

Lecture 15 - Oct. 31

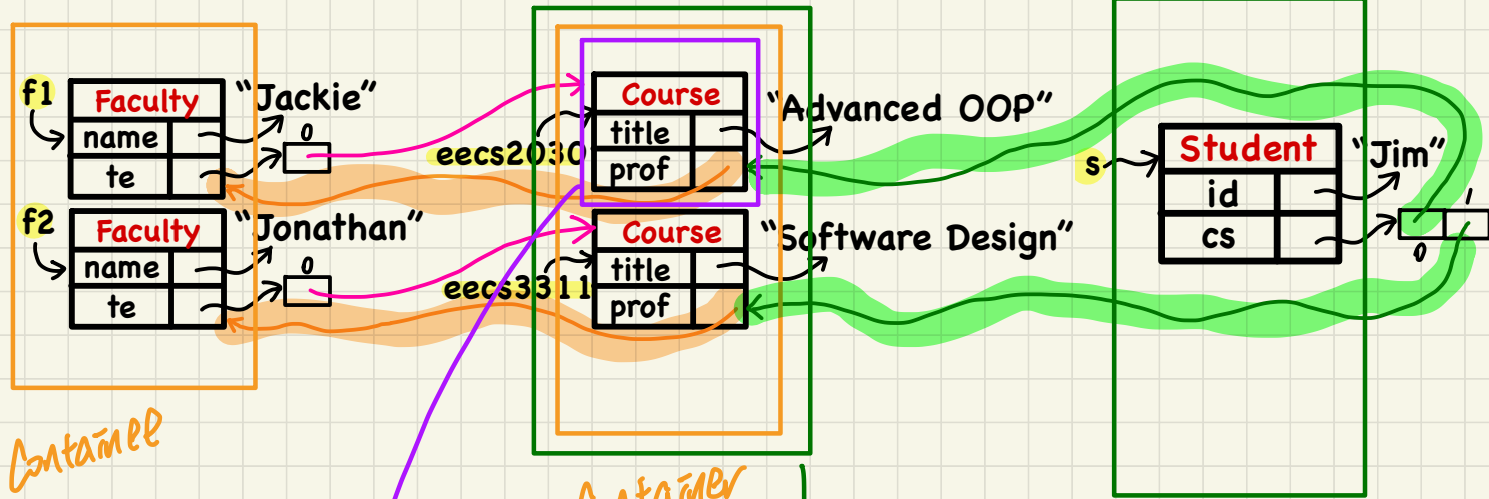
Aggregation and Composition

Modelling: Aggregation vs. Composition
Dot Notation Exercise
Copy Constructor, Shallow Copy

Announcements/Reminders

- **Lab3** due tomororw at noon
- **Lab4** to be released tomorrow
- **ProgTest2** results & feedback tentatively Monday Nov 11

Terminology: **Container** vs. **Containee**



Containee

Container

Container

shared by
1. some student s.cs[i]
2. some faculty f1.te[i]

"①" aliasing

Composition:

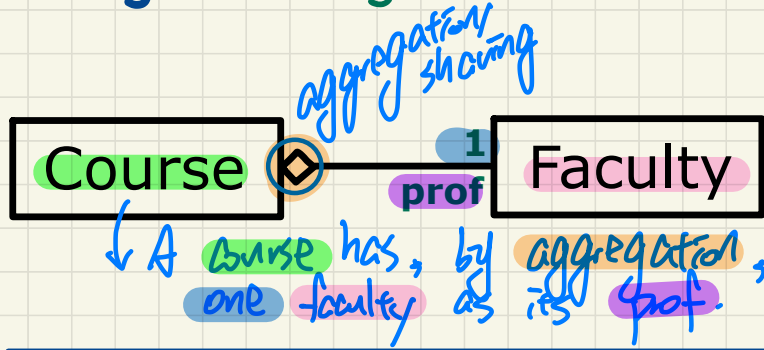
aggregation:

sharing/aliasing
min/home sharing

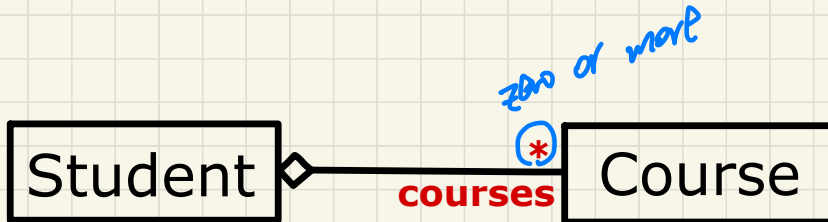
Aggregation: Design

Composition
no sharing.

Design 1: Single Containee



Design 2: Multiple Containees



Java Implementation

```
class Course {  
    Faculty prof;  
    ...  
}
```

```
class Faculty {  
    ...  
}
```

```
class Student {  
    Course[] courses;  
    ...  
}
```

```
class Course {  
    ...  
}
```

Aggregation (1)

Course	
title	
prof	

Faculty	
name	

```
class Course {
    String title;
    Faculty prof;
    Course(String title) {
        this.title = title;
    }
    void setProf(Faculty prof) {
        this.prof = prof;
    }
    Faculty getProf() {
        return this.prof;
    }
}
```

```
class Faculty {
    String name;
    Faculty(String name) {
        this.name = name;
    }
    void setName(String name) {
        this.name = name;
    }
    String getName() {
        return this.name;
    }
}
```

```
@Test
public void testAggregation1() {
    Course eecs2030 = new Course("Advanced OOP");
    Course eecs3311 = new Course("Software Design");
    Faculty prof = new Faculty("Jackie");
    eecs2030.setProf(prof);
    eecs3311.setProf(prof);
    assertTrue(eecs2030.getProf() == eecs3311.getProf());
    /* aliasing */
    prof.setName("Jeff");
    assertTrue(eecs2030.getProf() == eecs3311.getProf());
    assertTrue(eecs2030.getProf().getName().equals("Jeff"));

    Faculty prof2 = new Faculty("Jonathan");
    eecs3311.setProf(prof2);
    assertTrue(eecs2030.getProf() != eecs3311.getProf());
    assertTrue(eecs2030.getProf().getName().equals("Jeff"));
    assertTrue(eecs3311.getProf().getName().equals("Jonathan"));
}
```

Aggregation (2)

Student	
id	
cs	

Faculty	
name	
te	

Course	
title	
prof	

```
@Test
public void testAggregation2() {
    Faculty p = new Faculty("Jackie");
    Student s = new Student("Jim");
    Course eecs2030 = new Course("Advanced OOP");
    Course eecs3311 = new Course("Software Design");
    eecs2030.setProf(p);
    eecs3311.setProf(p);
    p.addTeaching(eecs2030);
    p.addTeaching(eecs3311);
    s.addCourse(eecs2030);
    s.addCourse(eecs3311);

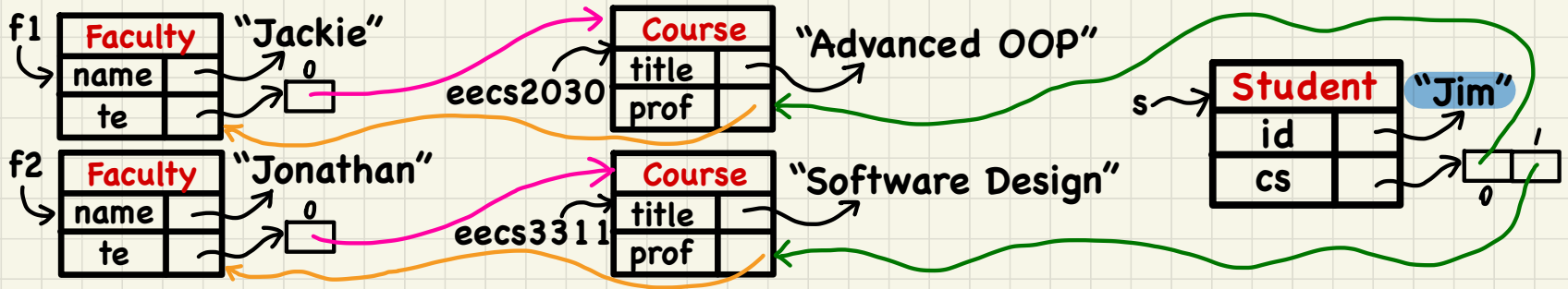
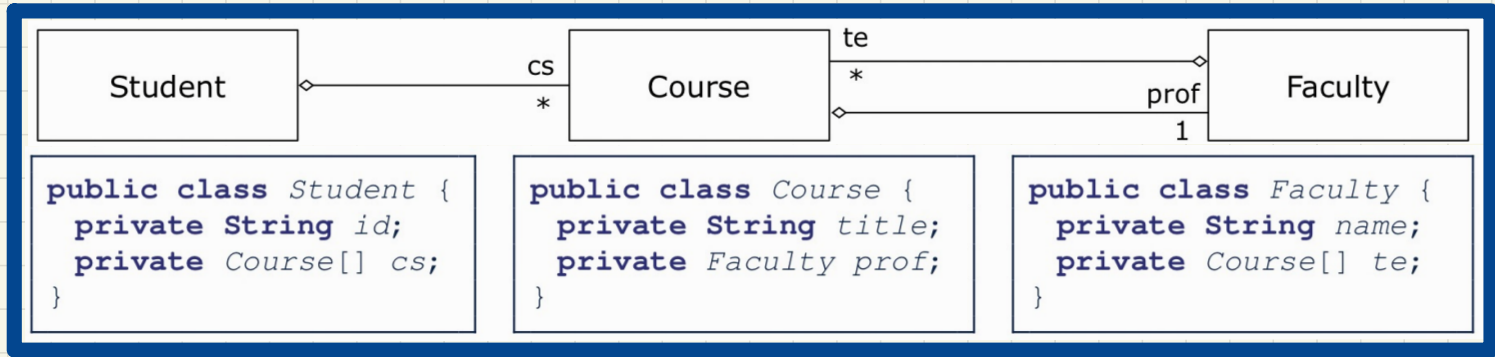
    assertTrue(eecs2030.getProf() == s.getCS()[0].getProf());
    assertTrue(s.getCS()[0].getProf()
        == s.getCS()[1].getProf());
    assertTrue(eecs3311 == s.getCS()[1]);
    assertTrue(s.getCS()[1] == p.getTE()[1]);
}
```

```
public class Student {
    private String id; Course[] cs; int noc; /* # of courses */
    public Student(String id) { ... }
    public void addCourse(Course c) { ... }
    public Course[] getCS() { ... }
}
```

```
public class Course { private String title; private Faculty prof; }
```

```
public class Faculty {
    private String name; Course[] te; int not; /* # of teaching */
    public Faculty(String name) { ... }
    public void addTeaching(Course c) { ... }
    public Course[] getTE() { ... }
}
```

Runtime Object Structure: Student, Course, Faculty



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).

```
class X {  
    ...  
    exp.privateAtt  
    ...  
}
```

```
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();  
        ① result = this.ob.bi; ✗  
        result = this.getB().bi;  
        result = this.ob.getBi();  
        result = this.getB().getBi();  
        ② result = this.ob.getA().ai; ✓  
        result = this.ob.getA().getAi();  
        result = this.ob.aa.ai;  
        result = this.ob.aa.getAi();  
        return result;  
    }  
}
```

ok
this.ob.bi
private

ⓑ does not compile 'cause the context class is diff: A

ok
this.ob.getA().ai
B public A

ok to access 'cause context of this / up is same: A

```
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 (1)



```
public class Student {
    private String id;
    private Course[] cs;
}
```

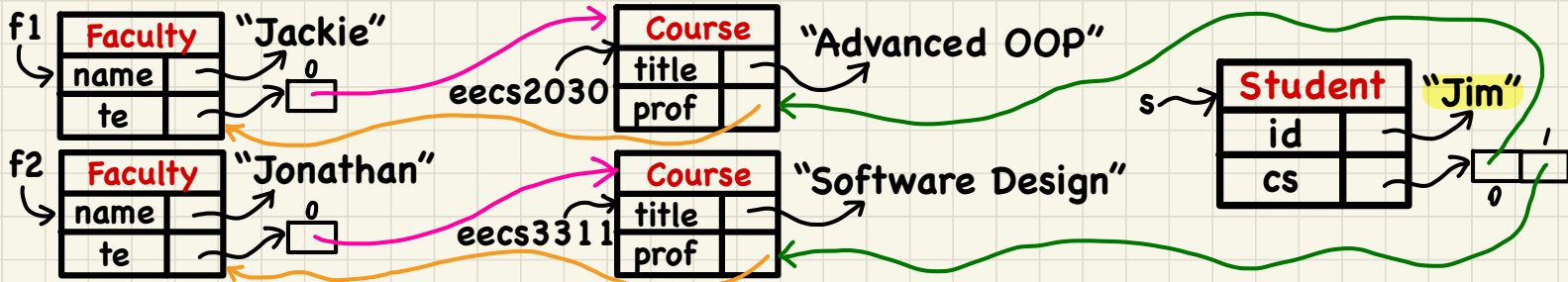
```
public class Course {
    private String title;
    private Faculty prof;
}
```

```
public class Faculty {
    private String name;
    private Course[] te;
}
```

```
/* Get the student's id.
 */
String getID() {
    this.id
}
```

```
/* Title of ith course
 */
String getTitle(int i) {
    this.cs[i].getTitle()
}
```

```
/* Name of
 * ith course's instructor
 */
String getName(int i) {
    this.cs[i].getProf().getName()
}
```



Dot Notation for Navigating Classes (2) *this.prof.^{private}te[i].title*



```

public class Student {
    private String id;
    private Course[] cs;
}
  
```

```

public class Course {
    private String title;
    private Faculty prof;
}
  
```

```

public class Faculty {
    private String name;
    private Course[] te;
}
  
```

```

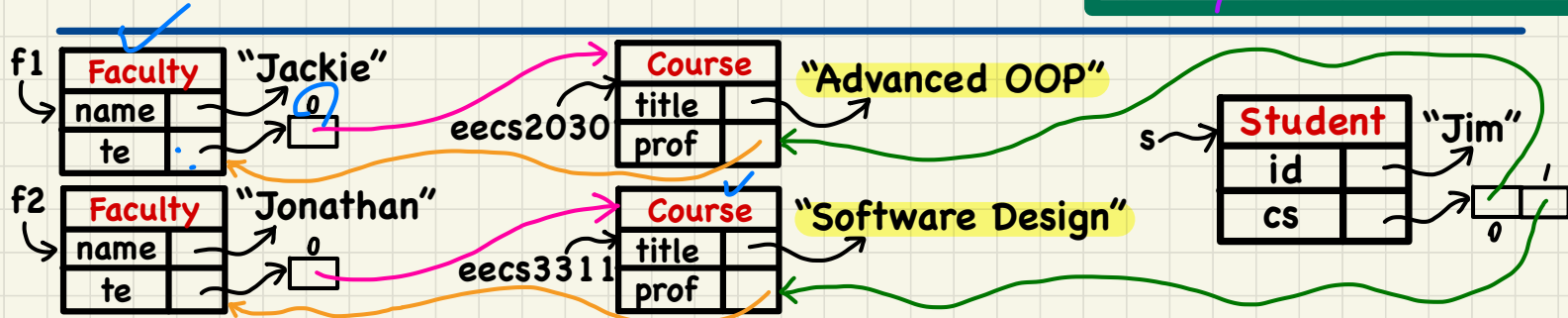
/* Get course's title.
*/
String getTitle() {
    this.title
}
  
```

```

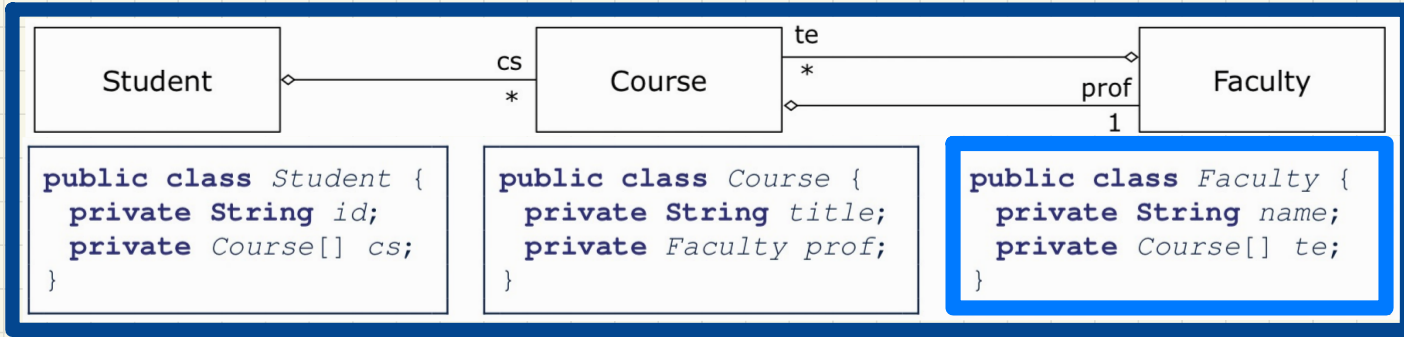
/* Name of instructor
*/
String getName() {
    this.prof.getName()
}
  
```

```

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



Dot Notation for Navigating Classes (3) Exercise

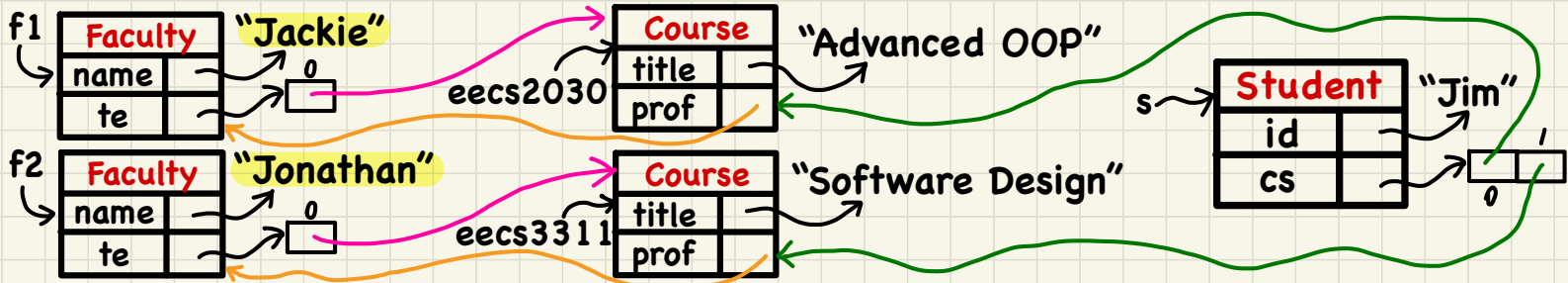


```
/* Name of instructor
*/
String getName() {

}
```

```
/* Title of instructor's
* ith teaching course
*/
String getTitle(int i) {

}
```



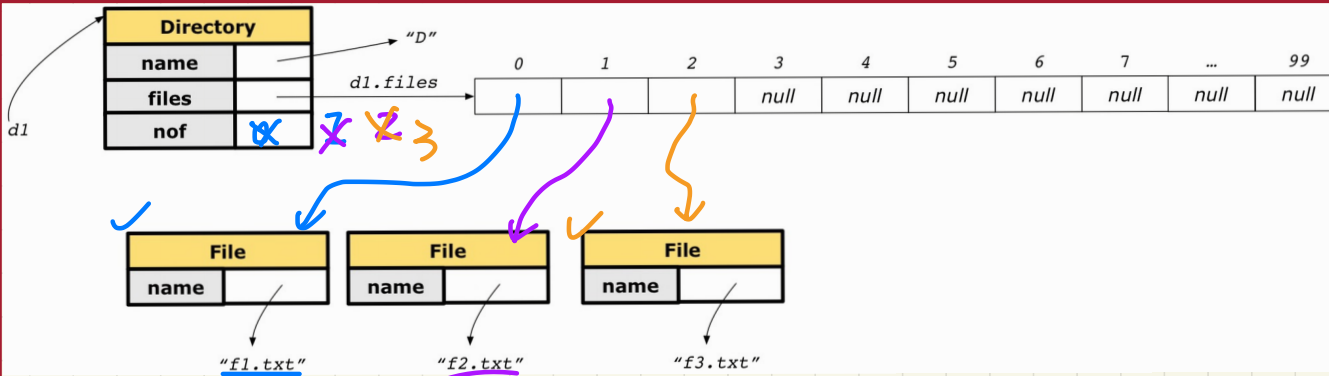
Composition: No Sharing

```
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     )  
}
```

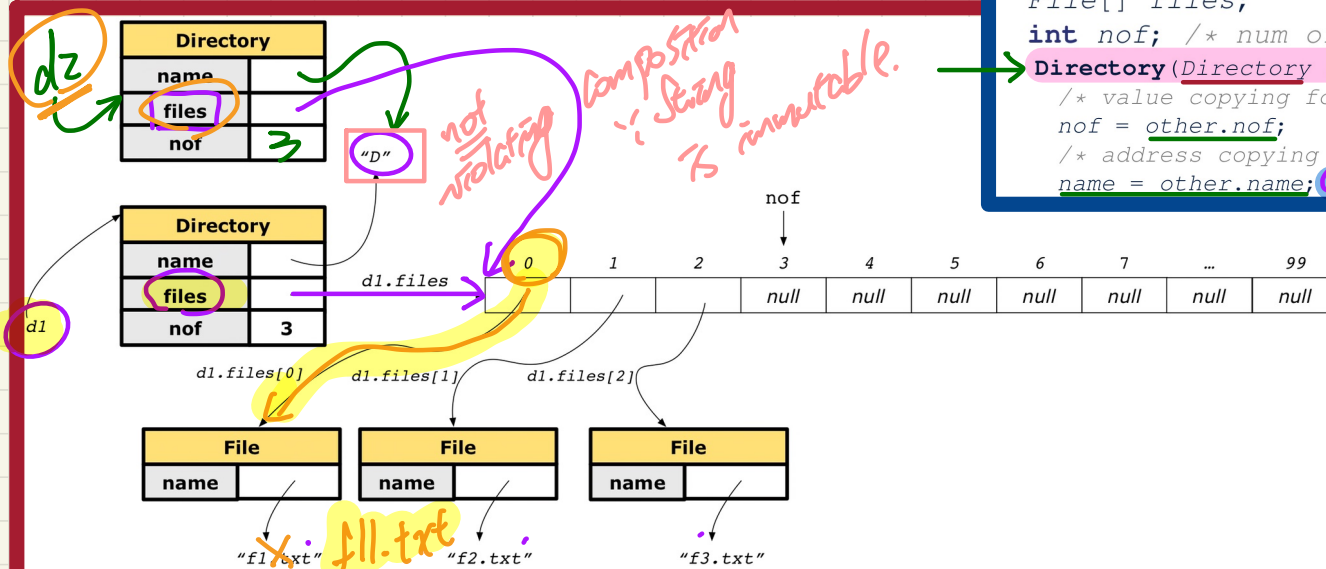


Composition: Copy Constructor (Shallow Copy)

```
@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);
    d2.files[0].changeName("f11.txt");
    assertFalse(d1.files[0].name.equals("f1.txt"));
}
```

also modifies a file shared by d1.

```
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;
    }
}
```



shallow copy: just copy the ref. of the array.

public class X {

copy constructor.

public X (X other) {

}