Lecture 2 - May 8

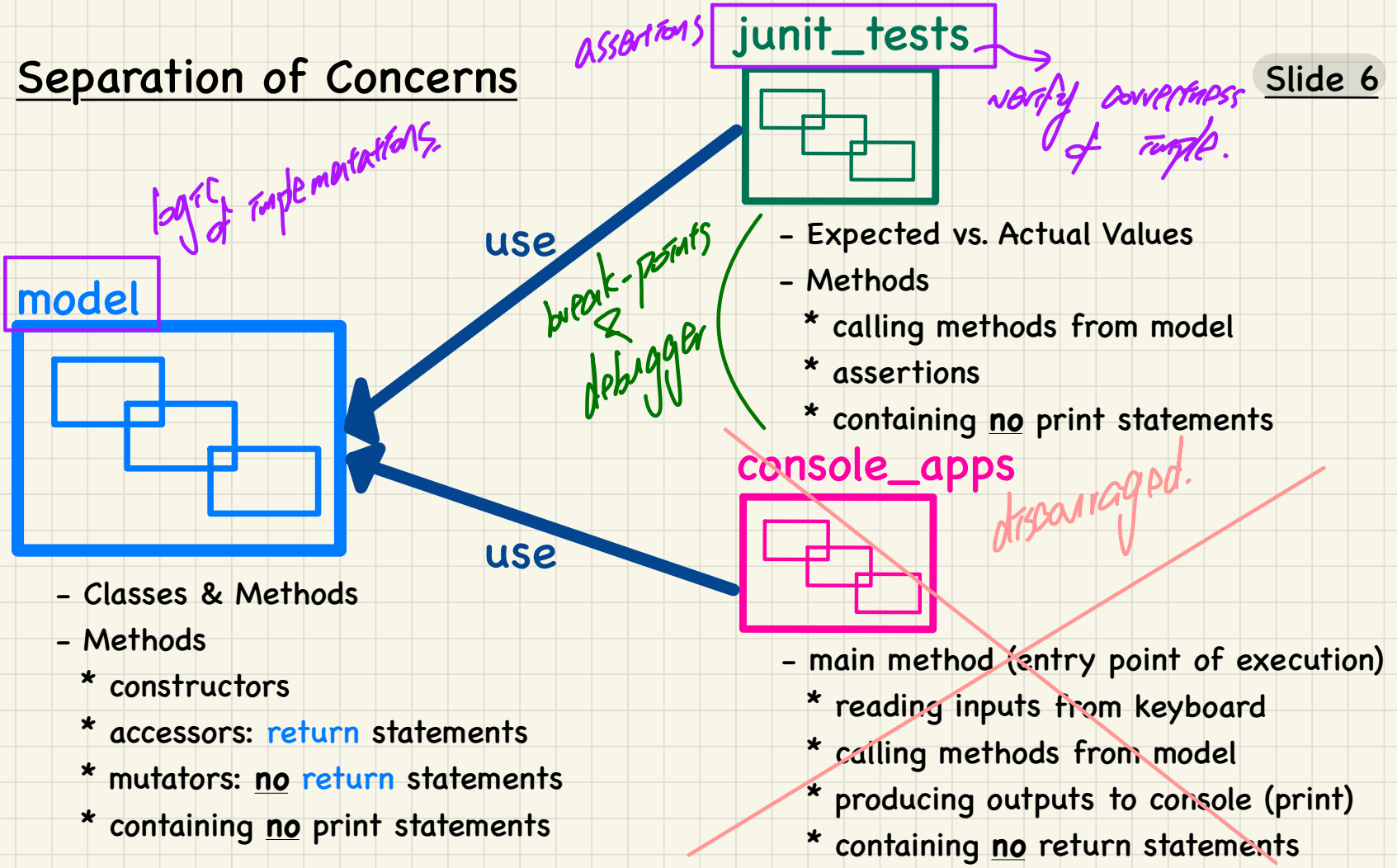
Review of OOP

Separation of Concerns: model vs. tests
OO: Observe-Model-Execute Process
Java Data Types: Primitive vs. Reference

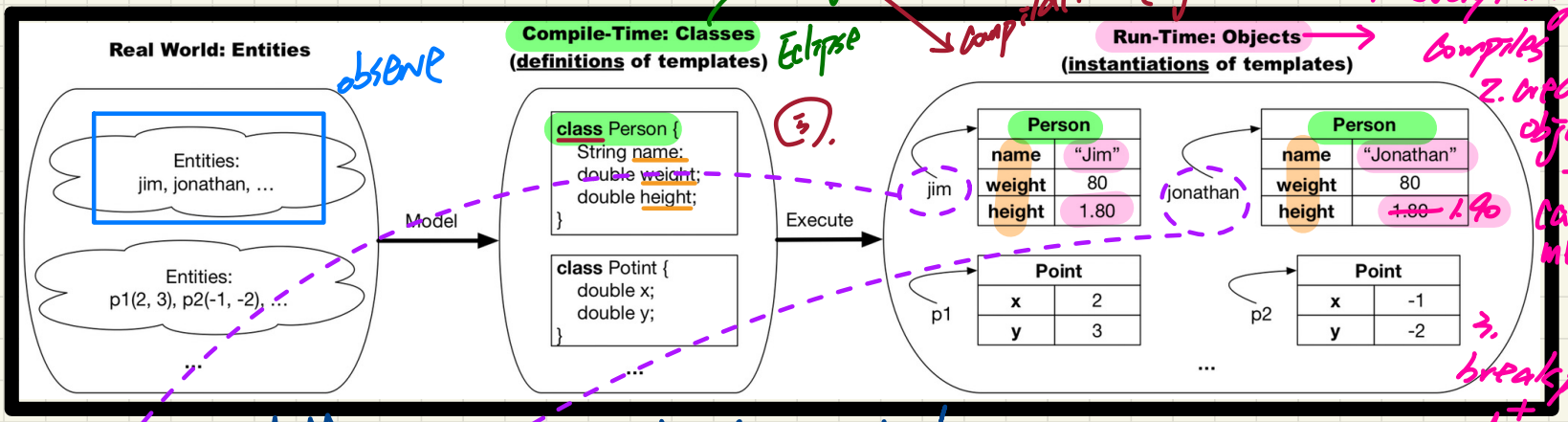
Announcements/Reminders

- First Class (Syllabus) recording & notes posted
- Today's class: notes template posted
- Priorities:
  - + LabOP1 (tutorial videos + PDFs)
  - + Due: Next Tuesday
- LabOP2 released next Monday
- Lab this Friday (May 9): self-guided, no TAs
- Lab next Friday (May 16):
  - + [≈ 40 min] Mock-up Programming Test 0
  - + [≈ 40 min] Q&As (TAs, Jackie)

Separation of Concerns



Observe-Model-Execute Process



observe

editing in Eclipse

no compilation errors

compilation errors (syntax, type)

1. everything compiles
2. create objects + call meth.
3. breakpoints + debugger

1. different entities of the same kind

(Entities) Persons

Jim

Jonathan

Entities:

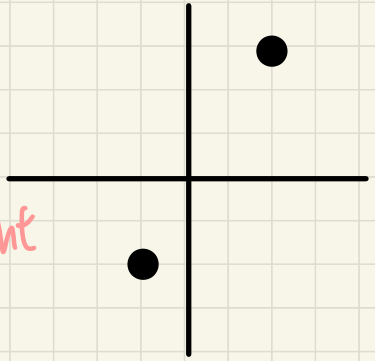
Attributes: * weight, height

Changes: growTaller

Inquiries: getHeight

Template:

Person



Entities:

Attributes:

Changes:

Inquiries:

Template:

OO

templates

instances

changes

suggestions

OOP (Java)

classes

objects

mutators/setters e.g. void setH(double height)

non-void.

accessors/getters e.g. String getW()

double getBMI()

jonathan, jim

- Entities of the same kind share the same set of attributes.

weight, height

- Values of attributes among entities (of the same kind) may have diff. values.

jim.height ≠ jonathan.height.

type error

```
String s = "23";
int i = 46;
int j = s * i;
```

type error

Object Oriented Programming (OOP)

- Templates (compile-time Java classes)
  - + attributes (common around instances)
  - + methods
    - * constructors
    - * accessors/getters
    - * mutators/setters
  - + Eclipse: Refactoring

modify the internal workings without any public/external users realizing it.
(e.g. create/use helper methods to remove code duplicates)

- Instances/Entities (runtime objects)
  - + instance-specific attribute values
  - + calling constructor to create objects
  - + using the "dot notation", with the right contexts, to:
    - * get attribute values
    - * call accessors or mutators

obj1.m1().m2().m3()...

Modelling: from Entities to Classes

Identify Critical Nouns & Verbs

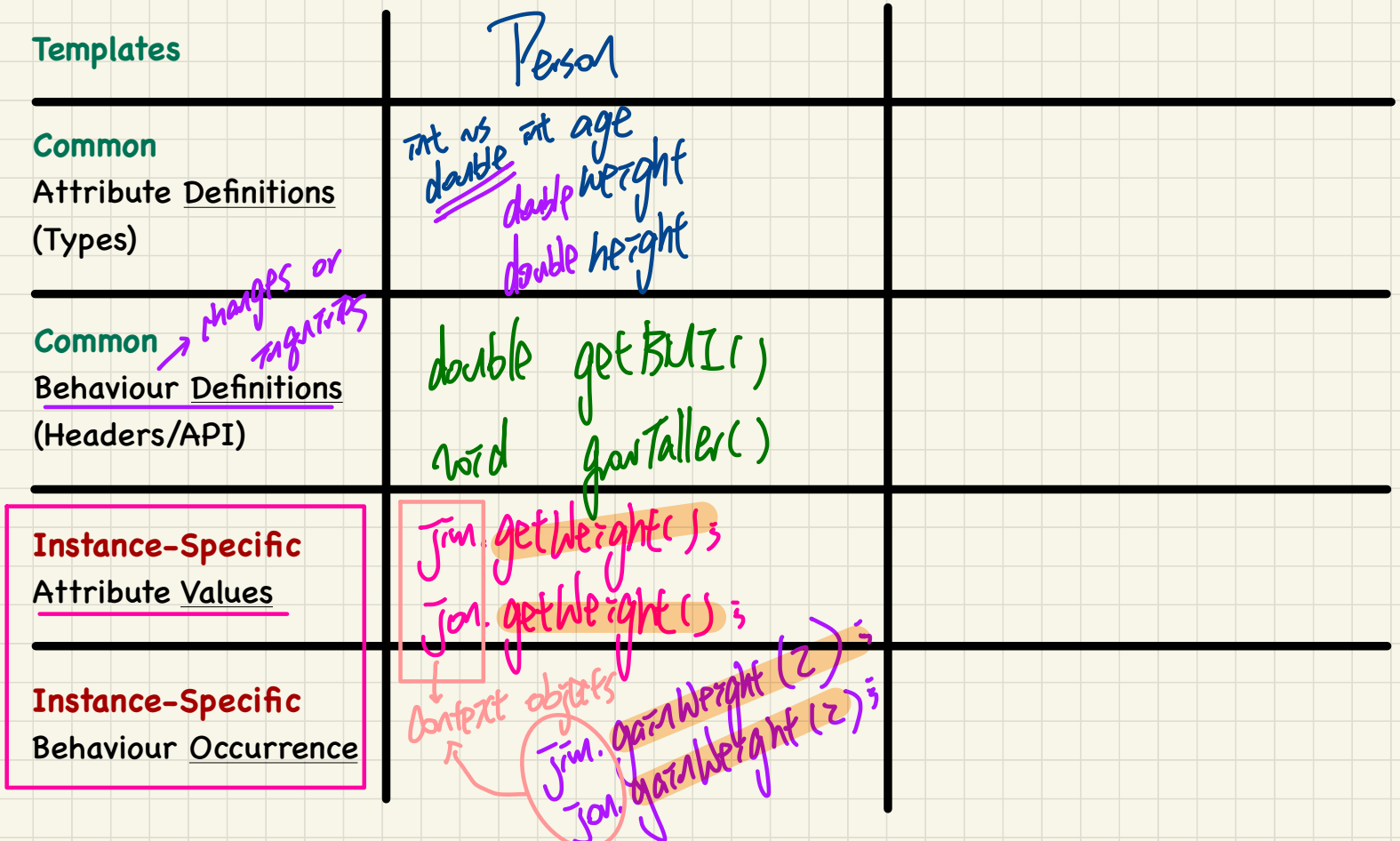
Example 1 (Exercise)

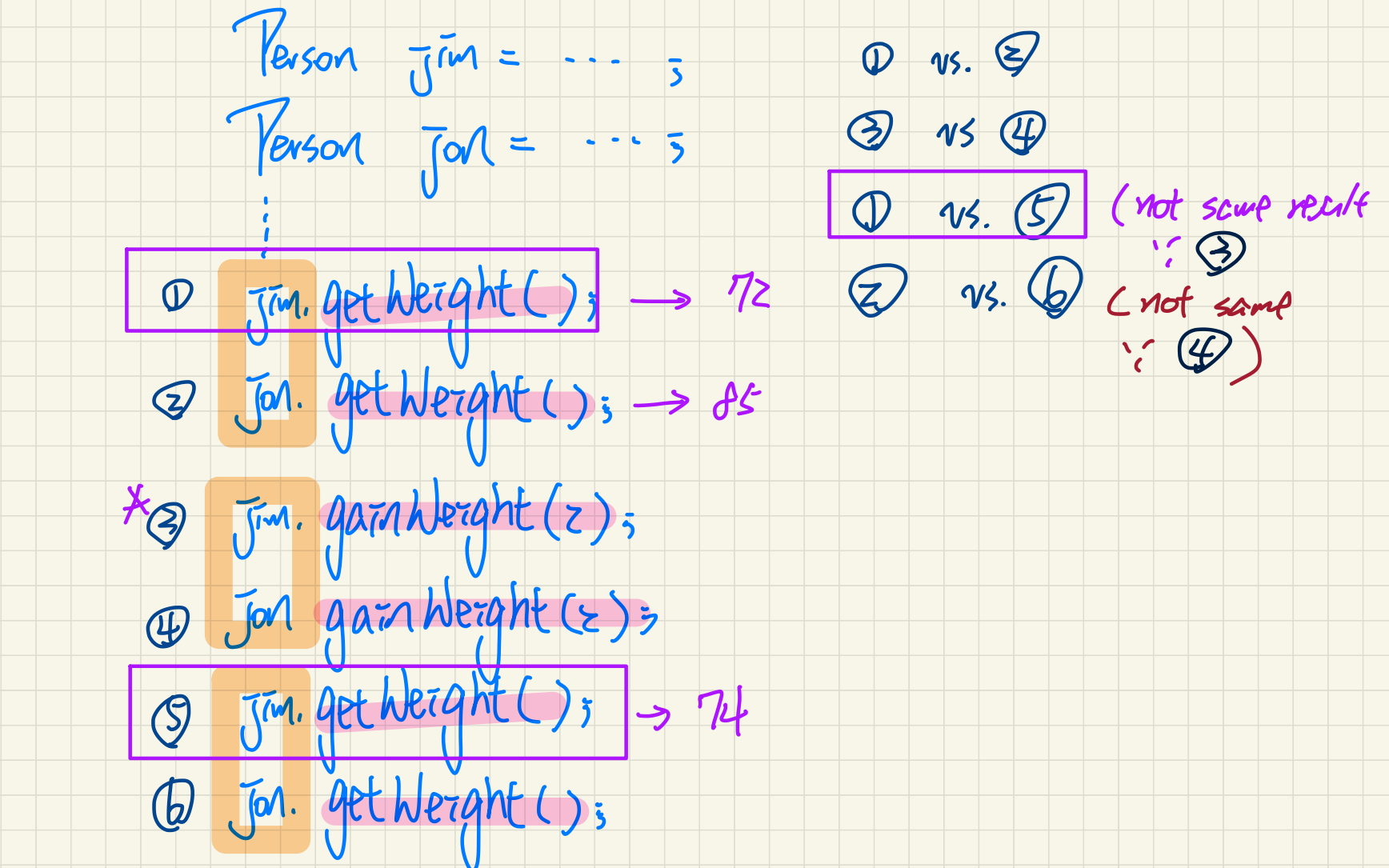
Points on a two-dimensional plane are identified by their signed distances from the X- and Y-axises. A point may move arbitrarily towards any direction on the plane. Given two points, we are often interested in knowing the distance between them.

Example 2

A person is a being, such as a human, that has certain attributes and behaviour constituting personhood: a person ages and grows on their heights and weights.

OO Thinking: Templates vs. Instances




Person jim = ... ;
Person jon = ... ;
① jim.getWeight(); → 72
② jon.getWeight(); → 85
* ③ jim.gainWeight(2);
④ jon.gainWeight(2);
⑤ jim.getWeight(); → 74
⑥ jon.getWeight();
① vs. ②
③ vs ④
① vs. ⑤ (not same result ∵ ③)
② vs. ⑥ (not same ∵ ④)

- In Java, each variable must be declared with a type.

static-typed PL.
dynamic-typed PL: e.g. Python.

e.g. `int i;` — at r.t. can store any int. value
`i = 23;` ✓
`i = 46.7;` ✗

e.g. `Person p;` — at r.t. can store only address of an object instantiated from the Person class

p.t. r.t.

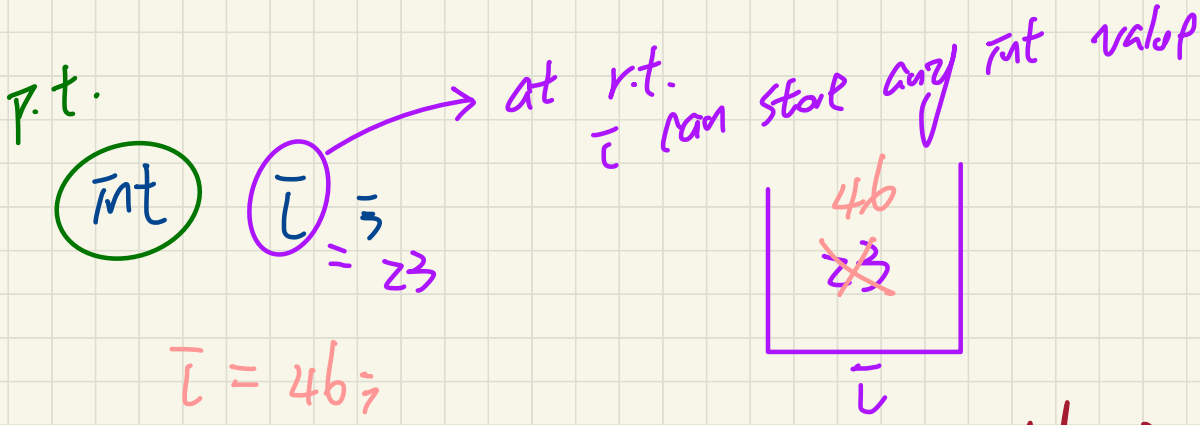
`p = ?;`

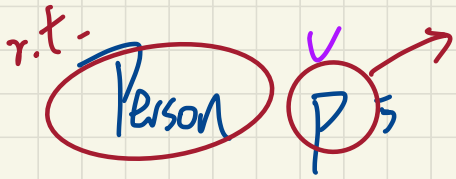
① `p = new Person(..);`

② `Person p2 = ...;`
`p = p2;` (aliasing)

* ③ `C obj = ...;`
`p = obj;` valid if C is a subclass of Person

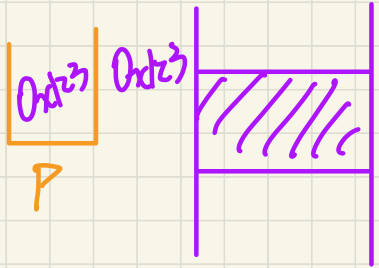
- A declared type denotes the set of values that's allowed to be stored in that variable.





at r.t.
p can store the ref. (address)
of some Person object

(1)



(2)