Lecture 4 - Sep. 19

Review on OOP

Tracing OO Programs, Aliasing, Arrays
Attributes/Parameters/Return Types
Anonymous Objects

1. deadline
2. prog. req.

Announcements

- Lab1 released (scheduled lab sessions & office hours)
- Lab0 Part 2 Due on Friday
- WrittenTest1

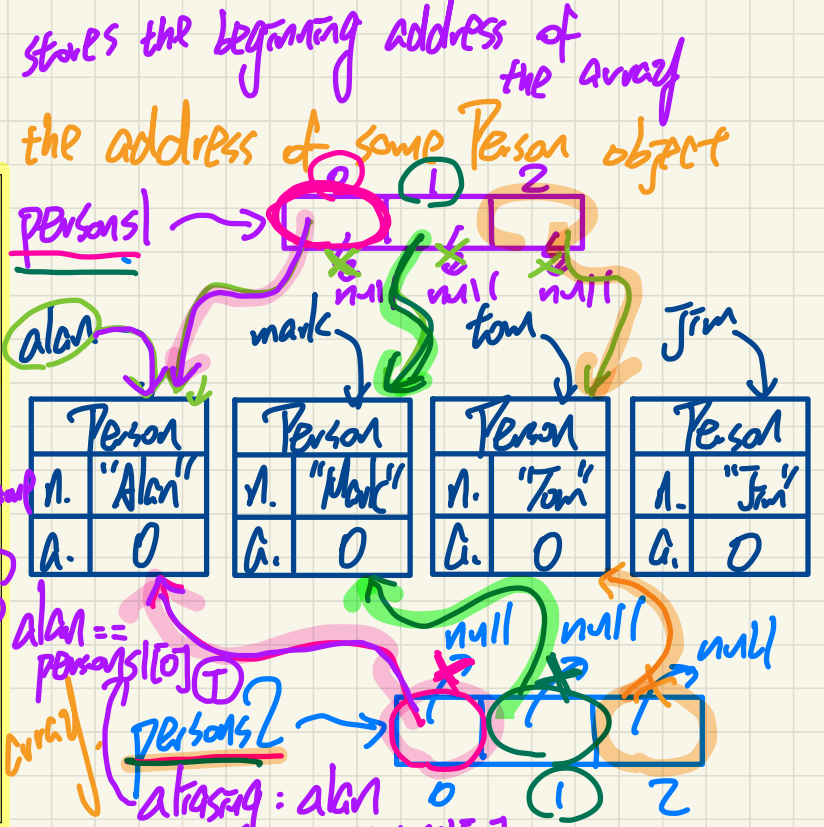
  (**make sure** you try logging into **eClass** in WSC)
- ProgTest1

Exercise

```
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  }
10 persons1[0].setAge(70);
11 System.out.println(jim.getAge());
12 System.out.println(alan.getAge());
13 System.out.println(persons2[0].getAge());
14 persons1[0] = jim;
15 persons1[0].setAge(75);
16 System.out.println(jim.getAge());
17 System.out.println(alan.getAge());
18 System.out.println(persons2[0].getAge());
```

Person[] persons1 = new Person[3]; ✓

persons1[0] = alan; // copy add. stored in alan to index 0.
persons1[1] = mark;
persons1[2] = tom;



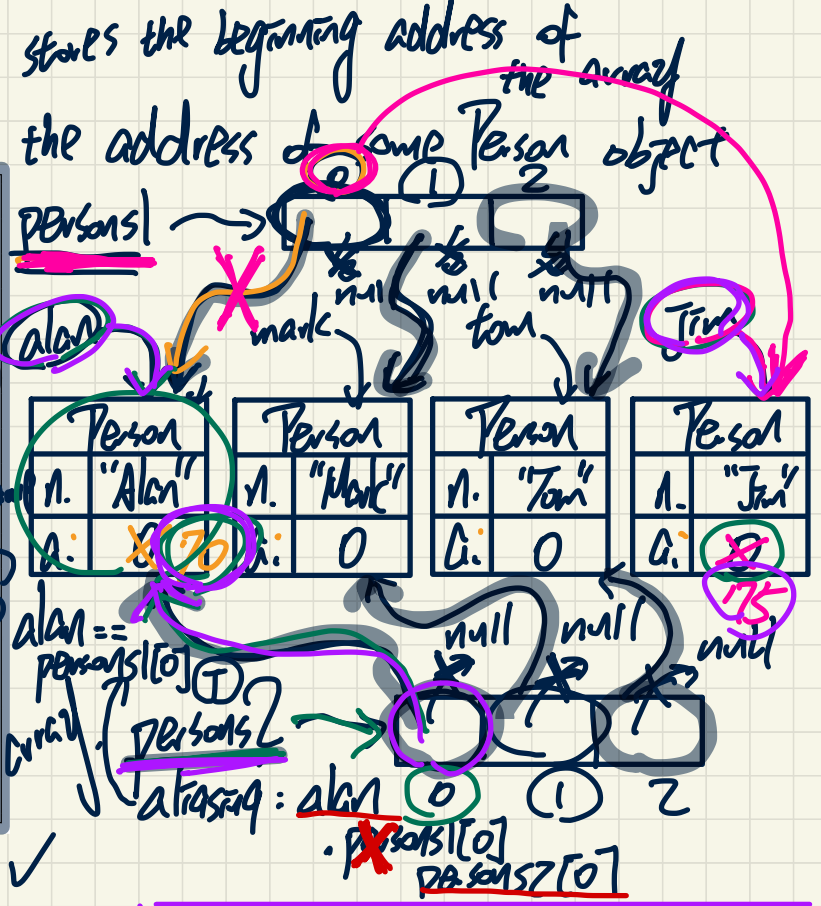
Exercise

Person[] persons1;

persons1 stores the beginning address of the array

each index of the array stores the address of some Person object

```
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  }
10 persons1[0].setAge(70);
11 System.out.println(jim.getAge());
12 System.out.println(alan.getAge());
13 System.out.println(persons2[0].getAge());
14 persons1[0] = jim;
15 persons1[0].setAge(75);
16 System.out.println(jim.getAge());
17 System.out.println(alan.getAge());
18 System.out.println(persons2[0].getAge());
```



Person[] persons1 = new Person[3]; ✓

→ persons1[0] = alan; // copy add. stored in alan to index 0.

→ persons1[1] = mark;

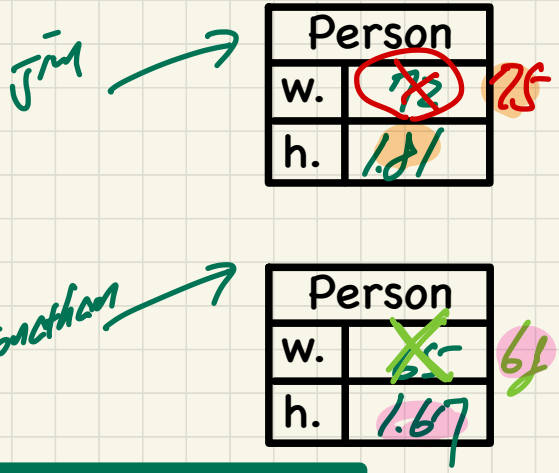
→ persons1[2] = tom;

alan.toString() → address

Accessors/Getters vs. Mutators/Setters

```java
public class Person {
    /*
     * Attributes.
     * Person instances have the same attribute names.
     * Person instances have specific attribute values.
     */
    double weight;
    double height;

    /* Accessors/Getters */
    public double getBMI() {
        double bmi = this.weight / (this.height * this.height);
        return bmi;
    }

    /* Mutators/Setters */
    public void gainWeightBy(double amount) {
        this.weight = this.weight + amount;
    }
}
```

```java
@Test
public void test_3() {
    Person jim = new Person(72, 1.81);
    Person jonathan = new Person(65, 1.67);

    assertEquals(21.977, jim.getBMI(), 0.01);
    assertEquals(23.307, jonathan.getBMI(), 0.01);

    jim.gainWeightBy(3);
    jonathan.gainWeightBy(3);

    assertEquals(22.893, jim.getBMI(), 0.01);
    assertEquals(24.382, jonathan.getBMI(), 0.01);
}
```

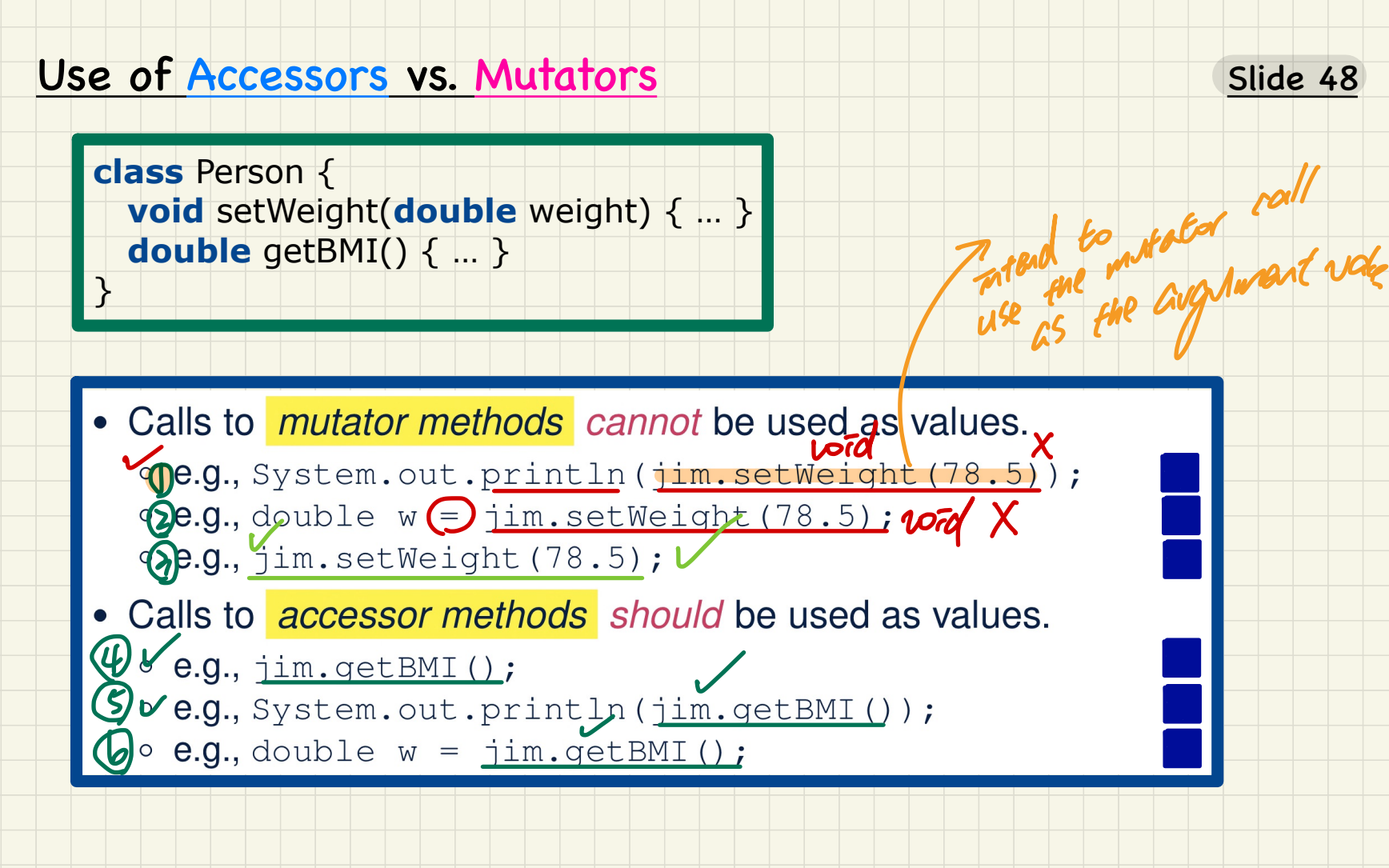


Object Oriented Programming (OOP)

- 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

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));` (void) X
  - e.g., `double w = jim.setWeight(78.5);` (void) X
  - 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

mainly for private helper methods

- **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)`

p1.getDFO(); p1.getDF(p2);

Reference-Typed Return Values

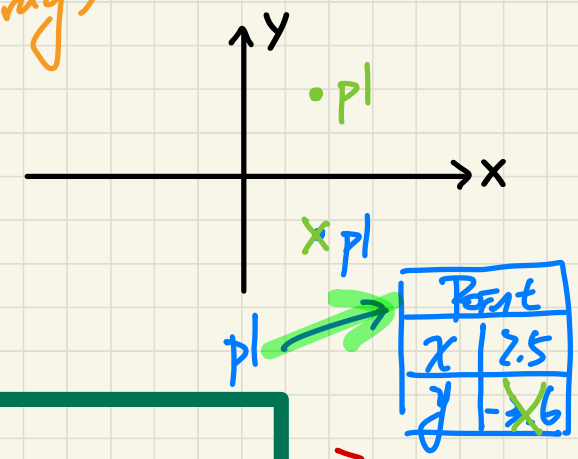
class MyClass {
  att;
}

1. primitive type
2. ref type
   - single-valued
   - multi-valued (array)

class Person {
  Person spouse;
  Person[] children;
}

```
public class Point {
  public void moveUpBy(int i) { y = y + i; }
  Point movedUpBy(int i) {
    Point np = new Point(x, y);
    np.moveUp(i);
    return np;
  }
}
```

```
public class PointTester {
  public static void main(String[] args) {
    Point p1 = new Point(2.5, -3.6);
    p1.moveUp(7.8);
    Point p2 = p1.movedUpBy(6.4);
    System.out.println(p1 == p2);
  }
}
```



- does not modify this
- modify some local var.

Anonymous Objects


```
1 double square(double x) {
2   double sqr = x * x;
3   return sqr; }
```

name

anonymous exp.

```
1 double square(double x) {
2   return x * x; }
```

```
1 Person getP(String n) {
2   Person p = new Person(n);
3   return p; }
```

name

```
1 Person getP(String n) {
2   return new Person(n); }
```

anonymous obj.

```
class Member {
  private Order[] orders;
  private int noo;
  /* constructor ommitted */
  public void addOrder(Order o) {
    this.orders[this.noo] = o;
    this.noo++;

  public void addOrder(String n, double p, double q) {
    [redacted]
  }
}
```

overloading - Exercise

① treat this as a helper method.

Lab0 P2

this.addOrder(new Order(---));

Order o = new Order(n,p,q);

this.orders[this.noo] = o;
this.noo++;

this.addOrder(5);

dup.