Wednesday March 13

Lecture 18

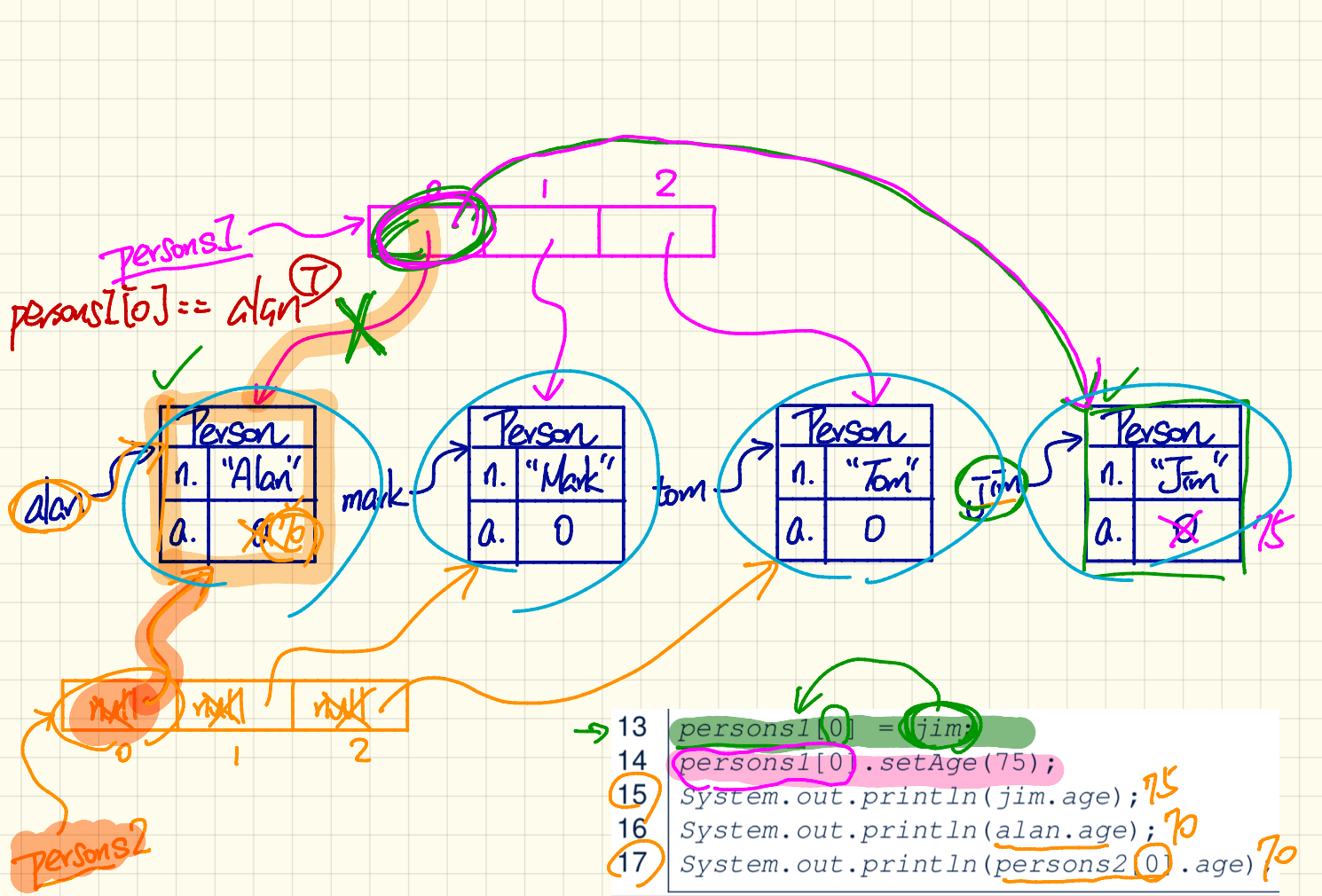
- Lab 6
- Lab Test 3
  - ~ Guide
  - ~ Practice Test
- Tutorial Videos

```
1  Person alan = new Person("Alan");
2  Person mark = new Person("Mark");
3  Person tom = new Person("Tom");
4  Person jim = new Person("Jim");
5  Person[] persons1 = {alan, mark, tom};
6  Person[] persons2 = new Person[persons1.length];
7  for(int i = 0; i < persons1.length; i ++) {
8    persons2[i] = persons1[i]; }
9  persons1[0].setAge(70);
10 System.out.println(jim.age);
11 System.out.println(alan.age);
12 System.out.println(persons2[0].age);
13 persons1[0] = jim;
14 persons1[0].setAge(75);
15 System.out.println(jim.age);
16 System.out.println(alan.age);
17 System.out.println(persons2[0].age);
```

alan.age == 0

0
1
2

class Person {
String name
int age
}


persons1
0
1
2
persons1[0] == alan?
Person
n. "Alan"
a. 0
Person
n. "Mark"
a. 0
Person
n. "Tom"
a. 0
Person
n. "Jim"
a. 75
alan
mark
tom
Jim
persons2
0
1
2
13 persons1[0] = jim;
14 persons1[0].setAge(75);
15 System.out.println(jim.age);
16 System.out.println(alan.age);
17 System.out.println(persons2[0].age)
75
70
70

Person[] persons;

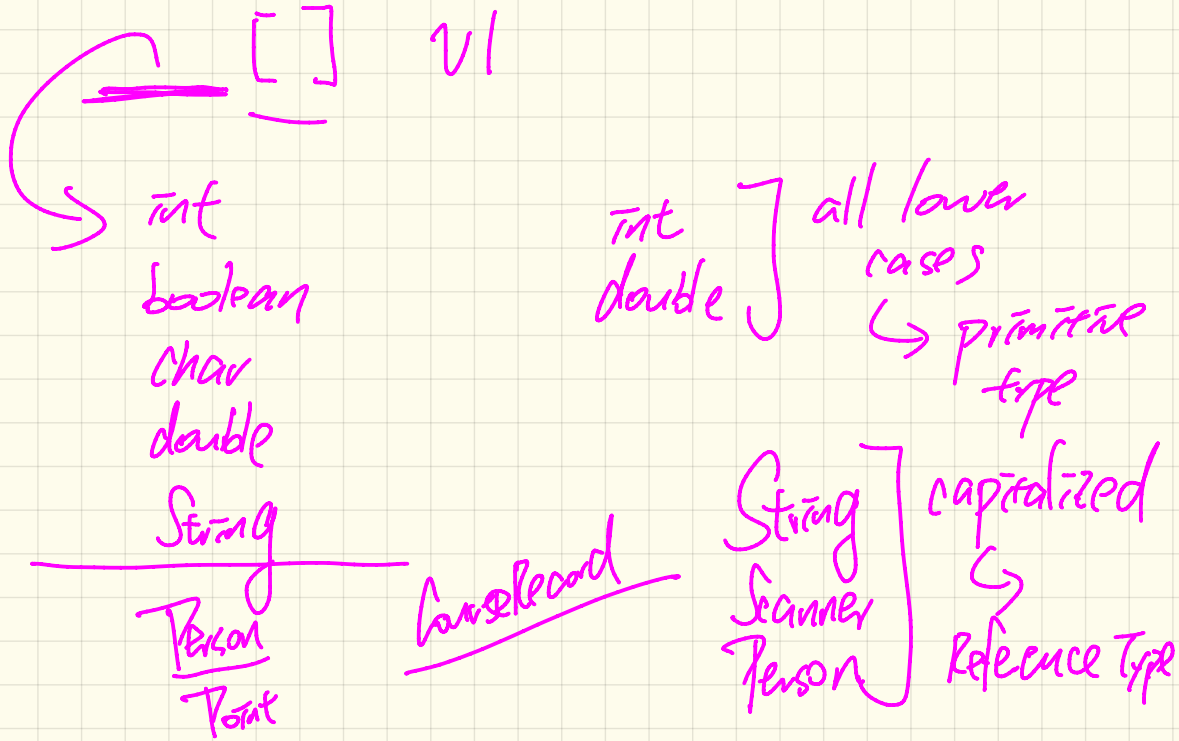
↳ as if:

Person persons[0]

Person persons[1]

⋮

Person persons[persons.length - 1]


[ ]
v1
int
boolean
char
double
String
Person
Point
casesensitive
int
double
all lower cases
primitive type
String
Scanner
Person
capitalized
Reference Type

String s1 = null;

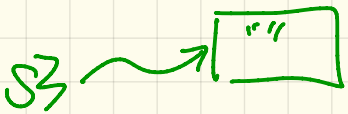
↳ store null address to begin with.

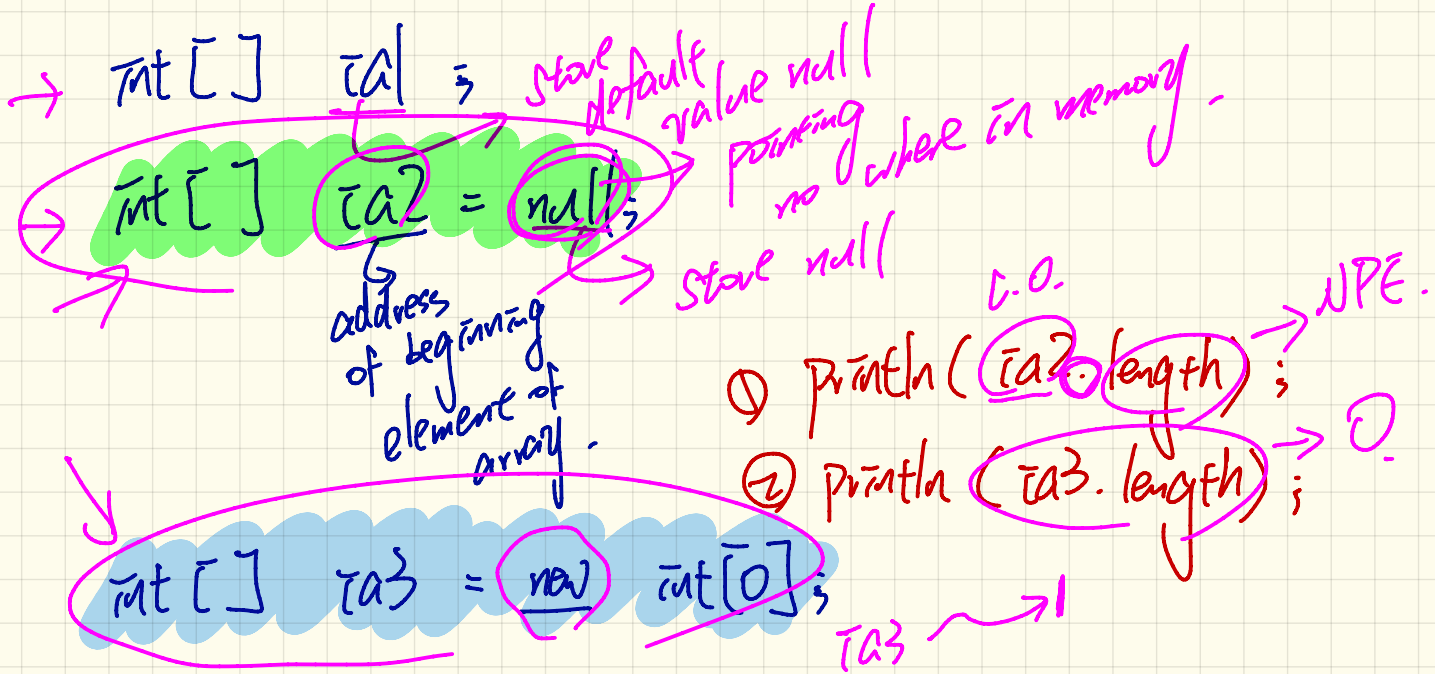
String s2;

↳ store default value null

String s3 = "";

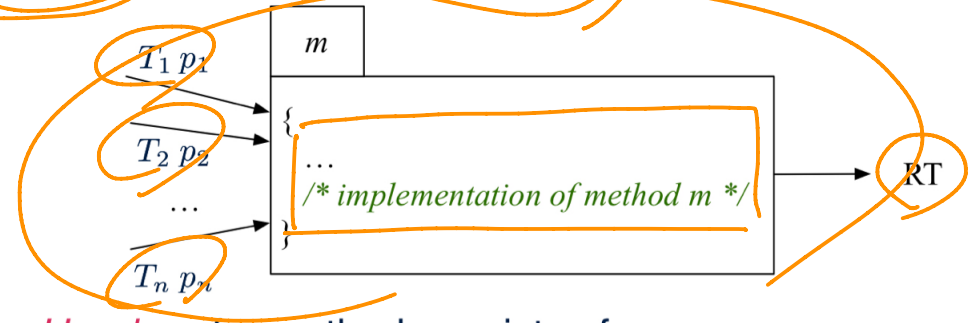
↳ empty string. "" " "




int [] ia ;
store default value null
int [] ia2 = null;
pointing no where in memory.
store null
address of beginning element of array.
C.O.
NPE.
① println(ia2.length);
② println(ia3.length);
0.
int [] ia3 = new int[0];
ia3

What is a method?

- A *method* is a named block of code, *reusable* via its name.



- The *Header* of a method consists of:
  - Return type [*RT* (which can be `void`)]
  - Name of method [*m*]
  - Zero or more *parameter names* [$p_1, p_2, \ldots, p_n$]
  - The corresponding *parameter types* [$T_1, T_2, \ldots, T_n$]
- A call to method *m* has the form: $m(a_1, a_2, \ldots, a_n)$
  Types of *argument values* $a_1, a_2, \ldots, a_n$ must match the the corresponding parameter types $T_1, T_2, \ldots, T_n$.

Parameters vs. Arguments

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

   double getDistanceFrom(Point other) {…}

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

parameters

Class/Template Definition

name of param.

parameters

```
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);
   }
}
```

argument

argument

argument

double getDist(Point p1 , Point p2)

other

other

Method Usages

Kinds of Methods

1. *Constructor*
   - Same name as the class. No return type. *Initializes* attributes.
   - Called with the **new** keyword.
   - e.g., `Person jim = new Person(50, "British");`
2. *Mutator*
   - *Changes* (re-assigns) attributes
   - `void` return type
   - Cannot be used when a value is expected
   - e.g., `double h = jim.setHeight(78.5);` is illegal!
3. *Accessor*
   - *Uses* attributes for computations (without changing their values)
   - Any return type other than `void`
   - An explicit *`return` statement* (typically at the end of the method) returns the computation result to where the method is being used.
   - e.g., `double bmi = jim.getBMI();`
   - e.g., `println(p1.getDistanceFromOrigin());`

Use of Accessors vs. Mutators

```
class Person {
  void setWeight(double weight) {...}
  double getBMI() {...}
}
```

- Calls to *mutator methods* *cannot* be used as values.
  - e.g., `System.out.println(jim.setWeight(78.5));` ×
  - e.g., `double w = jim.setWeight(78.5);` ×
  - e.g., `jim.setWeight(78.5);` ✓
- Calls to *accessor methods* *should* be used as values.
  - e.g., `jim.getBMI();` ×
  - e.g., `System.out.println(jim.getBMI());` ✓
  - e.g., `double w = jim.getBMI();` ✓

Method Parameters

- **Principle 1:** A *constructor* needs an *input parameter* for every attribute that you wish to initialize.

  e.g., `Person(double w, double h)` vs. `Person(String fName, String lName)`

- **Principle 2:** A *mutator* method needs an *input parameter* for every attribute that you wish to modify.

  e.g., In `Point`, `void moveToXAxis()` vs. `void moveUpBy(double unit)`

- **Principle 3:** An *accessor method* needs *input parameters* if the attributes alone are not sufficient for the intended computation to complete.

  e.g., In `Point`, `double getDistFromOrigin()` vs. `double getDistFrom(Point other)`