Lecture 15 - Nov 2

Composition, Inheritance

Dotted Notation vs. Private Attributes
Compositions
The Student Management Problem

Announcements

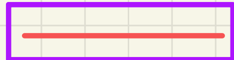
- ProgTest1: Visit office hours to discuss your solution
- Lab3 due next Wednesday (equals & copy constructor)
- WrittenTest2 to be released by early Friday
- ProgTest2: guide to be released soon

↳ Lab2.

Dot Notation: Private Attributes/Fields

Principle: **Private** attribute is accessible if the context object's type matches the context class (where the method is defined).

```
public class A {
    private B ob;
    private int ai;
    public B getB() { return this.ob; }
    public int getAi() { return this.ai; }
    public int am() {
        int result;
        result = this.ai;
        result = this.getAi();
        result = this.ob.bi;
        result = this.getB().bi;
        result = this.ob.getBi();
        result = this.getB().getBi();
        result = this.ob.getA().ai;
        result = this.ob.getA().getAi();
        result = this.ob.oa.ai;
        result = this.ob.oa.getAi();
        return result;
    }
}
```

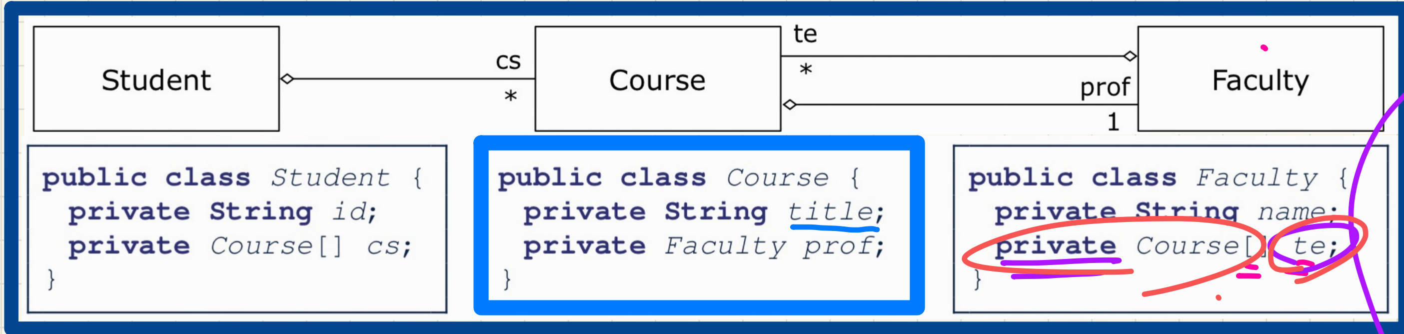


```
public class B {
    private A oa;
    private int bi;

    public A getA() {
        return this.oa;
    }

    public int getBi() {
        return this.bi;
    }
}
```

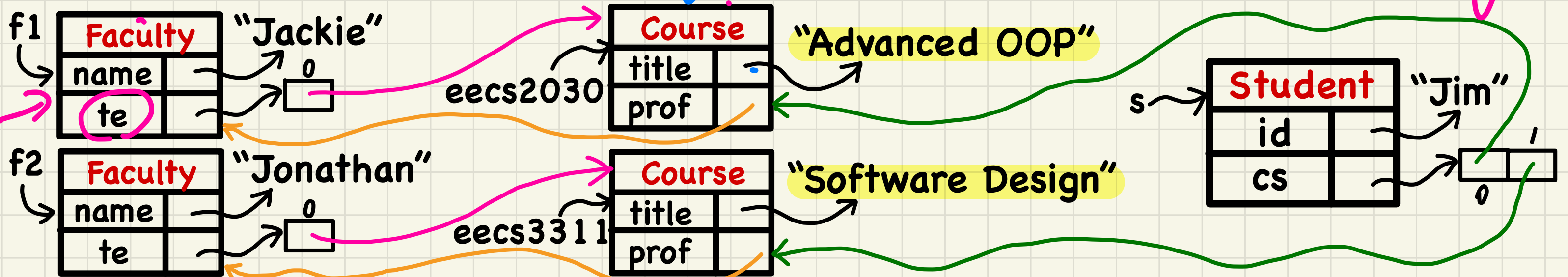
Dot Notation for Navigating Classes (2)




```
/* Get course's title.
 */
String getTitle() {
  return this.title;
}
```

```
/* Name of instructor
 */
String getName() {
  return this.getProf().getName()
}
```

```
/* Title of instructor's
 * ith teaching course
 */
String getTitle(int i) {
  return this.getProf().getTe()[i].getTitle();
}
```



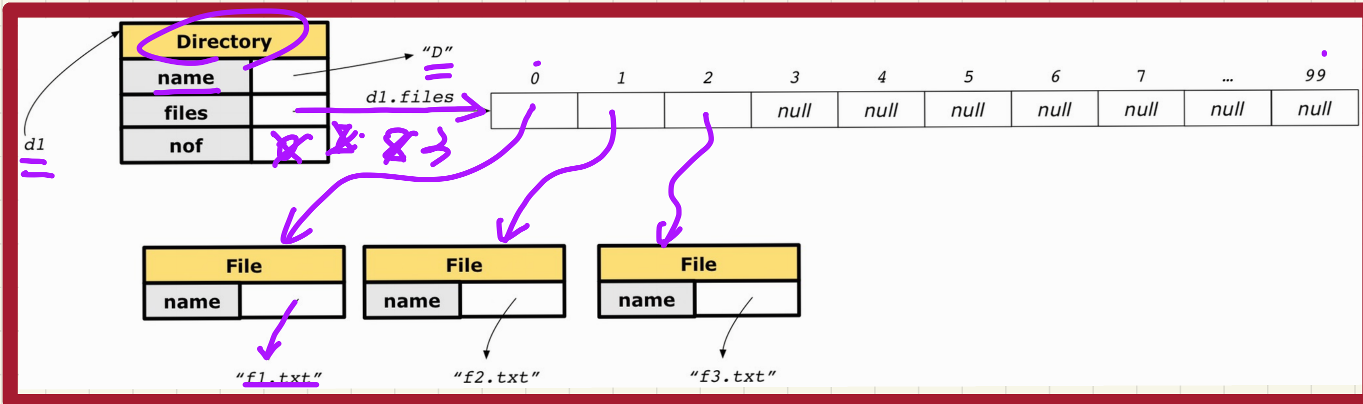
Composition: No Sharing

```
class Directory {
  String name;
  File[] files;
  int nof; /* num of files */
  Directory(String name) {
    this.name = name;
    files = new File[100];
  }
  void addFile(String fileName) {
    files[nof] = new File(fileName);
    nof ++;
  }
}
```

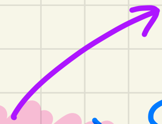
```
class File {
  String name;
  File(String name) {
    this.name = name;
  }
}
```

```
public File[] getFiles() {
  return this.files;
}
```

```
1  @Test
2  public void testComposition() {
3    Directory d1 = new Directory("D");
4    d1.addFile("f1.txt");
5    d1.addFile("f2.txt");
6    d1.addFile("f3.txt");
7    assertTrue(
8      d1.files[0].name.equals("f1.txt"))
9  }
```



class X {



copy constructor

X (X other) {

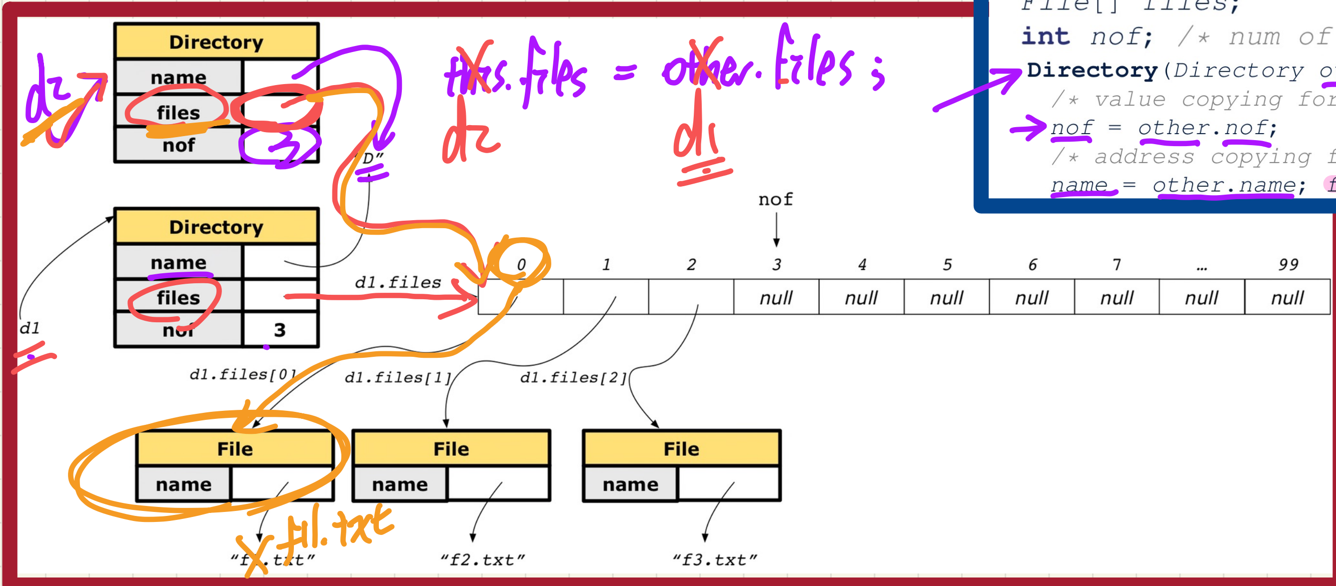
}

}

Composition: Copy Constructor (Shallow Copy)

```java
@Test
public void testShallowCopyConstructor() {
  Directory d1 = new Directory("D");
  d1.addFile("f1.txt"); d1.addFile("f2.txt"); d1.addFile("f3.txt");
  Directory d2 = new Directory(d1);
  assertTrue(d1.files == d2.files); /* violation of composition */
  d2.files[0].changeName("f11.txt");
  assertFalse(d1.files[0].name.equals("f1.txt")); }
```

```java
class Directory {
  String name;
  File[] files;
  int nof; /* num of files */
  Directory(Directory other) {
    /* value copying for primitive type */
    nof = other.nof;
    /* address copying for reference type */
    name = other.name; files = other.files; }
```



Composition: Copy Constructor (Deep Copy)

```
@Test
public void testDeepCopyConstructor() {
  Directory d1 = new Directory("D");
  d1.addFile("f1.txt"); d1.addFile("f2.txt"); d1.addFile("f3.txt");
  Directory d2 = new Directory(d1);
  assertTrue(d1.files != d2.files); /* composition preserved */
  d2.files[0].changeName("f11.txt");
  assertTrue(d1.files[0].name.equals("f1.txt")); }
```

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

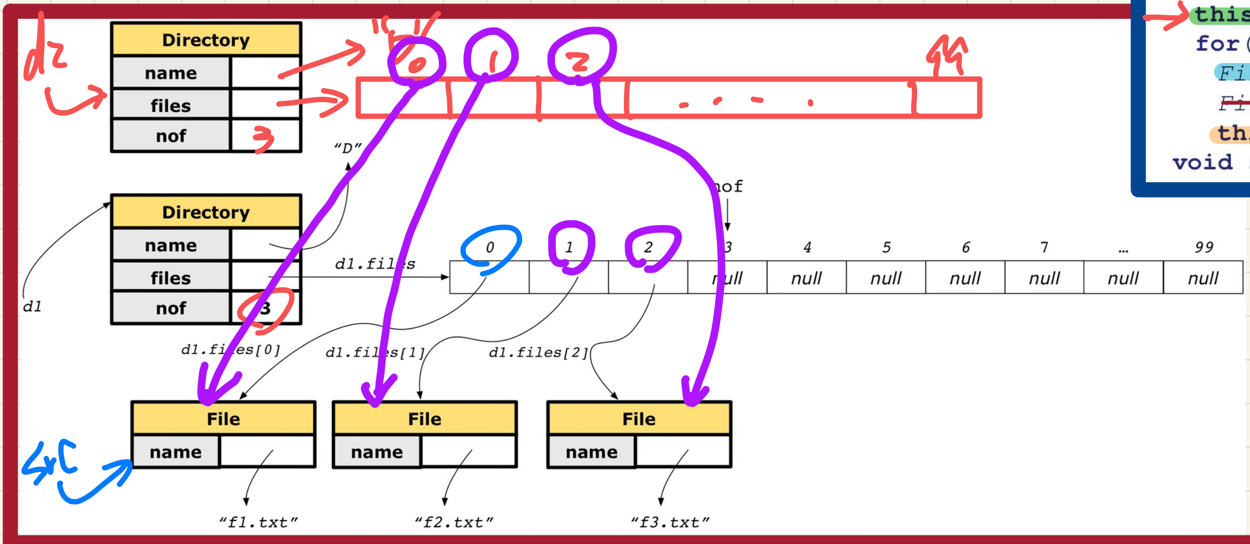
```
class Directory {
  Directory(String name) {
    this.name = new String(name);
    files = new File[100]; }
  Directory(Directory other) {
    this (other.name);
    for(int i = 0; i < nof; i ++) {
      File src = other.files[i];
      File nf = new File(src);
      this.addFile(nf); } }
  void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

Exercise: Copy Constructor (Composition?)

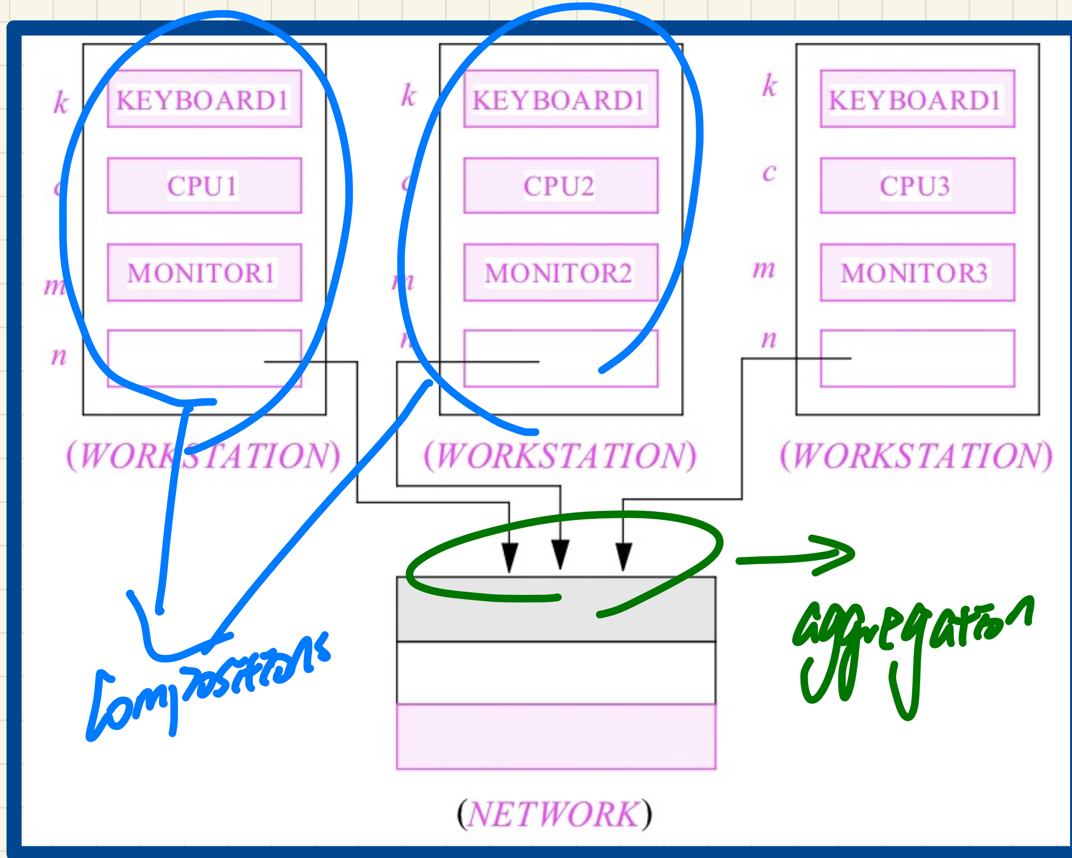
```
@Test
public void testDeepCopyConstructor() {
  Directory d1 = new Directory("D");
  d1.addFile("f1.txt"); d1.addFile("f2.txt"); d1.addFile("f3.txt"
  Directory d2 = new Directory(d1);
  assertTrue(d1.files != d2.files);
  d2.files[0].changeName("f11.txt");
  assertTrue(d1.files[0] == d2.files[0]);
```

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

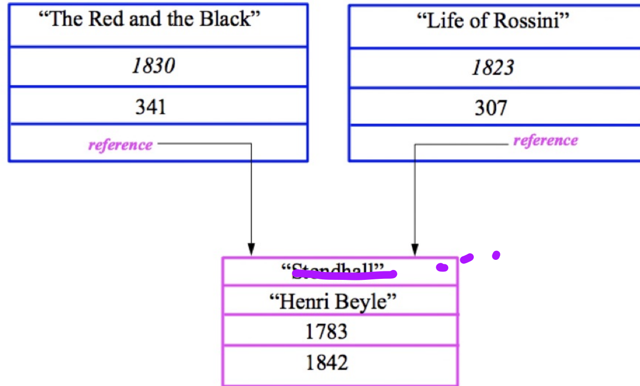
```
class Directory {
  Directory(String name) {
    this.name = new String(name);
    files = new File[100]; }
  Directory(Directory other) {
    this (other.name);
    for(int i = 0; i < nof; i ++) {
      File src = other.files[i];
      File nf = new File(src);
      this.addFile(nf); } }
  void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

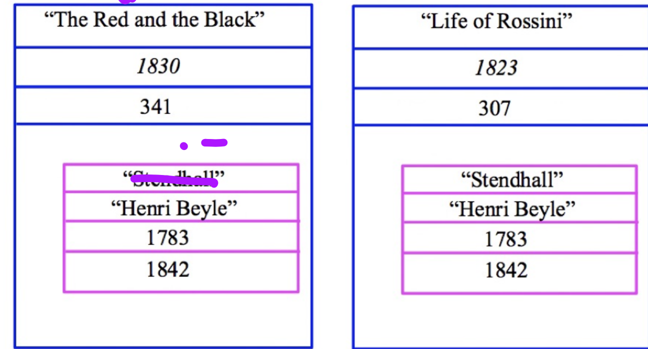


Modelling: Aggregation vs. Composition



Implementation: Aggregation or Composition





Hyperlinked author page

Physical printed copies

Inheritance: Motivating Problem

Nouns -> classes, attributes, accessors

Verbs -> mutators

Student[ ]

Problem: A student management system stores data about students. There are two kinds of university students: resident students and non-resident students. Both kinds of students have a name and a list of registered courses. Both kinds of students are restricted to register for no more than 10 courses. When calculating the tuition for a student, a base amount is first determined from the list of courses they are currently registered (each course has an associated fee). For a non-resident student, there is a discount rate applied to the base amount to waive the fee for on-campus accommodation. For a resident student, there is a premium rate applied to the base amount to account for the fee for on-campus accommodation and meals.

depends on the kind of student.

should not apply simultaneously!