Lecture 4 - May 15

Review of OOP

Visualizing Objects
Tracing OOP: Creations, Method Calls
Reference Aliasing
Anonymous Objects

Announcements/Reminders

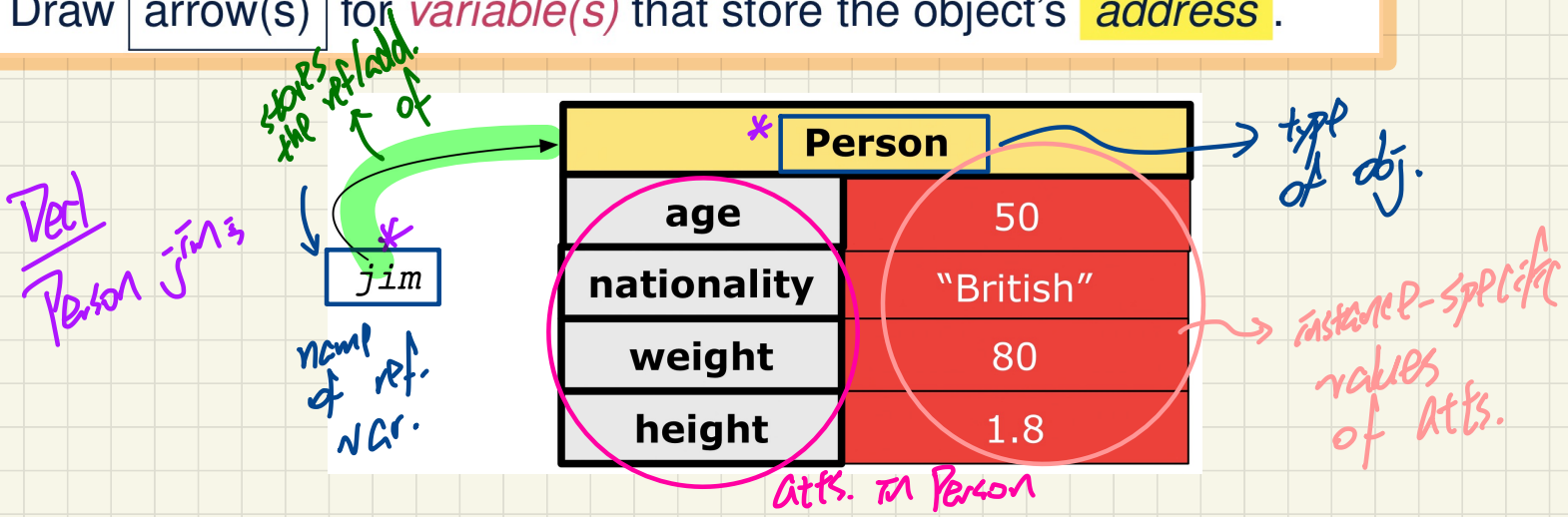
- Today's class: notes template posted
- ProgTest0 next Friday (May 23)
- Lab this Friday (May 16):
  - + [≈ 40 min] Mock-up ProgTest0
  - + [≈ 40 min] Q&As (TAs, Jackie)
- Priorities:
  - + LabOP2 (tutorial videos + PDFs)
  - + Due: Next Monday
- Lab1 to be released next Monday

download
Eclipse
import
develop/JUnit
export
submit

Tracing OO Code: Visualizing Objects

To visualize an object:

- Draw a **rectangle box** to represent *contents* of that object:
  - **Title** indicates the *name of class* from which the object is instantiated.
  - **Left column** enumerates *names of attributes* of the instantiated class.
  - **Right column** fills in *values* of the corresponding attributes.
- Draw **arrow(s)** for *variable(s)* that store the object's *address*.



Effects of Creating New Objects

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

    /*
     * Constructors
     */
    public Person() {

    }

    public Person(double weight, double height) {
        this.weight = weight;
        this.height = height;
    }
}
```

model

- Variable Shadowing
- Visualizing Objects
- Context Object
- this
- dot notation

```
@Test
public void test_1() {
    Person jim = new Person(72, 1.81);
    Person jonathan = new Person(65, 1.67);
    assertFalse(jim == jonathan);
    assertNotSame(jim, jonathan);
    assertNotEquals(jim, jonathan);
}
```

JUnit

pass

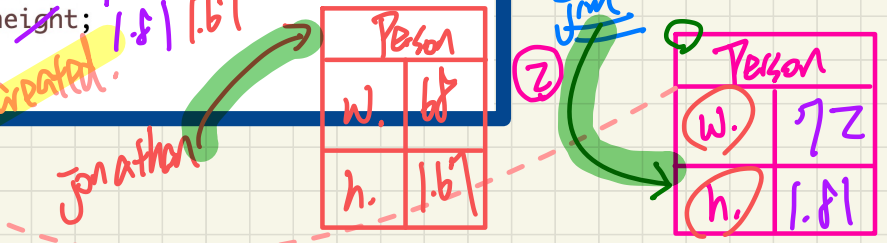
false

* Person jim = new Person(72, 1.81);

① ② ③

① declares a ref. var. "jim" that stores add/ref of some Person obj.

③ store add. of new object to ref. var. "jim"



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

    /*
     * Constructors
     */
    public Person() {

    }

    public Person(double weight, double height) {
        this.weight = weight;
        this.height = height;
    }
}
```

int weight

double

int

int t = 23;
double d = t;

coercion

23.0

d

d = 46.23;
t = d; X

46.23

d

t = (int) d;

46

t

JUnit assertions : pass vs. fail

boolean expression : True vs. False

assertTrue (true) → pass assertFalse (true) → fail

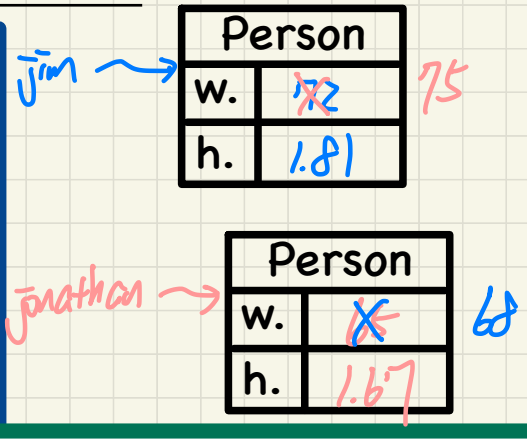
assertTrue (false) → fail assertFalse (false) → pass

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);
}
```

assertEquals(v1, v2, 0.01)

↳ pass if: $|v1 - v2| \leq$ 0.01 (ε tolerance)

Use of Accessors vs. Mutators


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

return ...;

- 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();` ✓

nothing returned

not called in a context where a return value expected

compiles, but not useful

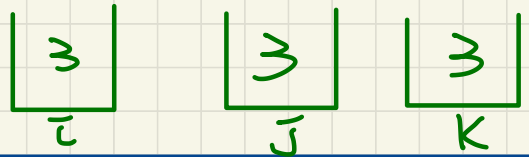
Copying Primitive vs. Reference Values

* store the address in p1 into p2 → p1 and p2 store the same add.

Primitive

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

and || or



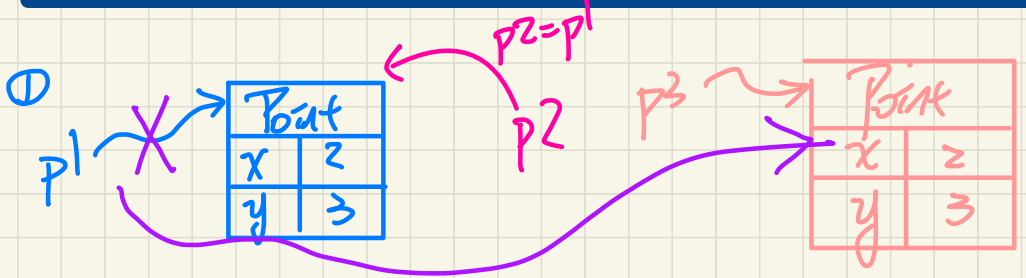
① do p1 and p2 store the same add?
② do p1 and p2 point to the same obj?

Reference

```
① Point p1 = new Point(2, 3);
Point p2 = p1;   System.out.println(p1 == p2);
Point p3 = new Point(2, 3);
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);
```

T

false false



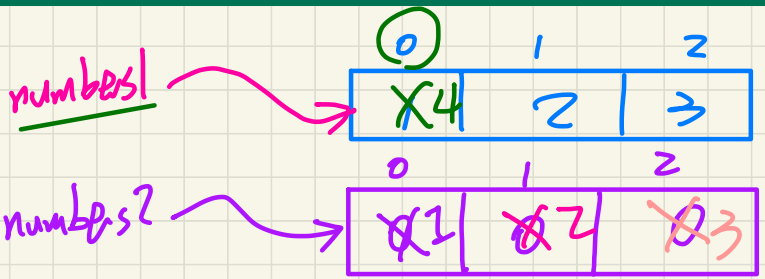
p1 = p3;

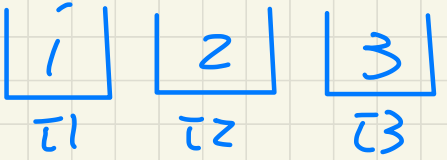
p1 points to wherever p3 points to

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]);
```

steps the beginning add 1 away





i

0 n2[0] = n1[0];

1 n2[1] = n1[1];

2 n2[2] = n1[2];

✗ i < n1.length is (F).

Anonymous Objects


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

named exp

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

anonymous exp.

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

named obj.

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

anonymous obj.

Exercise

```
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) {
    Order no = new Order(n, p, q);
    this.orders[this.noo] = no;
    this.noo++;
  }
}
```