

Lecture 16 - Nov. 5

Composition, Inheritance

Composition: Deep Copy

Design Attempts without Inheritance

Inheritance: Use of extends, super

Announcements/Reminders

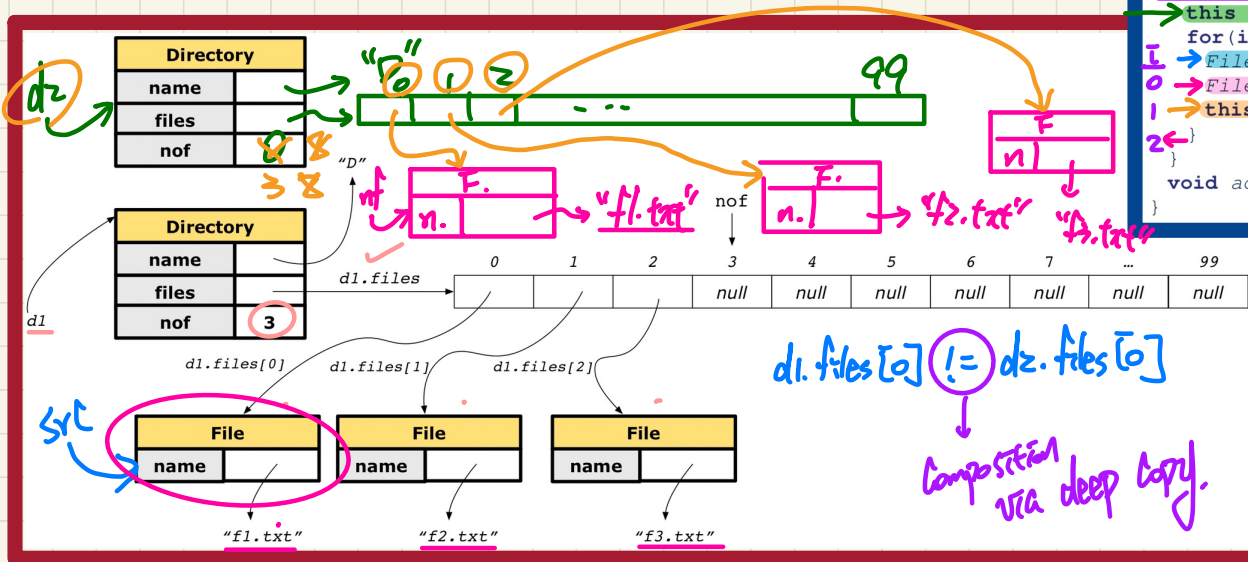
- **Lab4** released (**ProgTest3** on November 20)
- Guide & Questions for **WrittenTest2** to be released tmw
- In-Lab Demo on **Inheritance** tomorrow
- **ProgTest2** results & feedback tentatively Monday Nov 11

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);
    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 < other.nof; i++) {
            File src = other.files[i];
            File nf = new File(src);
            this.addFile(nf);
        }
        void addFile(File f) { ... }
    }
}
```



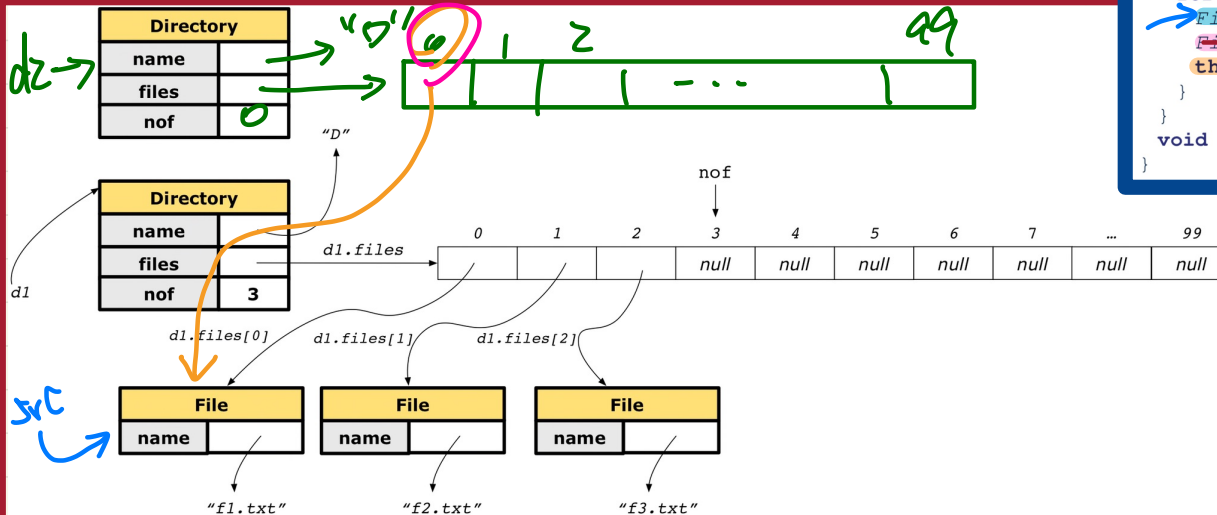
Exercise: Copy Constructor (Composition?)

```
@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]);
}
```

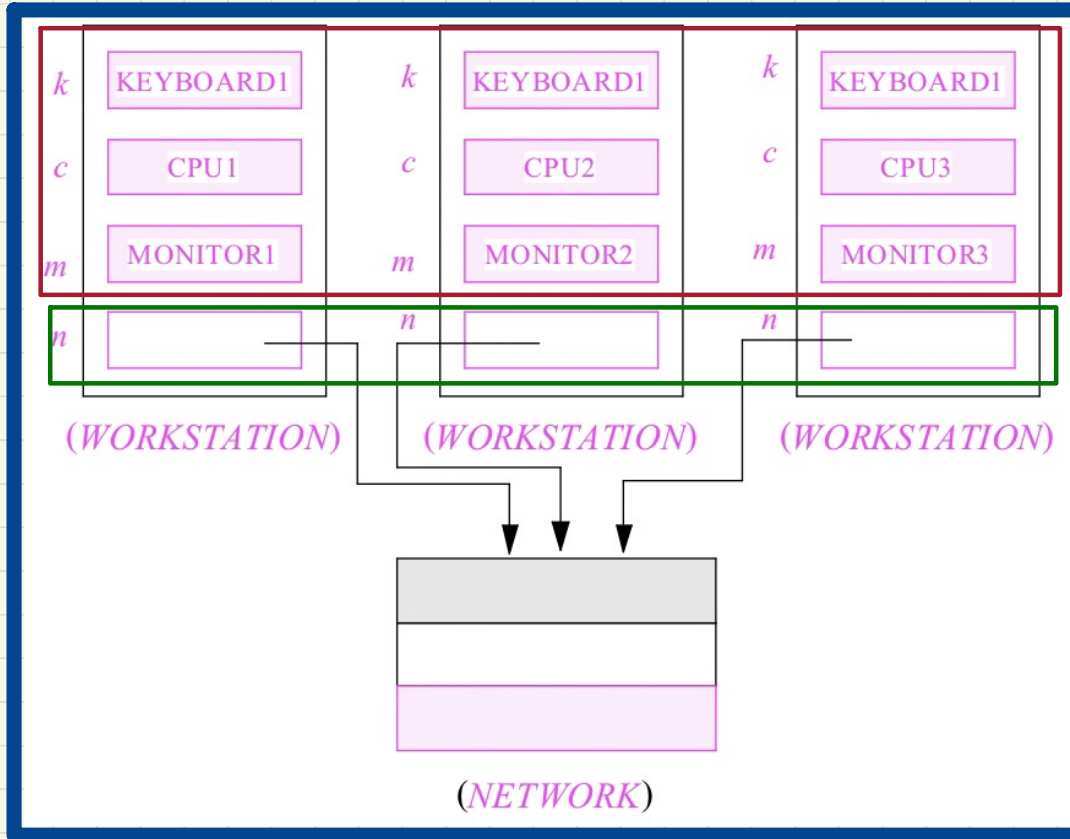
```
class File {
    File(File other) {
        this.name =
            new String(other.name);
    }
}
```

```
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 < other.nof; i++) {
            File src = other.files[i];
            File nf = new File(src);
            this.addFile((nf) src);
        }
    }
    void addFile(File f) { ... }
}
```

$d1.files \neq d2.files$ $d2.files[0] == \textcircled{D}$



Modelling: Aggregation vs. Composition

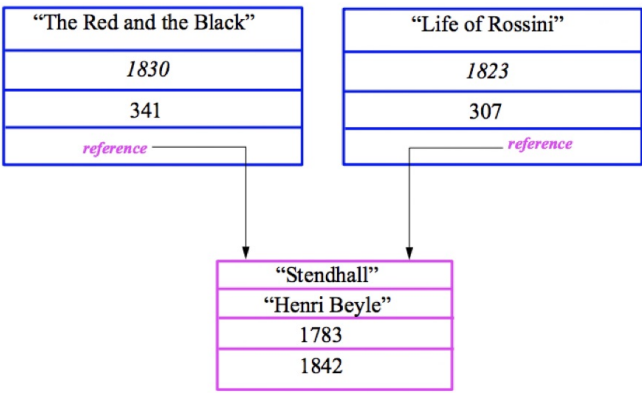


Composition

aggregation

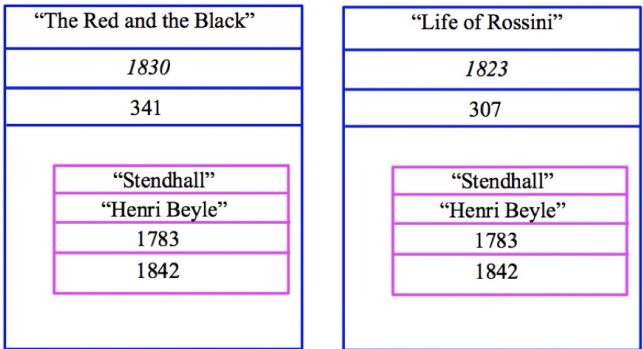
Implementation: Aggregation or Composition

author as an *aggregation*



Hyperlinked author page

author as a *composition*



Physical printed copies

Inheritance: Motivating Problem

Nouns -> classes, attributes, accessors

Verbs -> mutators

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.

First Design Attempt

how to simulate OO in a non-OO language?

```
public class Student {  
    private Course[] courses;  
    private int noc;  
  
    private int kind;  
    private double premiumRate;  
    private double discountRate;  
  
    public Student (int kind){  
        this.kind = kind;  
    }  
    ...  
}
```

Handwritten notes:
- **RS** (for `kind`)
- **NRS** (for `premiumRate` and `discountRate`)
- `kind` is annotated with `==1` and `==2`
- `premiumRate` and `discountRate` are underlined in pink

`Student rs = new Student(1);`
`Student nrs = new Student(2);`

```
public double getTuition(){  
    double tuition = 0;  
    for(int i = 0; i < this.noc; i++){  
        tuition += this.courses[i].fee;  
    }  
    if (this.kind == 1) {  
        return tuition * this.premiumRate;  
    }  
    else if (this.kind == 2) {  
        return tuition * this.discountRate;  
    }  
}
```

Handwritten notes:
- `tuition` is boxed in blue
- `base amt.` is written in blue next to the `tuition` box
- `if (this.kind == 1)` and `else if (this.kind == 2)` are highlighted in orange

```
public void register(Course c){  
    int MAX = -1;  
    if (this.kind == 1) { MAX = 6; }  
    else if (this.kind == 2) { MAX = 4; }  
    if (this.noc == MAX) { /* Error */ }  
    else {  
        this.courses[this.noc] = c;  
        this.noc ++;  
    }  
}
```

Handwritten notes:
- `if (this.kind == 1)` and `else if (this.kind == 2)` are highlighted in orange
- `/* Error */` is in grey

First Design Attempt

```
public class Student {  
    private Course[] courses;  
    private int noc;  
  
    private int kind;  
    private double premiumRate;  
    private double discountRate;  
  
    public Student (int kind){  
        this.kind = kind;  
    }  
    ...  
}
```

only apply to RS

only apply to NRS

```
public double getTuition(){  
    double tuition = 0;  
    for(int i = 0; i < this.noc; i++){  
        tuition += this.courses[i].fee;  
    }  
    if (this.kind == 1) {  
        return tuition * this.premiumRate;  
    }  
    else if (this.kind == 2) {  
        return tuition * this.discountRate;  
    }  
}
```

```
public void register(Course c){  
    int MAX = -1;  
    if (this.kind == 1) { MAX = 6; }  
    else if (this.kind == 2) { MAX = 4; }  
    if (this.noc == MAX) { /* Error */ }  
    else {  
        this.courses[this.noc] = c;  
        this.noc ++;  
    }  
}
```

Good design?

Judge by

Cohesion

attributes and methods in a class follow theme

First Design Attempt

```
public class Student {  
    private Course[] courses;  
    private int noc;  
  
    private int kind;  
    private double premiumRate;  
    private double discountRate;  
  
    public Student (int kind){  
        this.kind = kind;  
    }  
    ...  
}
```

```
public double getTuition(){  
    double tuition = 0;  
    for(int i = 0; i < this.noc; i++){  
        tuition += this.courses[i].fee;  
    }  
    if (this.kind == 1) {  
        return tuition * this.premiumRate;  
    }  
    else if (this.kind == 2) {  
        return tuition * this.discountRate;  
    }  
    else if (this.kind == 3) { ... }
```

```
public void register(Course c){  
    int MAX = -1;  
    if (this.kind == 1) { MAX = 6; }  
    else if (this.kind == 2) { MAX = 4; }  
    if (this.noc == MAX) { /* Error */ }  
    else {  
        this.courses[this.noc] = c;  
        this.noc ++;  
    }  
}
```

Good design?

Judge by Single Choice Principle

- **Repeated** if-conditions
- A new kind is **introduced?**
- An existing kind is **obsolete?**

informational
"3"

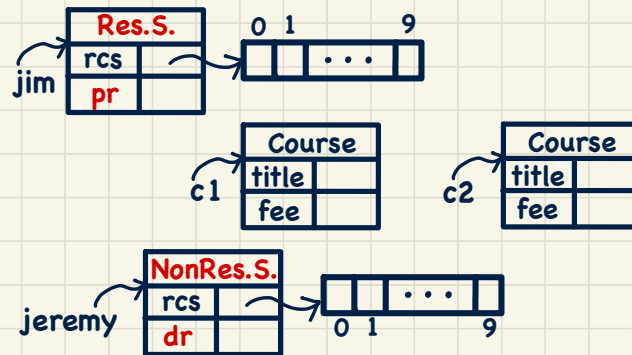
else if
(this.kind == 3)
{ ... }

Testing Student Classes (without inheritance)

```
public class ResidentStudent {
    private String name;
    private Course[] courses; private int noc;
    private double premiumRate; /* assume a m
    public ResidentStudent (String name) {
        this.name = name;
        this.courses = new Course[10];
    }
    public void register(Course c) {
        this.courses[this.noc] = c;
        this.noc ++;
    }
    public double getTuition() {
        double tuition = 0;
        for(int i = 0; i < this.noc; i ++ ) {
            tuition += this.courses[i].fee;
        }
        return tuition * this.premiumRate;
    }
}
```

```
public class NonResidentStudent {
    private String name;
    private Course[] courses; private int noc;
    private double discountRate; /* assume a
    public NonResidentStudent (String name) {
        this.name = name;
        this.courses = new Course[10];
    }
    public void register(Course c) {
        this.courses[this.noc] = c;
        this.noc ++;
    }
    public double getTuition() {
        double tuition = 0;
        for(int i = 0; i < this.noc; i ++ ) {
            tuition += this.courses[i].fee;
        }
        return tuition * this.discountRate;
    }
}
```

```
public class StudentTester {
    public static void main(String[] args) {
        Course c1 = new Course("EECS2030", 500.00); /* title and fee */
        Course c2 = new Course("EECS3311", 500.00); /* title and fee */
        ResidentStudent jim = new ResidentStudent("J. Davis");
        jim.setPremiumRate(1.25);
        jim.register(c1); jim.register(c2);
        NonResidentStudent jeremy = new NonResidentStudent("J. Gibbons");
        jeremy.setDiscountRate(0.75);
        jeremy.register(c1); jeremy.register(c2);
        System.out.println("Jim pays " + jim.getTuition());
        System.out.println("Jeremy pays " + jeremy.getTuition());
    }
}
```



Student Classes (with inheritance)

* invoke method defined at the super class

```
class Student { parent/super class
    String name;
    Course[] registeredCourses;
    int numberOfCourses;
    Student (String name) {
        this.name = name;
        registeredCourses = new Course[10];
    }
    void register(Course c) {
        registeredCourses[numberOfCourses] = c;
        numberOfCourses ++;
    }
    double getTuition() {
        double tuition = 0;
        for(int i = 0; i < numberOfCourses; i ++) {
            tuition += registeredCourses[i].fee;
        }
        return tuition; /* base amount only */
    }
}
```

child/sub class
RS inherits all attributes and methods from S

RS and NRS have the same parent/super class.

```
class ResidentStudent extends Student {
    double premiumRate; /* there's a mutator method */
    ResidentStudent (String name) { super(name); }
    /* register method is inherited */
    double getTuition() {
        double base = super.getTuition();
        return base * premiumRate;
    }
}
```

Invoking constructor at the super class

```
class NonResidentStudent extends Student {
    double discountRate; /* there's a mutator method */
    NonResidentStudent (String name) { super(name); }
    /* register method is inherited */
    double getTuition() {
        double base = super.getTuition();
        return base * discountRate;
    }
}
```