Lecture 15
Wednesday October 30

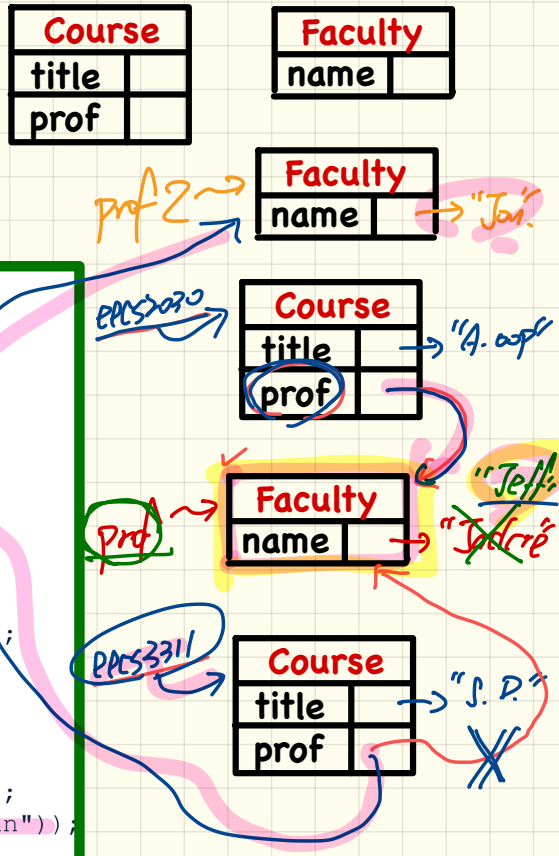
Aggregation (1)

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

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

```java
@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)

```
class Student {
  String id; ArrayList<Course> cs; /* courses */
  Student(String id) { this.id = id; cs = new ArrayList<>(); }
  void addCourse(Course c) { cs.add(c); }
  ArrayList<Course> getCS() { return cs; }
}
```

elements in list are of type Course

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

```
class Faculty {
  String name; ArrayList<Course> te; /* teaching */
  Faculty(String name) { this.name = name; te = new ArrayList<>(); }
  void addTeaching(Course c) { te.add(c); }
  ArrayList<Course> getTE() { return te; }
}
```

```
@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().get(0).getProf());
  assertTrue(s.getCS().get(0).getProf()
             == s.getCS().get(1).getProf());
  assertTrue(eecs3311 == s.getCS().get(1));
  assertTrue(s.getCS().get(1) == p.getTE().get(1));
}
```

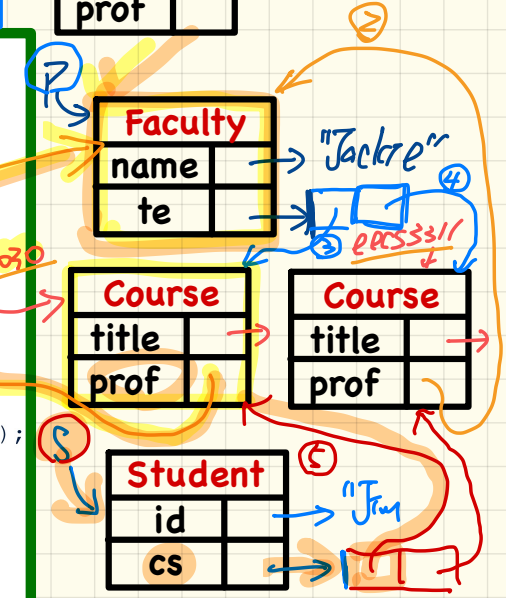
A.L.<Course>

Course

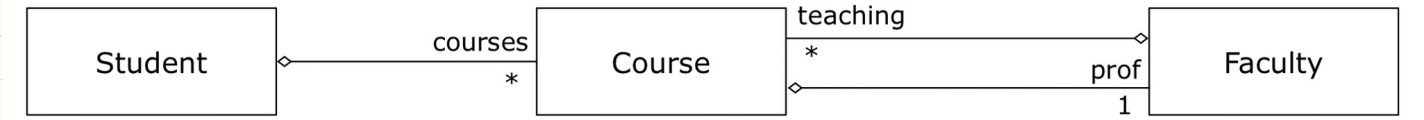
Student	
id	
cs	

Faculty	
name	
te	

Course	
title	
prof	



Dot Notation for Navigating Aggregations (1)




```
class Student {
  String id;
  ArrayList<Course> cs;
}
```

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

```
class Faculty {
  String name;
  ArrayList<Course> te;
}
```

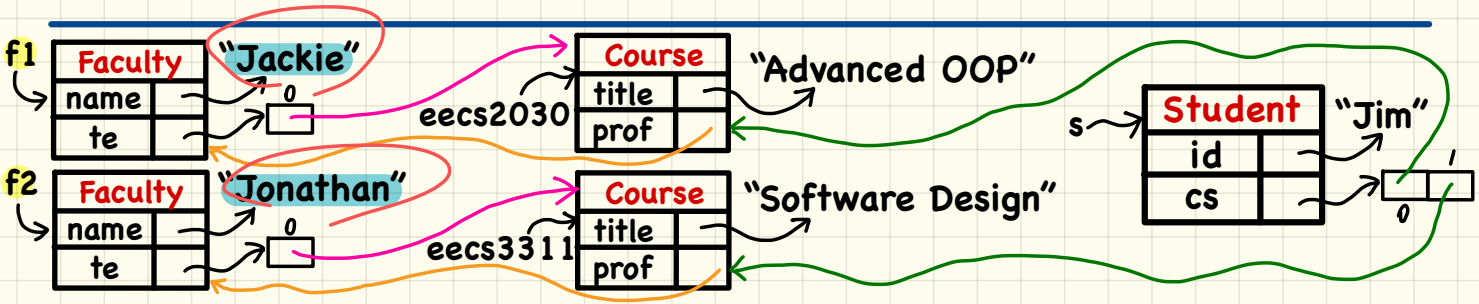
Examples

f1.getName()

f2.getName()

```
/* Name of this faculty
 */
String getName()
```

return name;



Dot Notation for Navigating Aggregations (2)


```
class Student {
  String id;
  ArrayList<Course> cs;
}
```

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

```
class Faculty {
  String name;
  ArrayList<Course> te;
}
```

```
/* Instructor's name
 * for this course
 */
String getName()
```

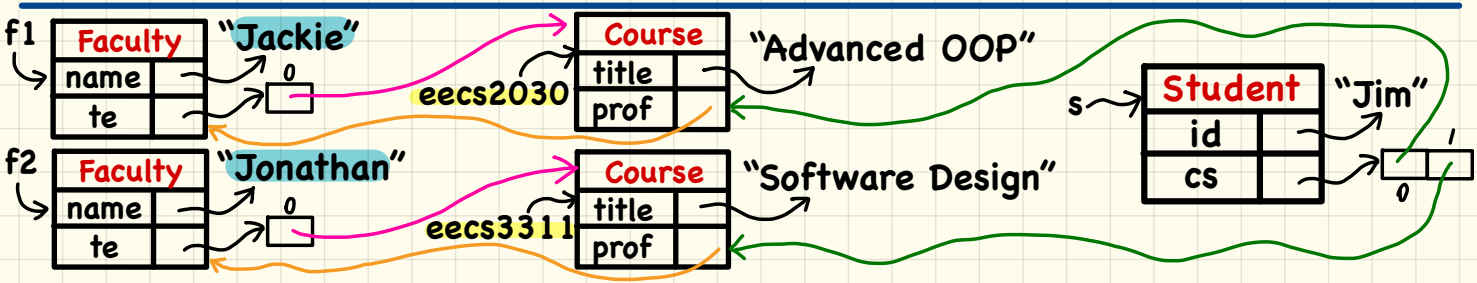
return this.faculty. . .
. prof . name

return this.prof.name

Examples

eecs2030.getName()

eecs3311.getName()



Dot Notation for Navigating Aggregations (3)


```
class Student {
  String id;
  ArrayList<Course> cs;
}
```

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

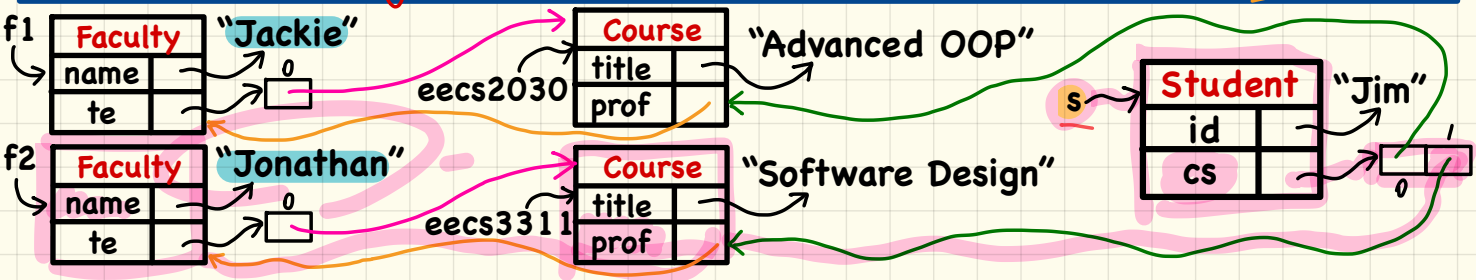
```
class Faculty {
  String name;
  ArrayList<Course> te;
}
```

```
/* Instructor's name for
 * course stored at index i
 */
String getName(int i)
return this.cs.get(i).prof.name
```

Examples

s.getName(0)

s.getName(1)



Dot Notation for Navigating Aggregations: Exercise


```
class Student {
  String id;
  ArrayList<Course> cs;
}
```

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

```
class Faculty {
  String name;
  ArrayList<Course> te;
}
```

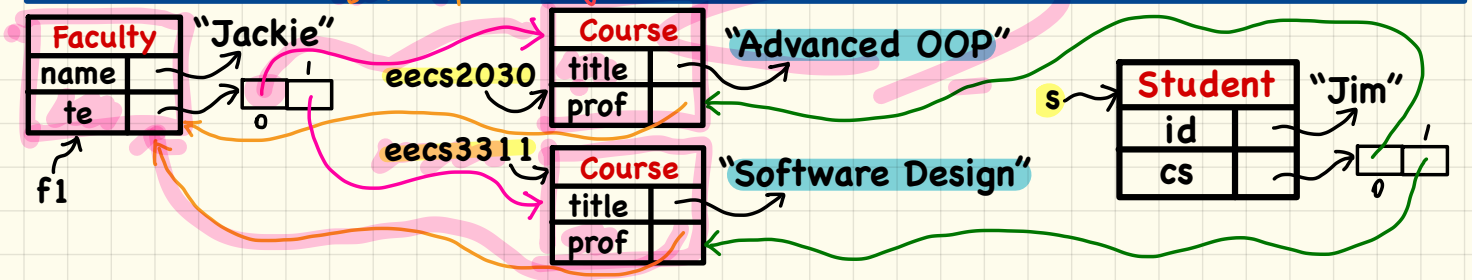
```
/* Title of the ith teaching course
 * of the instructor for this course
 */
String getTitle(int i)
```

return this.prof.te.get(i).title

Examples

eecs2030.getTitle(1)

eecs3311.getTitle(0)

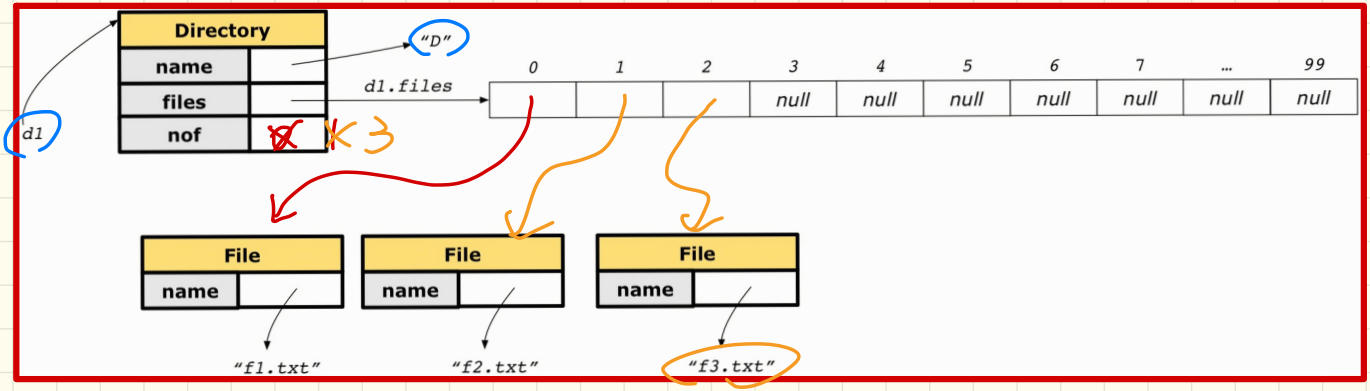


Composition: No Sharing

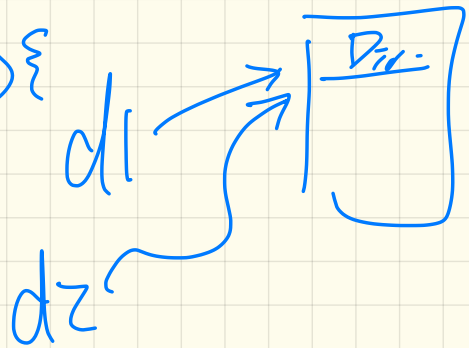
```
class Directory {
  String name;
  File[] files;
  int nof; /* num of files */
  Directory(String name) {
    this.name = name;
    files = new File[100];
  }
  void addFile(String fileName) {
    files[nof] = new File(fileName);
    nof ++;
  }
}
```

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

```
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 Directory {
    Directory(Directory other) {
        this = other;
    }
}
```




```
Directory d1 = new Directory("D1");
Directory d2 = new Directory(d1);
```

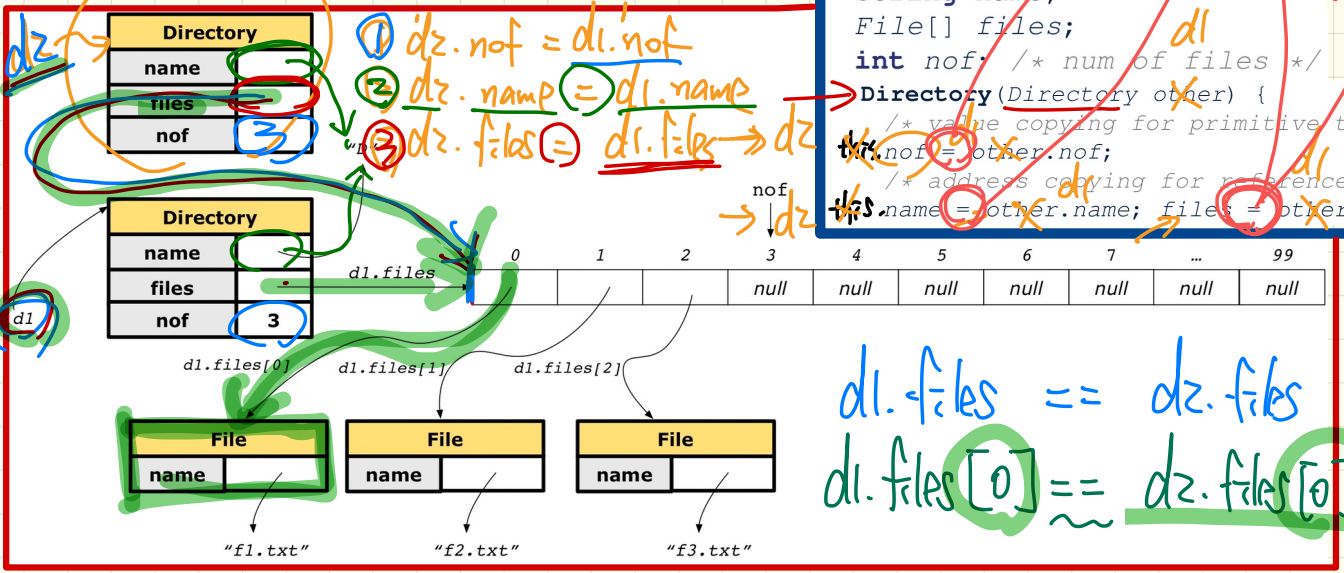
Composition: Copy Constructor (Shallow Copy)

```
@Test
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")); }
```

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

shallow copy

aliasing



dz.nof = d1.nof

dz.name = d1.name

dz.files = d1.files

d1.files == dz.files

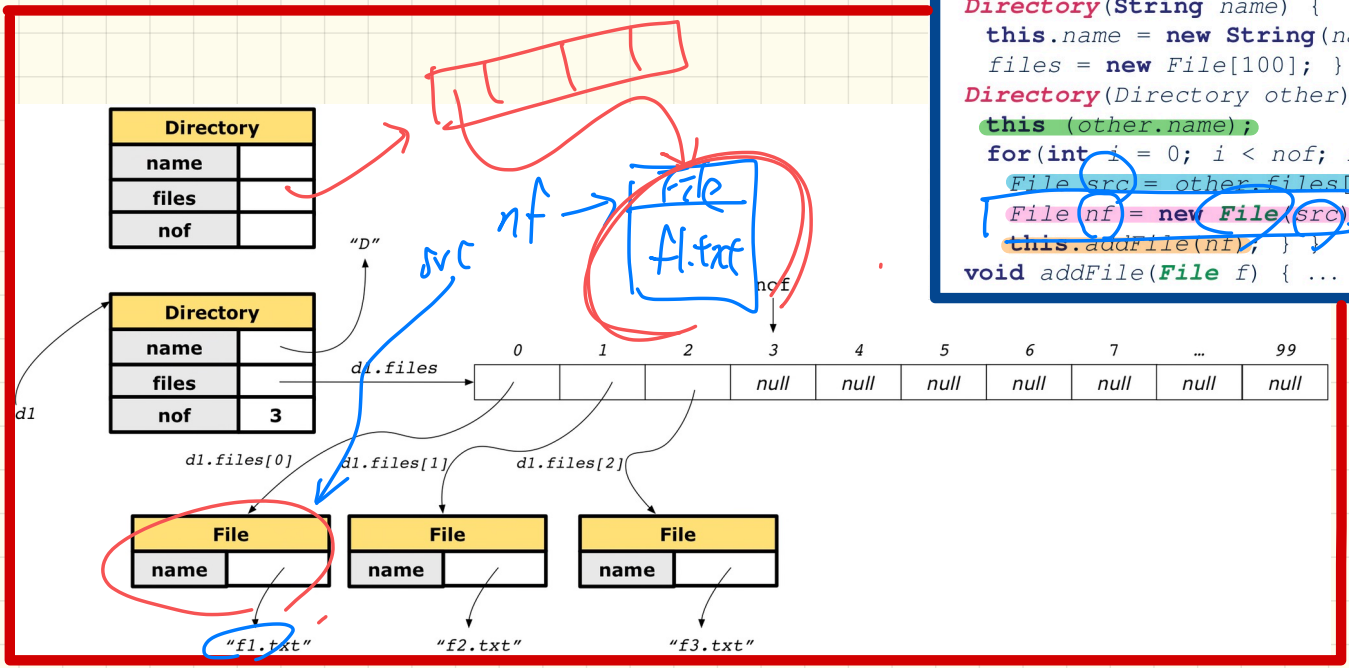
d1.files[0] == dz.files[0]

Composition: Copy Constructor (Deep Copy)

```java
@Test
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")); }
```

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

```java
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) { ... } }
```



Copy Constructor (Composition?)

File src = d1.files[0];

```
@Test
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")); }
```

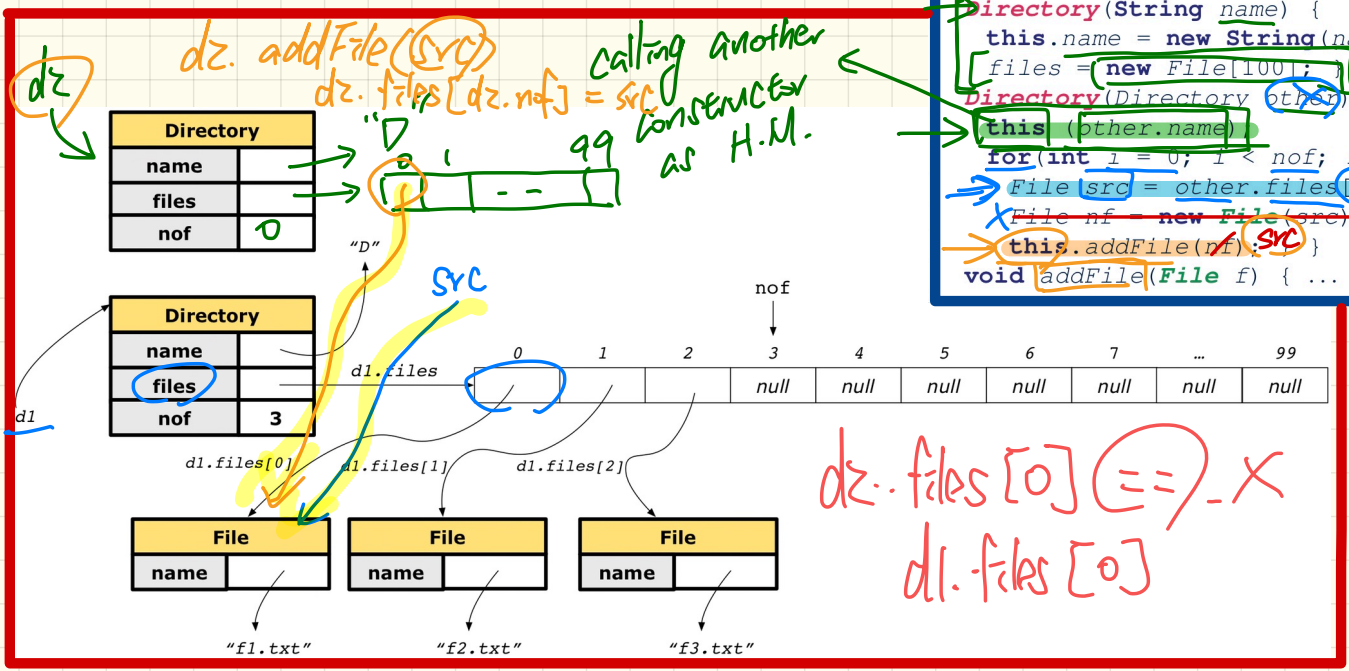
```
class File {
  File(File other) {
    this.name =
      new String(other.name);
  }
}
```

```
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) { ... } }
```

d2.addFile(src)

d2.files[d2.nf] = src

calling another constructor as H.M.



d2.files[0] == X d1.files[0]