Lecture 16 - July 8

Inheritance

SMS: 2nd Design Attempt
SMS: Use of Inheritance (extends, super)
Static Types, Expectations, Polymorphism

Announcements/Reminders

- Today's class: notes template posted
- On-demand extra TA help hours
- ProgTest1 result released
- ProgTest2 (July 11)
  + Guide released
  + PracticeTest released
  + Review Session: Wednesday (July 9)
- Lab4 released
- Priorities:
  + Lab4

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;
   }
}
```

```
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;
  }
}
```

separate, unrelated classes

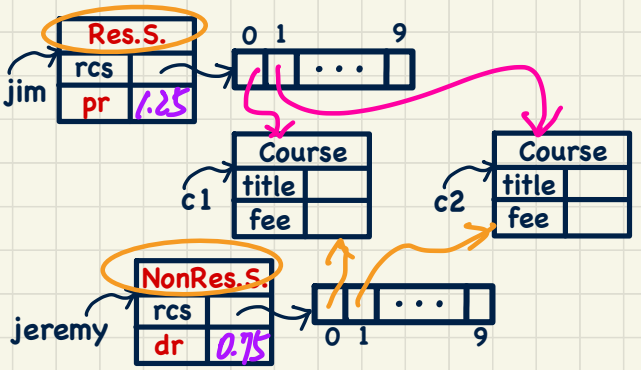
cohesion ✓

single-choice X

```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());
    System.out.println("Jeremy pays " + jeremy.getTuition());
  }
}
```

D.T: RS

D.T: NRS



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;
  }
}
```

Maintenance e.g., a new registration constraint:

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

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:

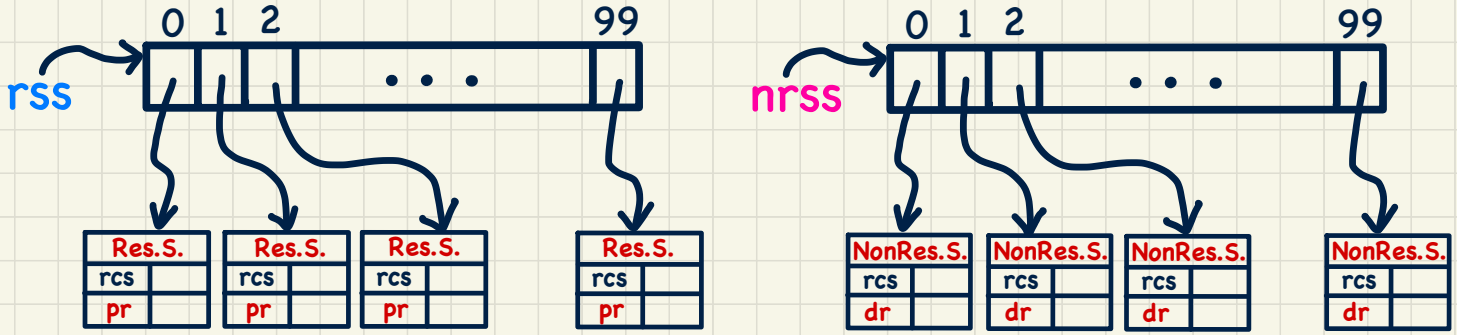
```
/* ... can be premiumRate or discountRate */
...
return tuition * inflationRate * ...;
```

A Collection of Students (without inheritance)

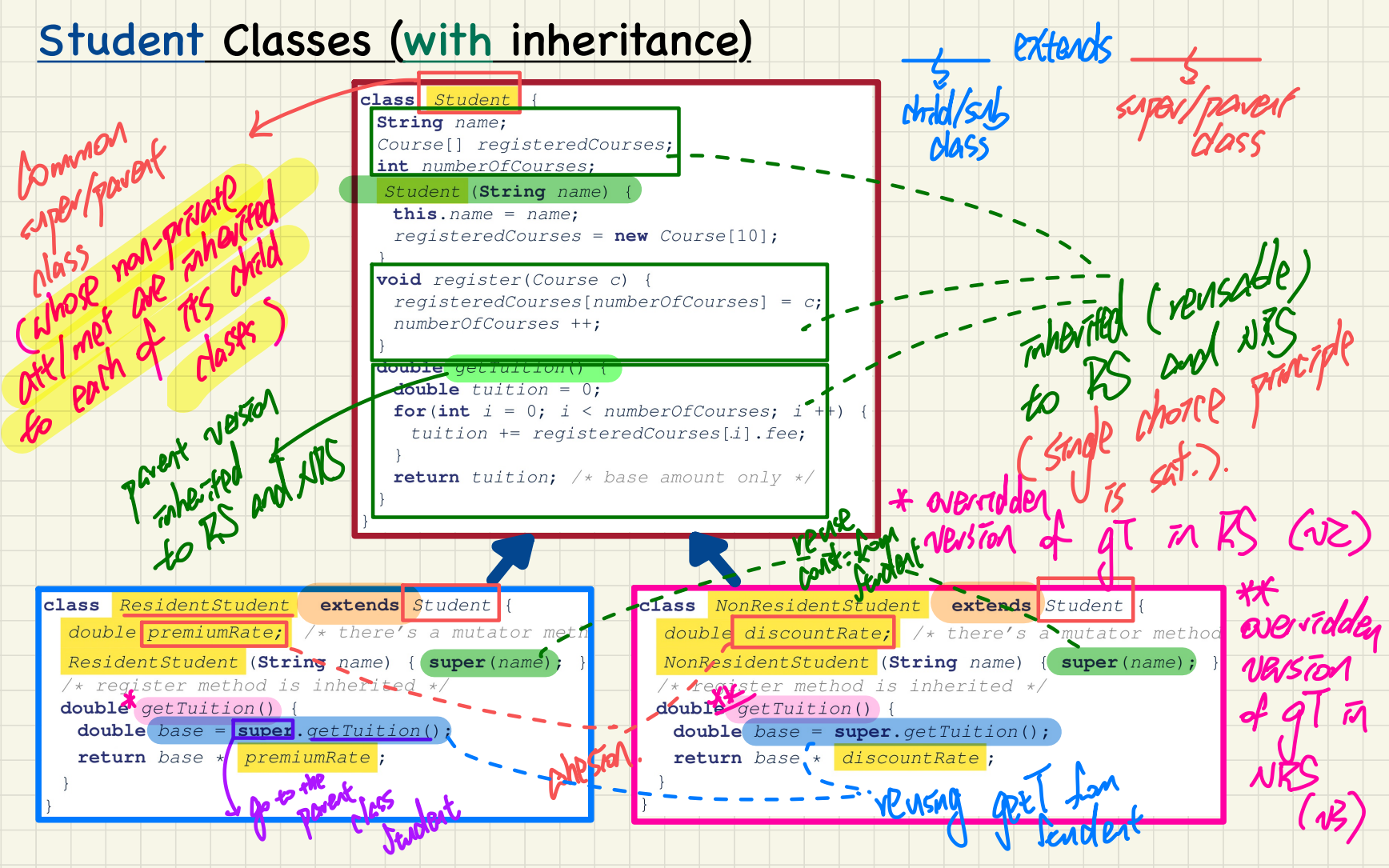
kinds
↳ # arrays
↳ # loops

```
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); }
  }
}
```

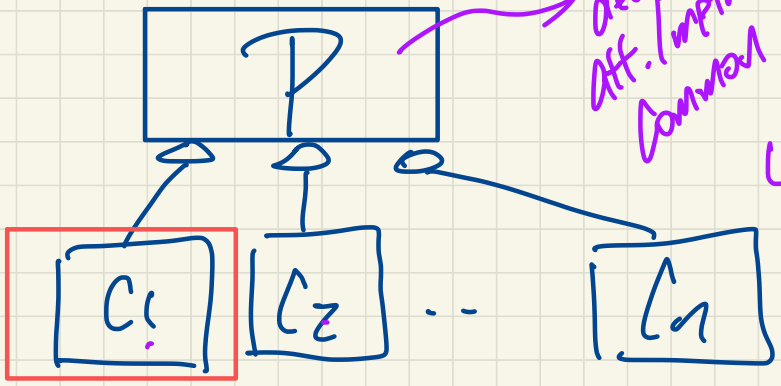
unrelated



Student Classes (with inheritance)



When programming with inheritance:



(1) declare aff./methods common to all subclasses (S.C.P. ✓)

(2) declare additional attrs/methods applicable to this subclass only (cohesion ✓)

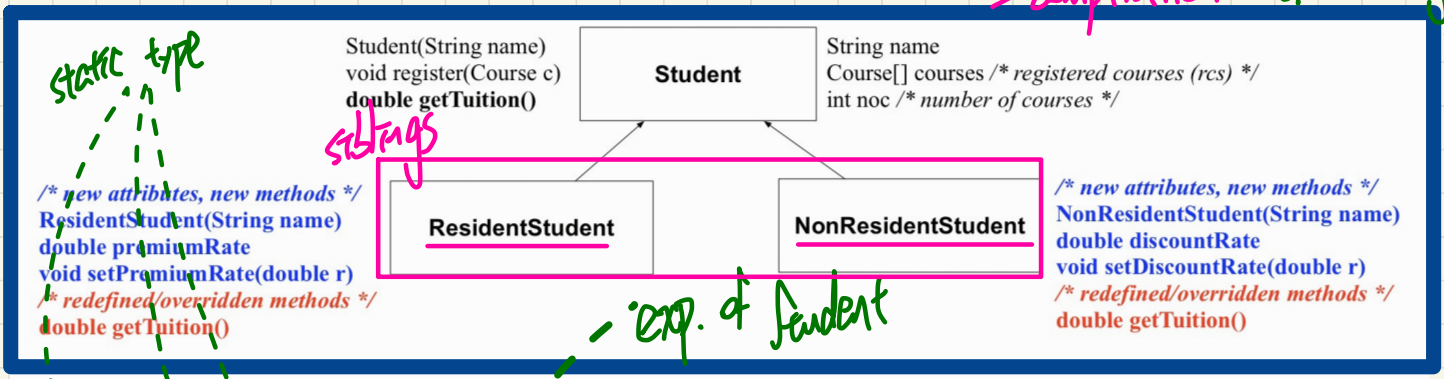
* child classes inherit exp. of their parent.

Student Classes (with inheritance): Expectations

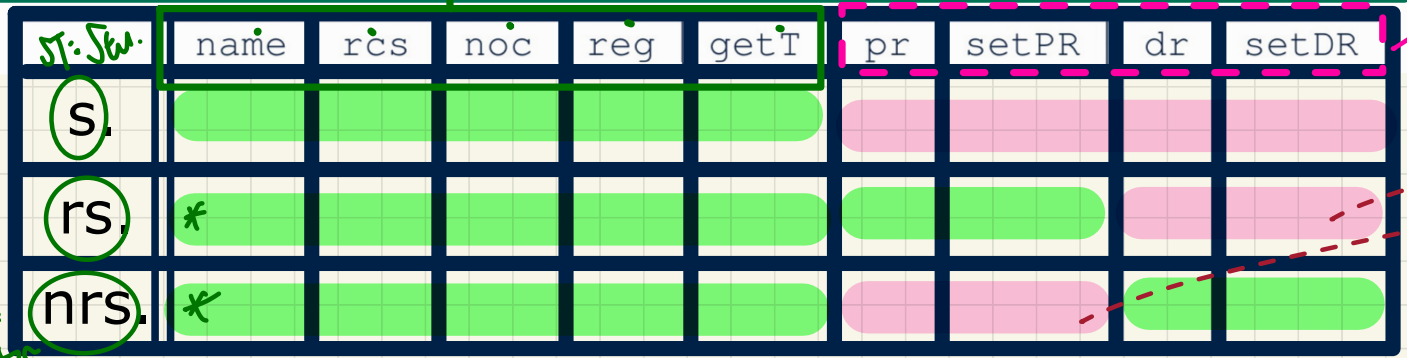
static type

exp. of a ref. variable, given some inheritance hierarchy, determines the range of diff. msgs that can be called on that variable

compilation




```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
```

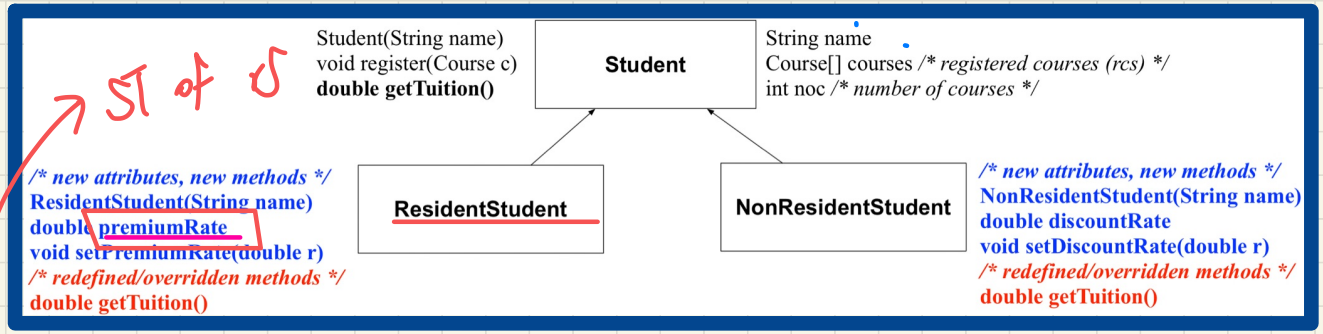


exp. of child classes.

exp. of sibling not expected.

Intuition: Polymorphism

4. rs = s should not compile!




```
1  Student s = new Student("Stella");
2  ResidentStudent rs = new ResidentStudent("Rachael");
3  rs.setPremiumRate(1.25);
4  s = rs; /* Is this valid? */
5  rs = s; /* Is this valid? */
```

1. Assume rs = s compiles.
2. Execute the var. assignment
3. Exp. of rs is RS, including pr. → rs.pr → crash at runtime ∵ Student obj. does not have pr