Lecture 4 - Sep 15

OOP Review

Tracing OOP: New Obj. & Method Calls
Variable Shadowing, Use of this
Reference Aliasing

Announcements/Reminders

- Today's class: notes template posted
- Priorities:
  - + Lab0P2 (tutorial videos + PDFs)
  - + Due: This Tuesday (Sep 16)
- Lab1 to be released this Tuesday (Sep 16)
- Lab last Wednesday (Sep 10): Demo on helper methods
- ProgTest0 (next Wednesday, Sep 24), Guide released
- Lab this Wednesday (Sep 17):
  - + Please go to your enrolled session.
  - + [≈ 40 min] Mock-up Programming Test 0
  - + [≈ 40 min] Q&As (TAs, Jackie)

Constructors not using this Keyword

model

```
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 newWeight, double newHeight) {
        weight = newWeight;
        height = newHeight;
    }
}
```

Use "this" keyword to denote the context object (which allows you to ref. attributes)

Question

- What if names of parameter & attribute are the same?
- implicit "this"

clashes with the attribute

weight height

this

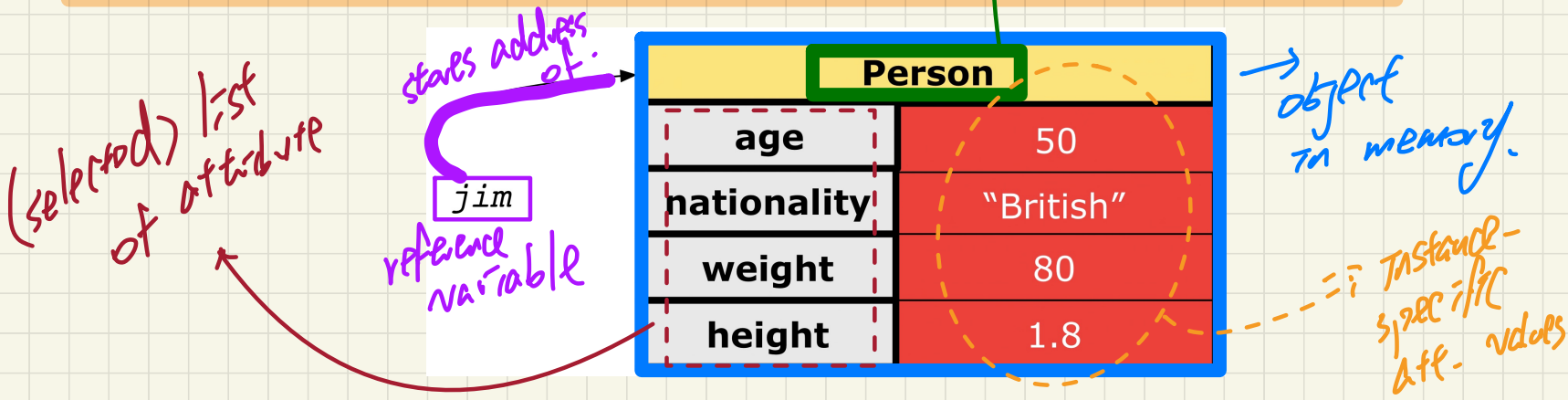
not referring to the intended attribute (it's shadowed by the parameter name)

(variable shadowing).

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*.



to different objects.

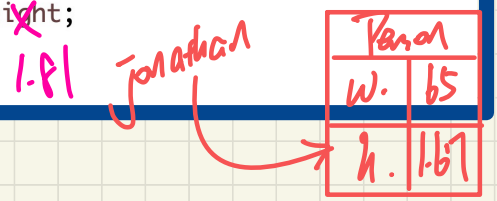
Effects of Creating New Objects

model

```
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;
    }
}
```

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

72 1.81



JUnit

```
@Test
public void test1() {
    Person jim = new Person(72, 1.81);
    Person jonathan = new Person(65, 1.67);
    assertTrue(jim != jonathan);
    assertFalse(jim == jonathan);
    assertNotSame(jim, jonathan);
    assertNotEquals(jim, jonathan);
}
```

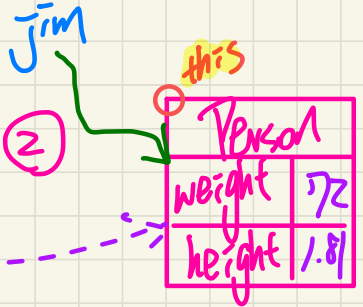
false

jim and jon. point *

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

① ② ③

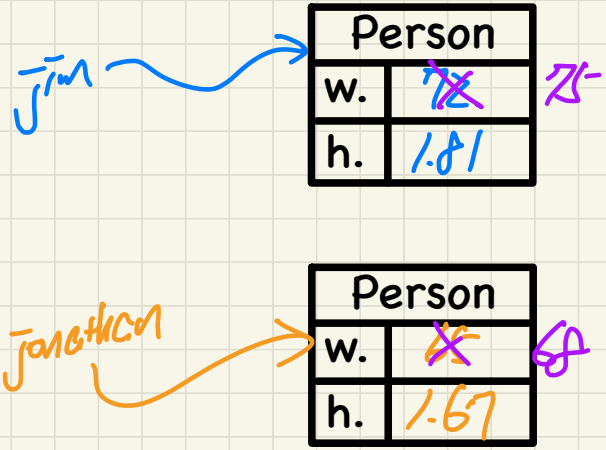
① declares a ref. var "jim" that stores add./ref. of some Person object



③ stores add. of new object to the ref. var.

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);
}
```

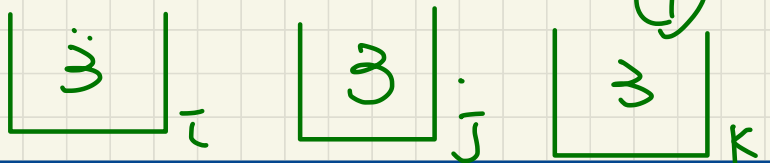
Copying Primitive vs. Reference Values

Primitive

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

and → T

T T



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

F or → F F

p3 →

Point	
x	2
y	3

* copy the ref./add. in p1 and store it into p2.

① p1 →

Point	
x	2
y	3

← p2

** p1 and p2 store the same address
↳ they're aliases of the same object.

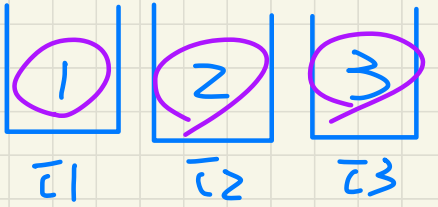
Copying Primitive Values


```
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]);
```

stores the starting add. of an int array

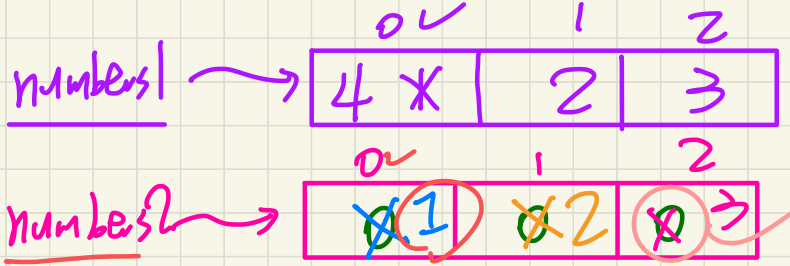
4

1.



i
0
1
2
3 no iteration

ns2[0] = ns1[0]
ns2[1] = ns1[1]
ns2[2] = ns1[2]



3 < ns1.length
3 < 3
F

default value for int is 0 !!
not null