Lecture 2 - Sep. 10

Review of OOP

Observe-Model-Execute Process
Java Data Types
NullPointerException
Parameters vs. Arguments
Scope of Variables

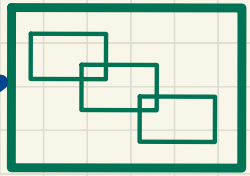
Announcements/Reminders

- Priority: Lab0P1 (tutorial videos + PDFs)
- Lab0P2 released soon
- Wednesday's lab: in-lab code demo!

helper methods

Separation of Concerns

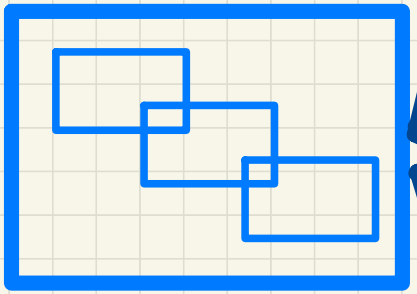
junit_tests



- Expected vs. Actual Values
- Methods
  * calling methods from model
  * assertions
  * containing **no** print statements

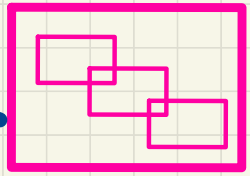
use

model



use

console_apps



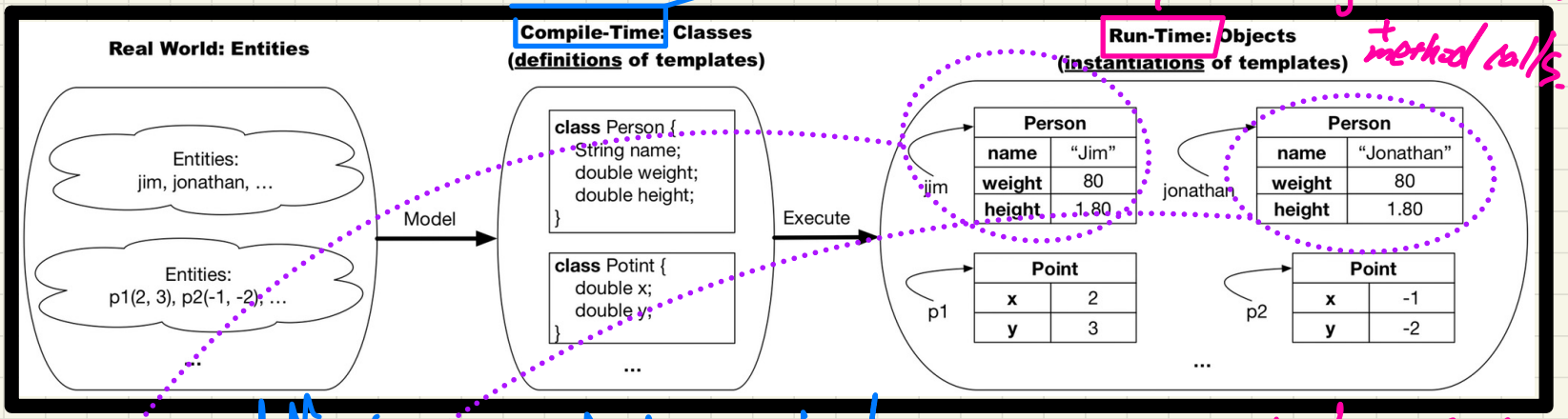
- Classes & Methods
- Methods
  * constructors
  * accessors: return statements
  * mutators: **no** return statements
  * containing **no** print statements

- main method (entry point of execution)
  * reading inputs from keyboard
  * calling methods from model
  * producing outputs to console (print)
  * containing **no** return statements

Observe-Model-Execute Process

developing code in editor

1. everything compiles
2. object creations + method calls



1. different entities of the same kind

3. break points + debugger

Jim

Jonathan

Person

Entities: Jim, Jonathan

Attributes: weight, height

Changes:

Inquiries: getWeight()

Template:

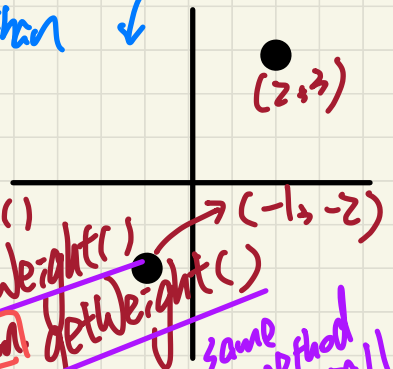
different context obj.

Jim getWeight()

Jonathan getWeight()

same method call

Exercise



Entities:

Attributes:

Changes:

Inquiries:

Template:

Object Oriented Programming (OOP)

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

change
↳ rename an attr
↳ rename a class
↳ helper methods.

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

Modelling: from Entities to Classes

Identify Critical Nouns & Verbs

Example 1

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.

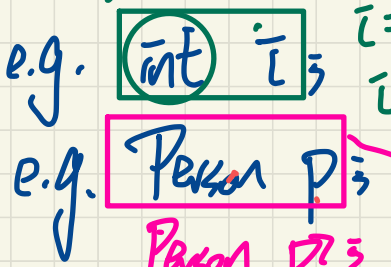
OO Thinking: Templates vs. Instances

jim.gainW(2);
c.o. expected to be modified for its att value

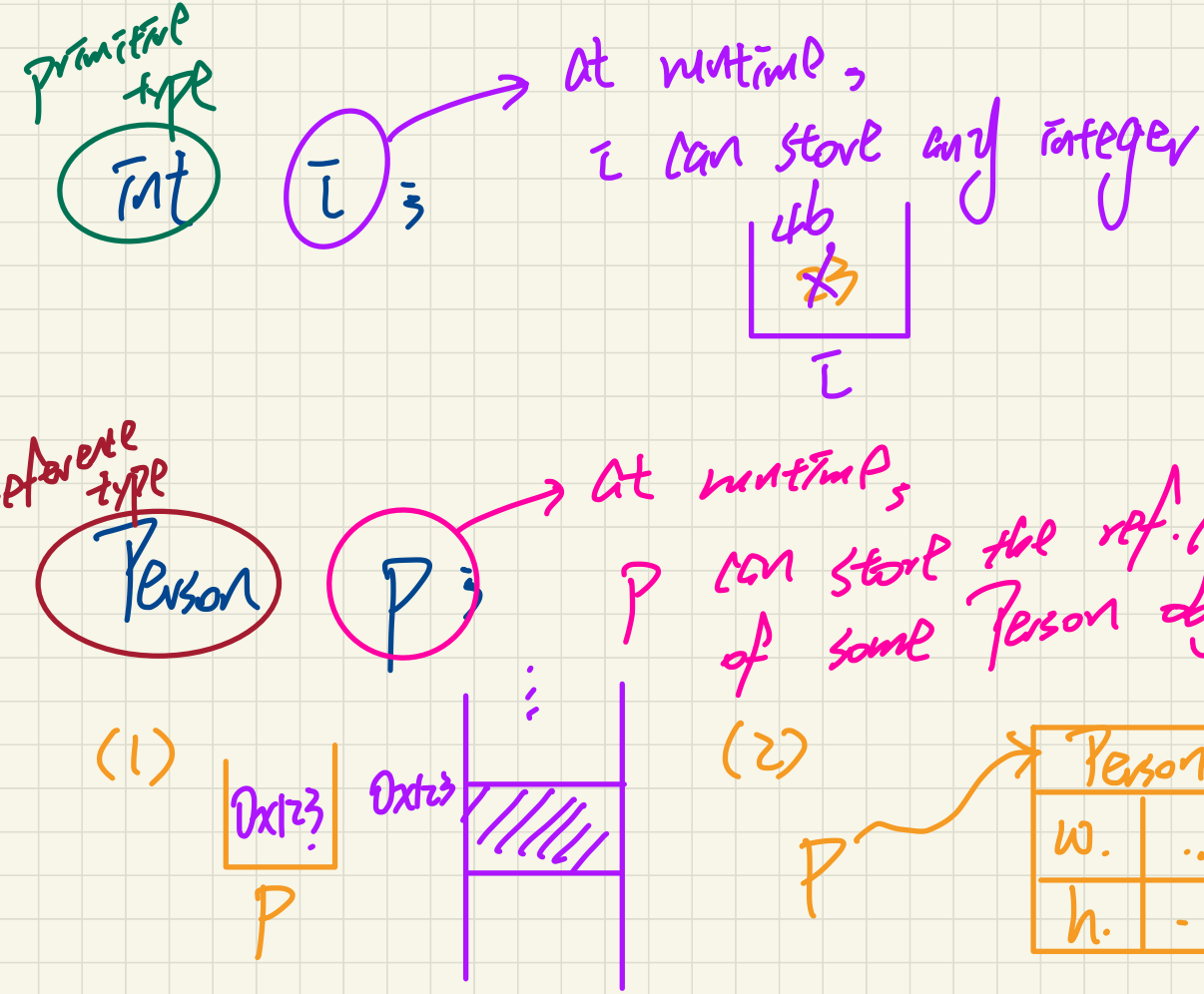
Templates	Person	
Common Attribute Definitions (Types)	double weight double height	
Common Behaviour Definitions (Headers/API)	double getWeight() accessor void gainWeight(double d) mutator	
Instance-Specific Attribute Values	jim.getBMI(); jon.getBMI();	
Instance-Specific Behaviour Occurrence	jim.gainW(2); jon.gainW(2);	

Exercise.

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



- A declared type denotes the set of values that is allowed to be stored in that variables.


primitive type
int
i
= 3;
At runtime,
i can store any integer
4b
3
i
reference type
Person
p
= ;
At runtime,
p can store the ref./address of some Person object.
(1)
0x123
p
0x123
(2)
p
Person
w.
..
h.
..

NullPointerException

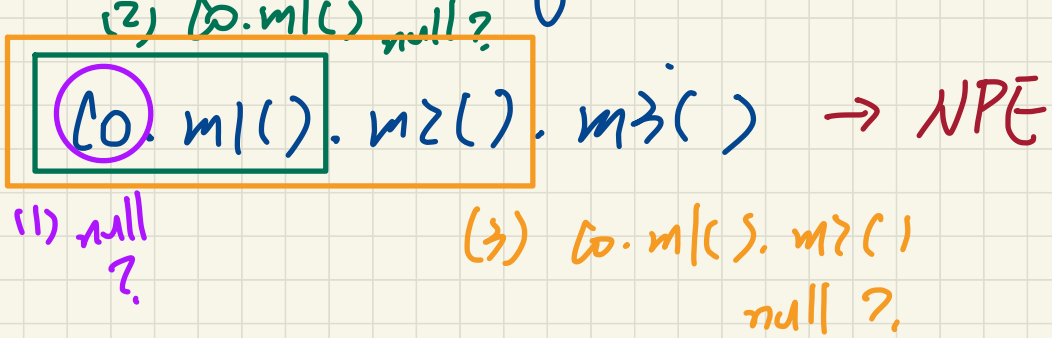
NPE if jim == null

jim.getBMI()

- method call

```
obj.m(...);
```

- context object is null



Slides 16 ~ 47

assigned readings!

Parameters vs. Arguments

```
class Point {
   Point(double x, double y) {…}

   double getDistanceFrom(Point other) {…}

   void move(char direction, double units) {…}
}
```

params

Template Definition

Method Usages

```
class PointTester {
   static void main(String[] args) {
      Point p1 = new Point(2.5, -3.6);
      Point p2 = new Point(-4.8, 5.9);
      double dist1 = p1.getDistanceFrom(p2);
      double dist2 = p2.getDistanceFrom(p1);
      p1.move('R', 7.6);
   }
}
```

arguments

Scope of Variables in a Class

- Class-level variables may be accessed/modified between methods.

  [Assume for now: non-static]
- Method-level parameters may be accessed within the declared method only.
- Method-level (local) variables may be accessed/modified within the declared method.

```
class MyClass {
  int i;
  int j;
  void m1 (int p1) {
    int m;
  }
  void m2 (int p2) {
    int n;
  }
}
```

class-level

method-level

local var.

m = i; m = p1; int m2 = m;

i = n; n = p1; n = m; X