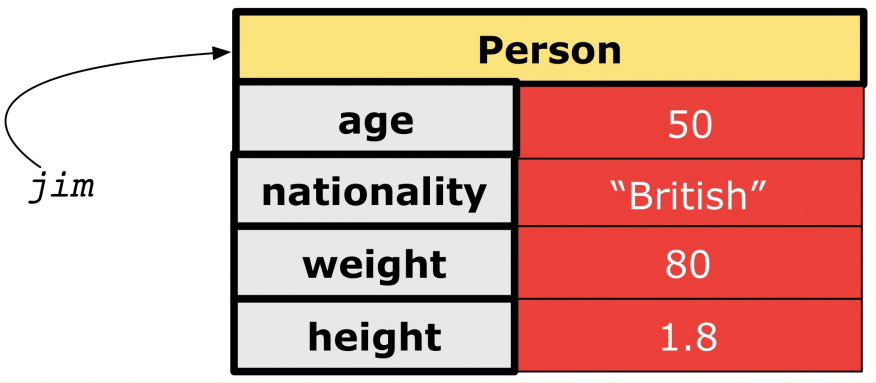
Tracing OO Code: Visualizing Objects

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

```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;
    }
}
```

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

JUnit

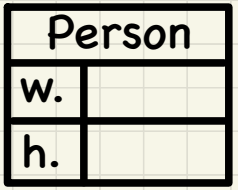
```java
@Test
public void test_1() {
    Person jim = new Person(72, 1.81);
    Person jonathan = new Person(65, 1.67);
    assertTrue(jim != jonathan);
    assertFalse(jim == jonathan);
    assertNotSame(jim, jonathan);
    assertNotEquals(jim, jonathan);
}
```

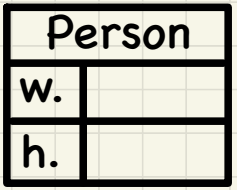
Accessors/Getters vs. Mutators/Setters

```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);
}
```

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));`
  - 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();`

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

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

Primitive

```
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);
```

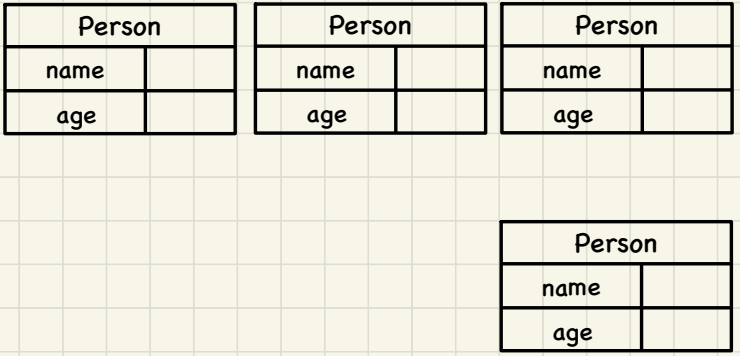
Reference

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]);
```

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());
```



Reference-Typed Return Values


```
public class Point {
   /* A mutator modifying the context Point object */
   public void moveUp (double i) {
      this.y = this.y + i;
   }

   /* An accessor returning a new Point object */
   public Point movedUpBy(double i) {
      Point np = new Point(this.x, this.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);
   }
}
```

Anonymous Objects


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

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

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

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

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) {



  }
}
```

Example: Reference to this


```
public class Person {
  private String name;
  private Person spouse;
  public Person(String name) {
    this.name = name;
  }
  public void marry(Person other) {



  }
}
```

```
Person jim = new Person("Jim");
Person elsa = new Person("Elsa");
jim.marry(elsa);
```