Sunday, December 30, 2018 1:53 PM

Reference Semantics

Primitives are stored directly in memory, so when you copy a primitive variable the value is copied, leaving the original variable unaffected. This is called value semantics.

This means that any variable holding an object (like an array) actually stores a reference to the object. Therefore, when you make a copy of an object variable, you copy the reference, which still points at the original data. This is called reference semantics.

Array Reference Semantics

```
int a[] = new int[5]; // [0, 0, 0, 0, 0]
a[0] = 10;          // [10, 0, 0, 0, 0]
int b[] = a;        // [10, 0, 0, 0, 0]
b[0] = 5;           // [5, 0, 0, 0, 0]
System.out.println("a = " + Arrays.toString(a));
System.out.println("b = " + Arrays.toString(b));
```

The two printlns at the end of this code would produce the following output:

```
a = [5, 0, 0, 0, 0]
b = [5, 0, 0, 0, 0]
```

Arrays as Parameters

```
public static void main(String[] args) {
   int[] a = new int[5];
   System.out.println("before method: " + Arrays.toString(a));
   myMethod(a);
   System.out.println("after method: " + Arrays.toString(a));
}

public static void myMethod(int[] b) {
   b[0] = 5;
}
```

The output of the above code would be:

.
before method: [0, 0, 0, 0, 0]
after method: [5, 0, 0, 0, 0]

create a new variable "b" that stores an array of the exact same values as a different array "a", but without affecting the original array
b = Arrays.copyOf(a, a.length);