EECS2030 Advanced Object-Oriented Programming (Fall 2021)

Q&A - Lecture 4

Wednesday, October 27

Announcement

- Lecture W7 (released: Oct. 25) → inheritance.
- Lab3 (released: Oct. 20; due: Nov 1)
- Written Test 2 (due: Oct. 28/29)

2pm EST

If the variable used in the reassignInt method was i instead of j then would i have changed to be 11? (Part B1; Timestamp 7:30)

```
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

not compiling

```
1  @Test
2  public void testCallByVal() {
3    Util u = new Util();
4    int i = 10;
5    assertTrue(i == 10);
6    u.reassignInt(i);
7    assertTrue(i == 10);
8  }
```

So to make a deep copy,
make new objects based on the originals
instead of making aliases of originals? YES

```
class File {
  File(File other) {
    this.name =
      new String(other.name);
  }
}
```

copy constructor of String class

```
class Directory {
  Directory(String name) {
    this.name = new String(name);
    files = new File[100]; }
  Directory(Directory other) {
    this (other.name);
    for(int i = 0; i < nof; i ++) {
      File src = other.files[i];
      File nf = new File(src);
      this.addFile(nf); } }
  void addFile(File f) { ... } }
```

with this: files[i] = other.files[i]

```
class A {
  int i;
  A(A other) {
  }
}
```

- To make a deep copy:

① primitive attributes
use assignment op.
this.i = other.i

② reference attribute
use assignment op + copy const.
(this.name = new String(othe.n))

Is there any specific lines that differ from aggregation to composition, is it creating a new object for each index of an array (new File) instead of saying new Directory?

Exercise: Implement the accessor in class `Directory`

```
class Directory {
  File[] files;
  int nof;
  File[] getFiles() {
    /* Your Task */
  }
}
```

1. return a new array File[]
2. Each element of the return array should:
   ① point to a deep copy of the file object
   ② not be a mere alias to the file object

copy constructor.

shallow copy

so that it ***preserves composition***, i.e., does not allow references of files to be shared.

In professional practice, what's more typical for getter functions returning ref.-type objects or arrays of ref.-type objects: returning shallow copies or returning deep copies?

↳ There's no right or wrong answer.

① If the app. requirement is that composition is preserved, then deep copy constructors at different levels.

② Otherwise, to save space, a shallow copy via aliases (allowing sharing) suffices.