2.3 Reference Semantics in Java

- Java, C#, Visual Basic, and Python employ reference semantics for "objects".
  - Value semantics is employed only built-in simple types, such as integers and real numbers, that are not class types.

    [a few exceptions in some of the languages, but not in Java]

- To understand reference semantics, we must understand the difference between an

  i) an object, and

  ii) an object reference.

- Consider this Java code (in some method).


```
Point a;
```

Create a new object reference **a** (on the run-time stack).

[*1 object reference, 0 objects*].

```
a = new Point(3,4);
```

Create a new object (on the heap). This object is anonymous. (We will call it Object #1.)

Assign **a** to refer to Object #1.

[*1 object reference, 1 object*].

```
Point b = a;
```

Create a new object reference **b** (on the run-time stack).

Assign **b** to refer to object #1.

[*2 object references, 1 object*].

```
Point c = new Point(7,5);
```

Create a new object reference **c** (on the run-time stack).

Create a new object (on the heap). We call it object #2.

Assign **c** to refer to object #2.

[*3 object references, 2 objects*].

```
a = c;
```

Assign **a** to refer to object #2.

[*3 object references, 2 objects*].


```
b = new Point(7,5);
```

Create a new object (on the heap). We will call it object #3.

Assign **b** to refer to object #3.

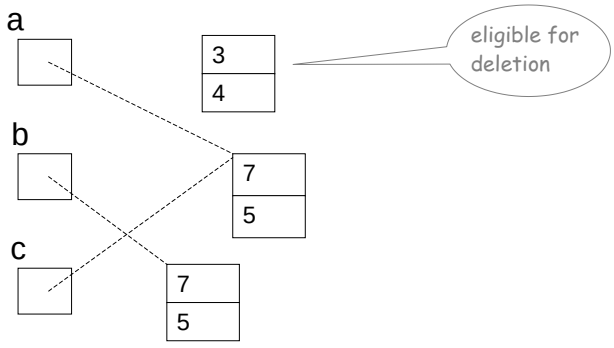
[*3 object references, 3 objects; but object #1 is inaccessible, and subject to deletion by the system*]

```
System.out.println( (a == b) ? 'e' : 'n' );
```

Prints 'n'. The equality operator compares object references, not objects. a refers to object #2; b to object #3.

```
System.out.println( (a == c) ? 'e' : 'n' );
```

Prints 'e'. a and c refer to the same object (object #3).



- Some differences between objects and object references, in Java.

	Object references	Objects
Where allocated:	On the run-time stack (unless object reference is an element of an array, or a field within an object).	On the heap.
Amount of memory:	Independent of the type of the object (typically 4 bytes).	Depends on type of the object.
Name:	Has a name (unless it is an element of an array, or a field within an object).	Always anonymous
Refers to: / Referred to by:	Refers to a single object, unless it is null.	Zero or more object references may refer to it. If 0, the object is eligible for deletion.
Type(s):	Has two types: (i) a declared type (static type), and (ii) an actual type (dynamic type).	Has only one type (actual type = declared type).

May contain:		May contain i) primitive types, ii) array references, iii) object references. May not contain i) arrays, ii) other objects.
Modified by:	Assigning to the object reference.	Invoking a mutating method of the object.
Modification disallowed by:	Declaring the object reference as final.	Defining no mutating methods in the object's class.
When deleted:	When control leaves the block in which it defined. However, if the object reference is an element of an array, or field within an object, it is deleted when the containing array or object is deleted.	By the system, when the garbage collection process detects that no references the object remain.

- The actual type of an object reference is the type of the object to which it refers.
  - The actual type must be a subclass of the declared type.
  - For example, in

    ```
    Token t = new BinaryOp();
    ```

    the declared type of `t` is `Token`, and the actual type is `BinaryOp`.