Lecture 16 - Nov 4

Inheritance

Design Principles: Cohesion, SCP
SMS: Design Attempt 1
SMS: Design Attempt 2
Implementing Inheritance in Java

Announcements/Reminders

- Today's class: notes template posted
- WrittenTest1 marks and feedback released
- ProgTest2 this Wednesday (November 5)
  + Guide released
  + PracticeTest2 released
  + Lab3 solution released
- Tracing Exercises: assertEquals and Person, PersonCollector

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.

Design Principles

1. Cohesion

↳ Attributes and methods in a single class should serve the same purpose.

be under a unifying theme

Unix principle
↳ each command should do one thing, and do it well

is cal pwd

2. Single Choice Principle

↳ A change to the software should impact a minimum number of places

not necessarily 1, but no more than necessary.

methods, classes.

violation: duplicates.

First Design Attempt

how to simulate OO in a non-OO language (e.g. C).

```
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;
   }
   ...
}
```

32

==1 RS
==2 NRS

RS

NRS

1 2

rs.getTuition()
nrs.getTuition()

```
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;
   }
}
```

loop amount

```
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 ++;
   }
}
```

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

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;
   }
   ...
}
```

mutually exclusive

↳ should be put in separate classes.

```
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

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;
   }
}
```

dup1

dup2

```
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 obselete?

Testing Student Classes (without inheritance)

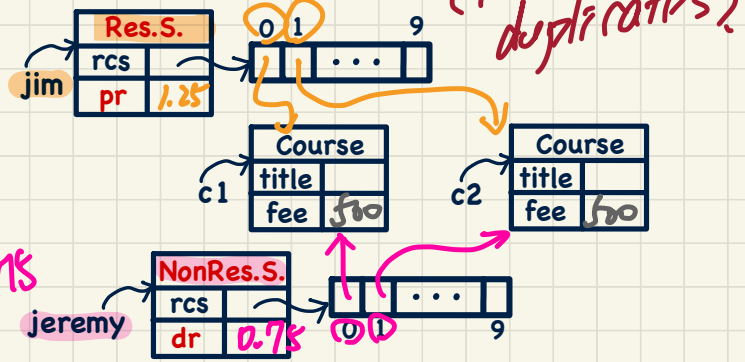
```java
public class ResidentStudent {
  private String name;
  private Course[] courses; private int noc;
  private double premiumRate; /* assume a m
  public ResidentStudent(String name) {
    this.name = name;
    this.courses = new Course[10];
  }
  public void register(Course c) {
    this.courses[this.noc] = 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;
  }
}
```

```java
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;
  }
}
```

cohesion (pr and dr put in different classes)

SCP violated (lots of duplications)

```java
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()); 1000 * 1.25
    System.out.println("Jeremy pays " + jeremy.getTuition()); 1000 * 0.75
  }
}
```



Student Classes (without inheritance): Maintenance (1)

```
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;
  }
}
```

every duplicate has to be changed!

Maintenance e.g., a new registration constraint:

```
if(numberOfCourses >= MAX_ALLOWANCE) {
   throw new TooManyCoursesException("Too Many Courses");
}
else { ... }
```

how many methods should be impacted?

Student Classes (without inheritance): Maintenance (2)

```
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;
  }
}
```

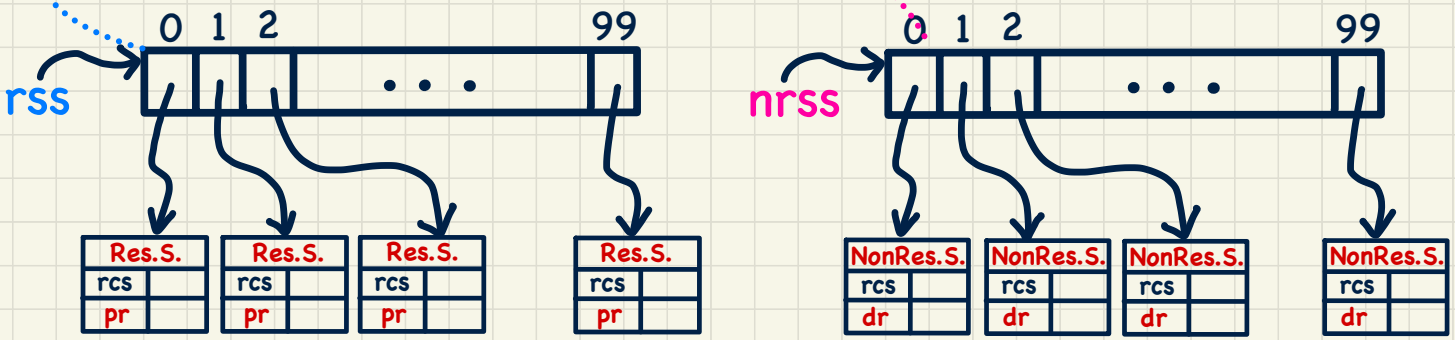
Maintenance e.g., a new tuition formula:

```
/* ... can be premiumRate or discountRate */
...
return tuition * inflationRate * ...;
```

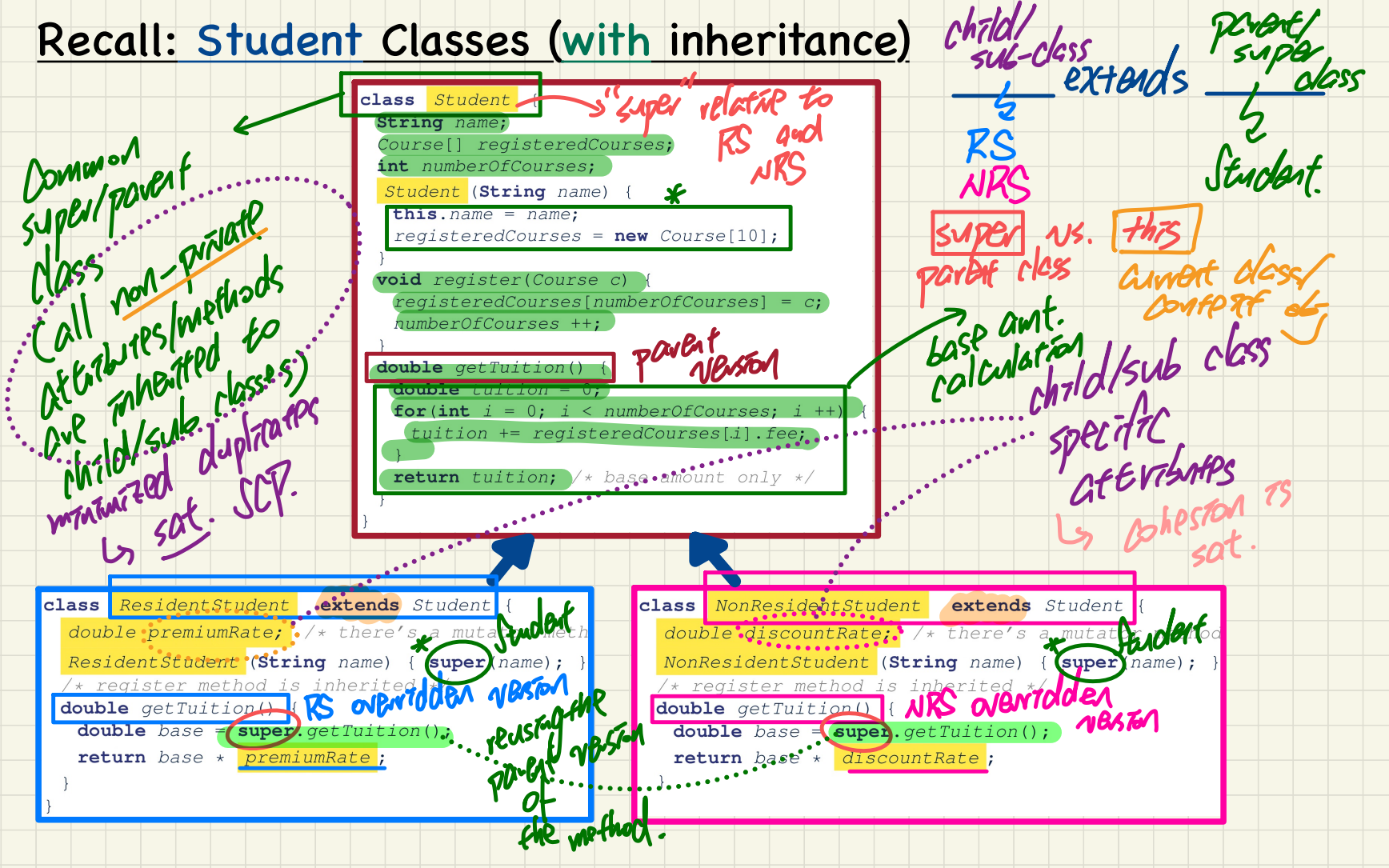
A Collection of Students (without inheritance)

```
public class StudentManagementSystem {
  private ResidentStudent[] rss;
  private NonResidentStudent[] nrss;
  private int nors; /* number of resident students */
  private int nonrs; /* number of non-resident students */
  public void addRS(ResidentStudent rs){ rss[nors]=rs; nors++; }
  public void addNRS(NonResidentStudent nrs){ nrss[nonrs]=nrs;nonrs++; }
  public void registerAll(Course c) {
    for(int i = 0; i < nors; i ++) { rss[i].register(c); }
    for(int i = 0; i < nonrs; i ++) { nrss[i].register(c); }
  }
}
```

arrays = # kinds of students



Recall: Student Classes (with inheritance)




```
class Student {
  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 */
  }
}
```

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

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

When programming with inheritance (OOP):