Lecture 3 - Sep. 12

Review of OOP

Use of this for Attribute Access
Tracing OO Code
Use of Mutators vs. Accessors
Design of Method Parameters
Reference Aliasing

Announcements/Reminders

- LabOP1 due tomorrow (Friday) at 12 noon!
- Priority: LabOP2 (tutorial videos + PDFs)
- Yesterday's in-lab demo materials (helper methods) released.

Constructors not using this Keyword

model

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

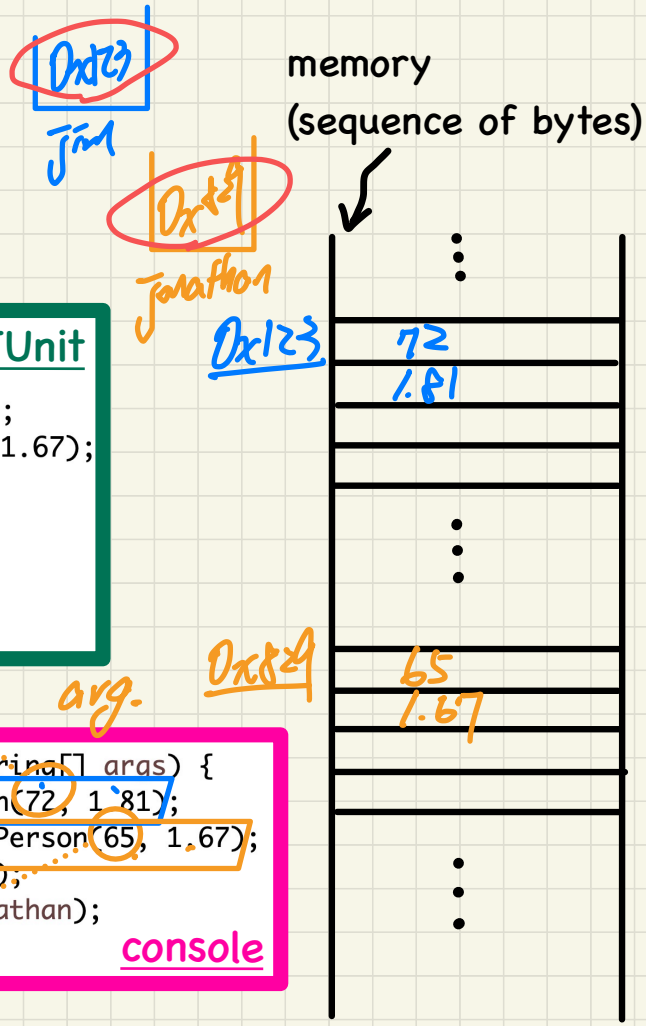
JUnit

```java
@Test
public void test_1() {
    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);
}
```

console

```java
public static void main(String[] args) {
    Person jim = new Person(72, 1.81);
    Person jonathan = new Person(65, 1.67);
    System.out.println(jim);
    System.out.println(jonathan);
}
```

- Default Constructor?
- Parameters vs. Arguments
- Reference Variables



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;
    }
}
```

Question

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

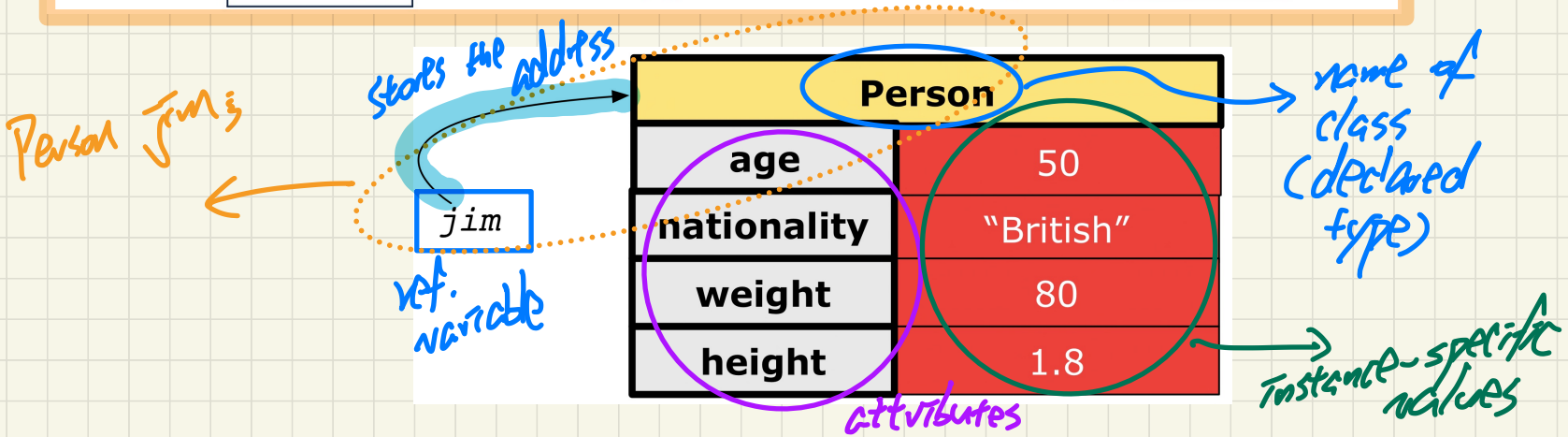
(variable) shadowing

1. does not initialize att. weight
2. instead, parm w. is assigned to itself.

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

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

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);
}
```

Person jim = new Person(72, 1.81);

① ② ③

① declaring a ref var.

jim

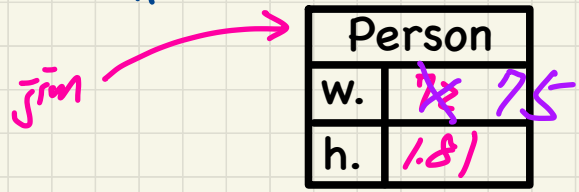
② creating an anonymous obj

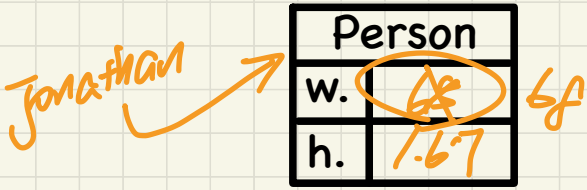
Person	
w.	72
h.	1.81

indicates that the class-level "weight" is refered to.

Accessors/Getters vs. Mutators/Setters

$bmi = \frac{w}{h^2}$






```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);
}
```

Use of Accessors vs. Mutators


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

mutator

accessor

- 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 the return val not used! ■
  - e.g., `System.out.println(jim.getBMI());` ✓
  - e.g., `double w = jim.getBMI();` ✓

void

void

X

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)`

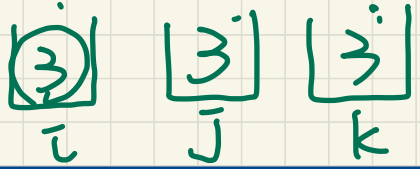
- **Principle 3:** An *accessor method* needs *input parameters* if the attributes alone are not sufficient for the intended computation to complete.

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

Copying Primitive vs. Reference Values

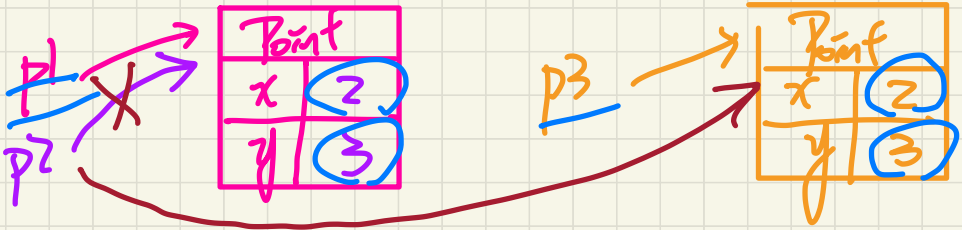
Primitive

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



Reference

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



p2 = p3 = 3

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]);
```