Lecture 4 - Sep. 17

Review of OOP

Reference Aliasing & Primitive Arrays

Announcements/Reminders

- Lab0P2 due this Friday at 12 **noon**!
- Lab1 to be released after Lab0P2 is due.
- Priority: Lab0P2 (tutorial videos + PDFs)
- This Thursday's office hour will be re-scheduled.

Product[] ps;
int nops;

Use of Accessors vs. Mutators


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

return —

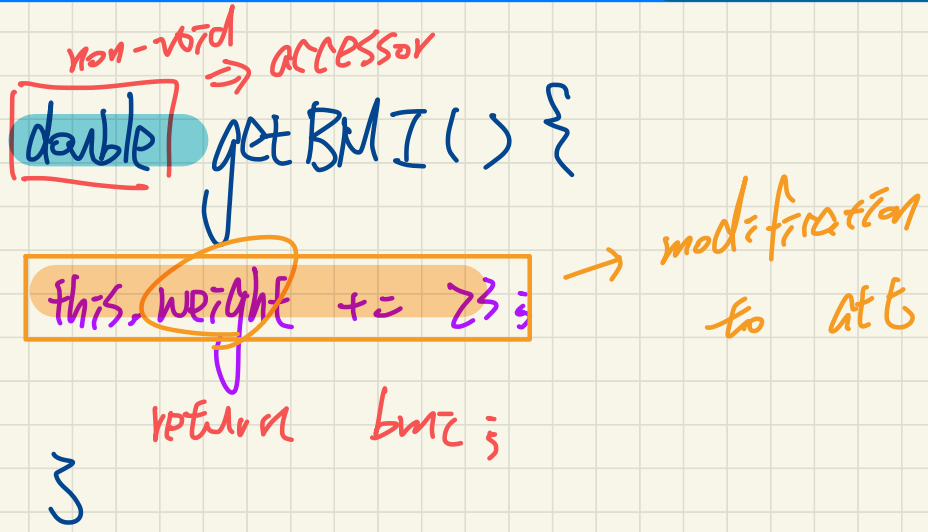
automatic garbage collection (automated mem. mgt.)

no return value

- 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();` → compiles! but leaving out the return value is not useful
  - e.g., `System.out.println(jim.getBMI());` ✓
  - e.g., `double w = jim.getBMI();` ✓

1. An accessor that's also a mutator

modifies some att.



2. A mutator that's an accessor is not possible.

? why.

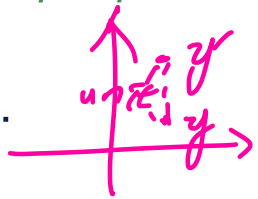
Method Parameters

- **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)`



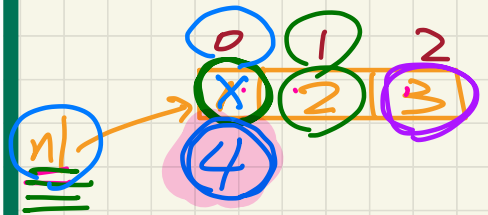
- **Principle 3:** An *accessor method* needs *input parameters* if the attributes alone are not sufficient for the intended computation to complete.

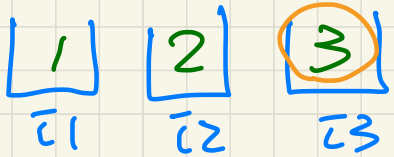
  e.g., In `Point`, `double getDistFromOrigin()` vs. `double getDistFrom(Point other)`

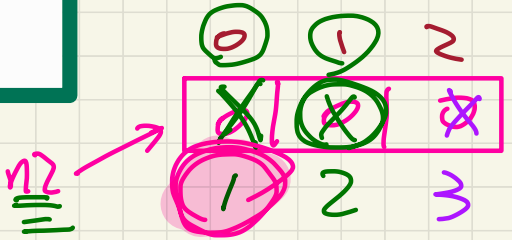
Copying Primitive Values

```
int i1 = 1;
int i2 = 2;
int i3 = 3;
int[] numbers1 = {i1, i2, i3};
int[] numbers2 = new int[numbers1.length];
for(int i = 0; i < numbers1.length; i ++) {
    numbers2[i] = numbers1[i];
}
numbers1[0] = 4;
System.out.println(numbers1[0]);
System.out.println(numbers2[0]);
```

alt.
int[] n2 = {n1[0], n1[1], n1[2]};







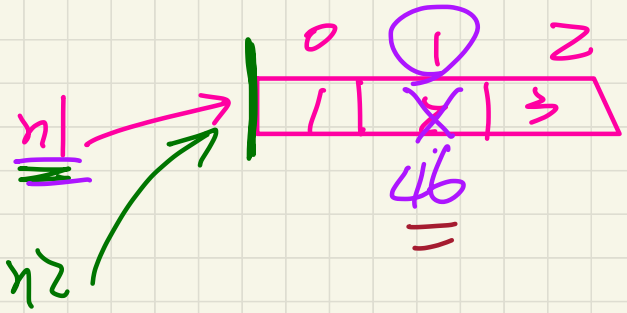
* How many iterations?

It #1 (i==0) n2[0] = n1[0];
It #2 (i==1) n2[1] = n1[1];
It #3 (i==2) n2[2] = n1[2];

```java
int i1 = 1;
int i2 = 2;
int i3 = 3;
int[] numbers1 = {i1, i2, i3};
int[] numbers2 = new int[numbers1.length];
for(int i = 0; i < numbers1.length; i++) {
  numbers2[i] = numbers1[i];
}
numbers1[0] = 4;
System.out.println(numbers1[0]);
System.out.println(numbers2[0]);
```

numbers1;

Copying the starting address of numbers1 to numbers2.



n1 == n2; T

n1[1] = 46;

n2[1] == 2; F

Copying Primitive vs. Reference Values

```
int i = 3;
int j = i;
int k = 3;
System.out.println(i == j);
System.out.println(k == i && k == j);
```

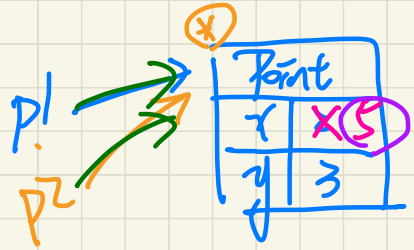
Primitive

② let p2 point to where p1 points to

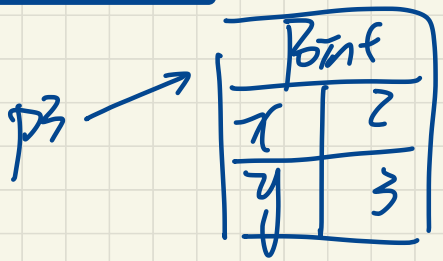
* ① copy the address stored in p1 into p2

```
Point p1 = new Point(2, 3);
Point p2 = p1;
Point p3 = new Point(2, 3);
Systme.out.println(p1 == p2);
Systme.out.println(p3 == p1 || p3 == p2);
Systme.out.println(p3.x == p1.x && p3.y == p1.y);
Systme.out.println(p3.x == p2.x && p3.y == p2.y);
```

Reference



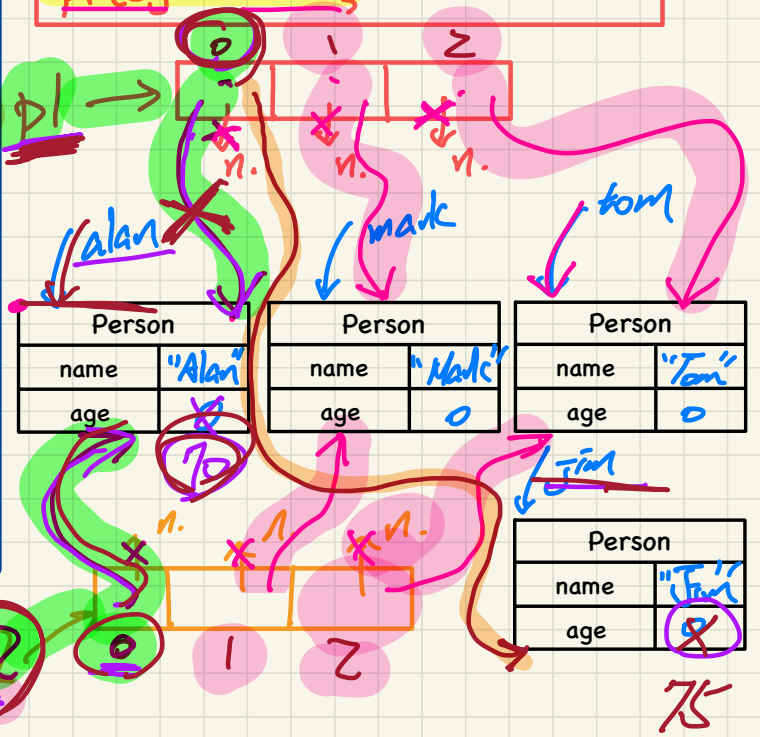
** p1.setX(5);
p2.getX(); 5



Copying Reference Values: Aliasing

```
Person alan = new Person("Alan");
Person mark = new Person("Mark");
Person tom = new Person("Tom");
Person jim = new Person("Jim");
*Person[] persons1 = {alan, mark, tom};
Person[] persons2 = new Person[persons1.length];
**for(int i = 0; i < persons1.length; i ++) {
  persons2[i] = persons1[i]; }
persons1[0].setAge(70);
System.out.println(jim.getAge());
System.out.println(alan.getAge());
System.out.println(persons2[0].getAge());
persons1[0] = jim;
persons1[0].setAge(75);
System.out.println(jim.getAge());
System.out.println(alan.getAge());
System.out.println(persons2[0].getAge());
```

* `Person[] p1 = new Person[3];`
`p1[0] = alan;` ① copy add. stored in alan to index 0 of p1.
`p1[1] = mark;`
`p1[2] = tom;`



** It #1 (i==0): `p2[0] = p1[0];`
It #2 (i==1) `p2[1] = p1[1];`
It #3 (i==2) `p2[2] = p1[2];`