I know this call is deprecated
[2]called directly or triggered implicitly

Information hiding — Access control

Classes, member variables and methods specify who is allowed to use them:

- `public` — Any code can use this.
- `private` — Only this class can use this.
- `protected` — (later) Only this class and subclasses can use this.
- — (later) Only code in this package can use this.

Unless you have a good reason, member variables should be private[3].

[3]Warning this is not enough to completely protect your invariants

Information hiding — Abstraction

If users of your class are calling its methods, do they need to know how state is represented (or how methods are implemented)?
Alternatively, can you change the internals of a class without breaking external code? — Easier if they don't have direct access to internals.
Consider: `AltComplex.java` and `CplxTest.java`

Last Magic — static

```
public class XYZ {

public static void main(String[] args) {
    // your code here
}

}
```

Defn: `static` == belongs to class as a whole not any particular object.
Uses:

- Shared constants / flags / logging variables
- Operations which only depend on the above (so you don't need to make and object to call the method on).
- For when you don't have an object (eg `main`).

import

```
import java.util.List;    // use short name
                          // for one class
import java.io.*;         // use short names for
                          // whole package
```

Java's `import` statement works differently to Python's.

Python
: import could execute code. You can't use a class without importing it.

Java
: You can use a class without importing it *if* you use its full name.

These are equivalent:

```
java.io.FileReader f;
```

```
import java.io.FileReader;
...
FileReader f;
```

`import`

Q: If `String` is actually `java.lang.String` then why don't we need to `import` it?
A: Everything in `java.lang` is imported automatically.