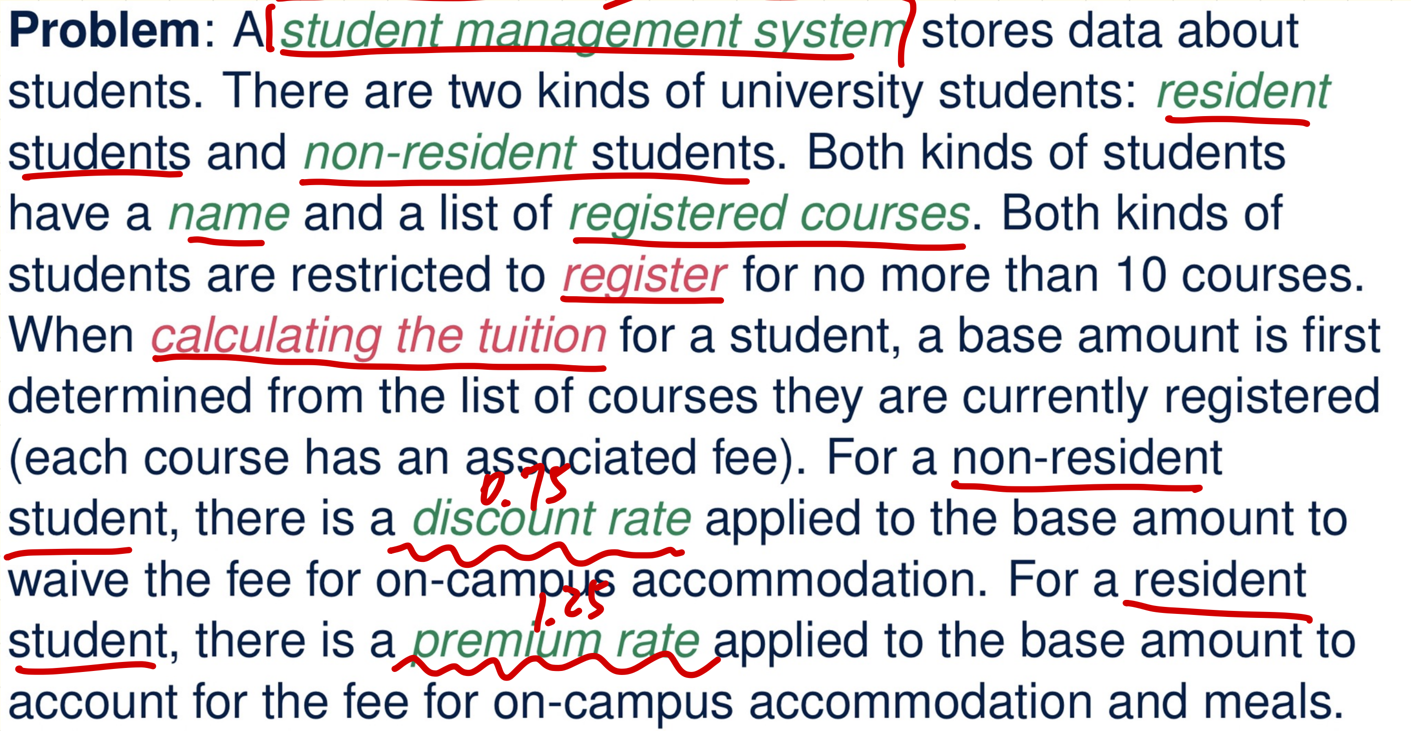
LECTURE 16

MONDAY NOVEMBER 4

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 0.75 *discount rate* applied to the base amount to waive the fee for on-campus accommodation. For a resident student, there is a 1.25 *premium rate* applied to the base amount to account for the fee for on-campus accommodation and meals.

First Design Attempt

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

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

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

repetition

First Design Attempt

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

related to different purposes:

RS

NRS

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

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

Good design?

Judge by Cohesion

all methods or attr. in a single class must be related to a single purpose.

First Design Attempt

```
class Student {
  Course[] courses;
  int noc;         1 : RS
  int kind;        2 : NRS
  double premiumRate;
  double discountRate;
                  1 2 3
  Student (int kind){
    this.kind = kind;
  }
  ...
}
```

3: IS

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

```
double register(Course c){
  int MAX;
  if (this.kind == 1) { MAX = 6; }
  else if (this.kind == 2) { MAX = 4; }
  elseif (this.kind = 3) { . . . }
  if (noc == MAX) { /* Error */ }
  else {
    courses[noc] = c;
    noc++;
  }
}
```

Good design?

Judge by Single Choice Principle

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

change should be done in a single place

V1

Student

The kind

pr

dr

issule of

cohesion

V2

Resident Student

pr

Non Resident Studen

dr

cohesion

repaired.

Testing Student Classes (without inheritance)

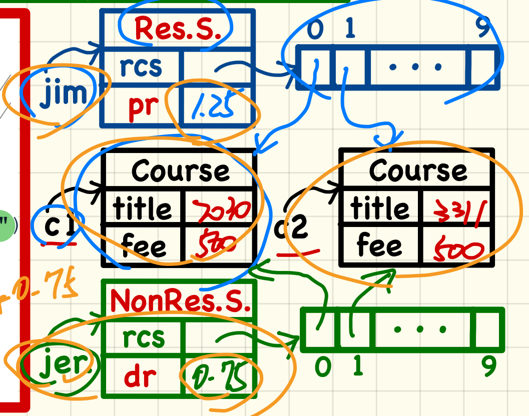
```
class ResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
   double premiumRate;  /* there's a mutator me
   ResidentStudent (String name) {
    this.name = name;
      registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition() {
    double tuition = 0;
    for(int i = 0; i < numberOfCourses; i ++) {
      tuition += registeredCourses[i].fee;
    }
    return tuition * premiumRate;
  }
```

```
class NonResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
   double discountRate;  /* there's a mutator me
   NonResidentStudent (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 * discountRate;
  }
```

```
class StudentTester {
  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());
  }
}
```

1000 * 1.25

1000 * 0.75



Student Classes (without inheritance): Maintenance (1)

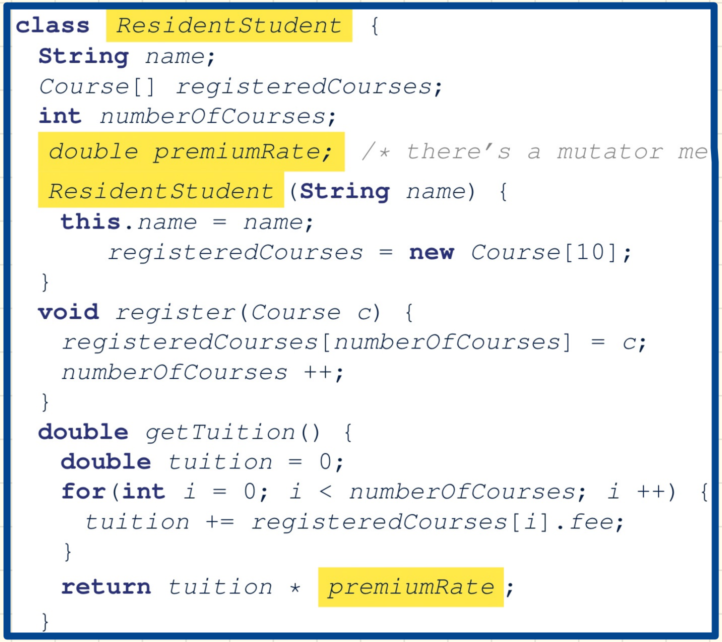
```java
class ResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  double premiumRate; /* there's a mutator me
  ResidentStudent (String name) {
    this.name = name;
      registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition() {
    double tuition = 0;
    for(int i = 0; i < numberOfCourses; i ++) {
      tuition += registeredCourses[i].fee;
    }
    return tuition * premiumRate;
  }
```

```java
class NonResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  double discountRate; /* there's a mutator me
  NonResidentStudent (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 * discountRate;
  }
```

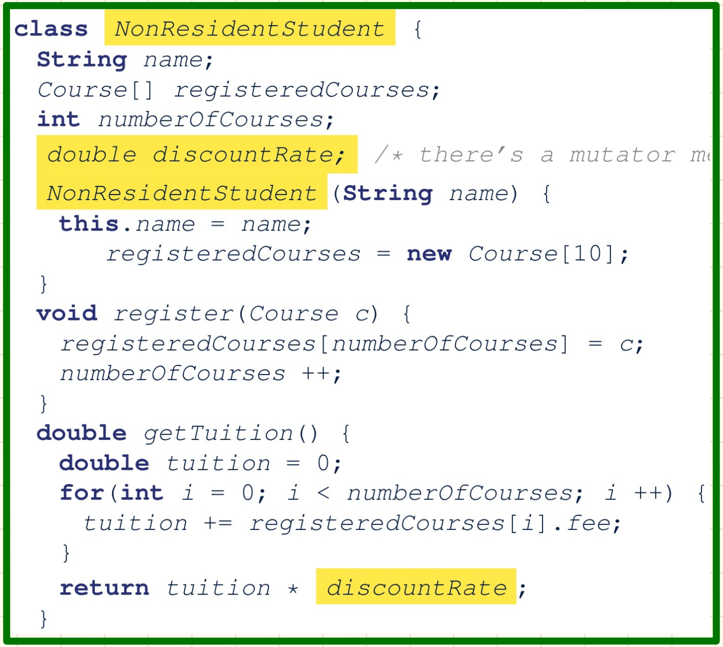
Maintenance: e.g., a new registration constraint

```java
if(numberOfCourses >= MAX_ALLOWANCE) {
  throw new IllegalArgumentException("Too Many Courses");
}
else { … }
```

Student Classes (without inheritance): Maintenance (2)



```
class ResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  double premiumRate; /* there's a mutator me
  ResidentStudent (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 * premiumRate;
  }
```

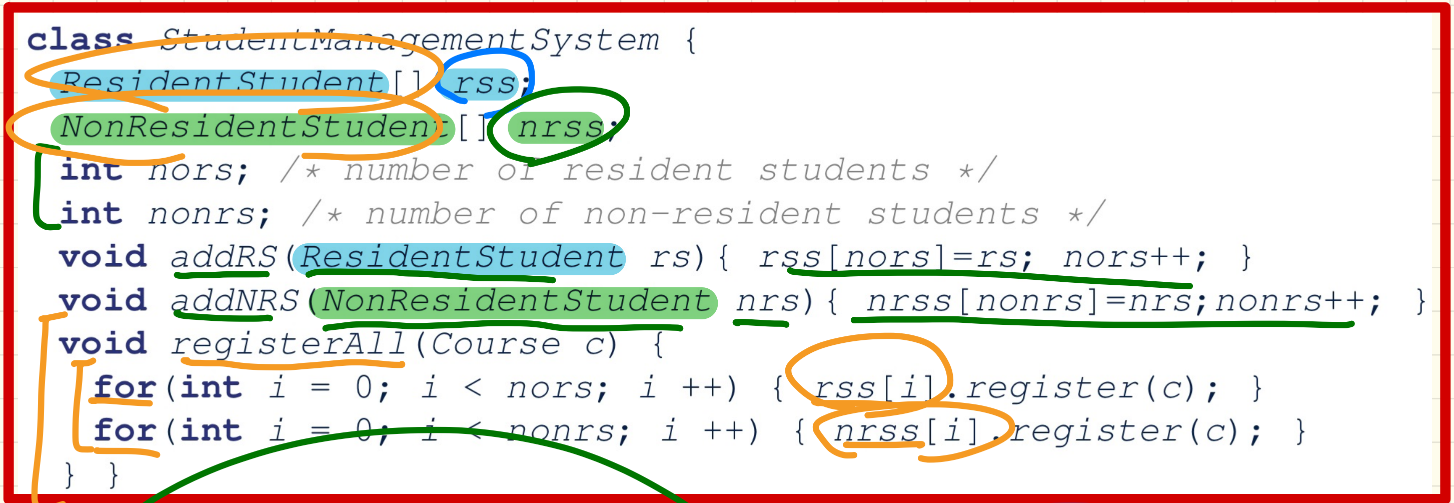


```
class NonResidentStudent {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  double discountRate; /* there's a mutator me
  NonResidentStudent (String name) {
    this.name = name;
        registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition() {
    double tuition = 0;
    for(int i = 0; i < numberOfCourses; i ++) {
      tuition += registeredCourses[i].fee;
    }
    return tuition * discountRate;
  }
```

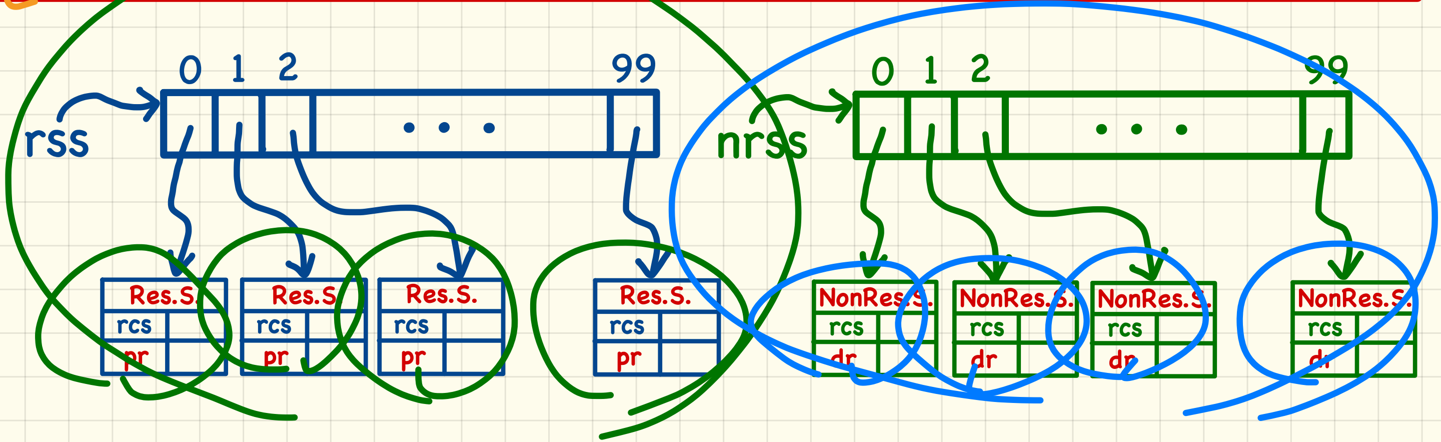
Maintenance: e.g., a new formula for tuition

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

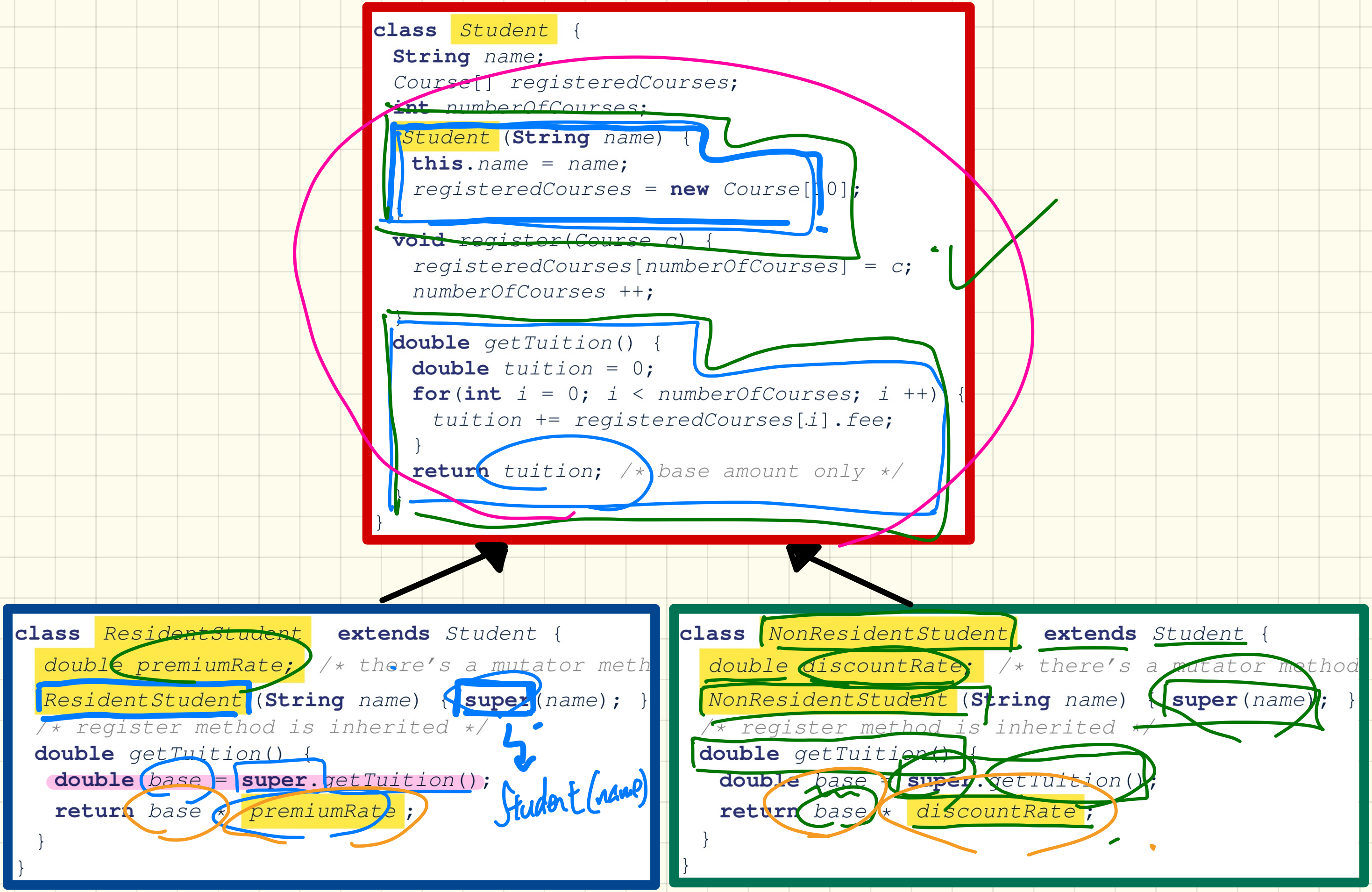
A Collection of Students (without inheritance)



```
class StudentManagementSystem {
  ResidentStudent[] rss;
  NonResidentStudent[] nrss;
  int nors; /* number of resident students */
  int nonrs; /* number of non-resident students */
  void addRS(ResidentStudent rs){ rss[nors]=rs; nors++; }
  void addNRS(NonResidentStudent nrs){ nrss[nonrs]=nrs;nonrs++; }
  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); }
} }
```



Student Classes (with inheritance)

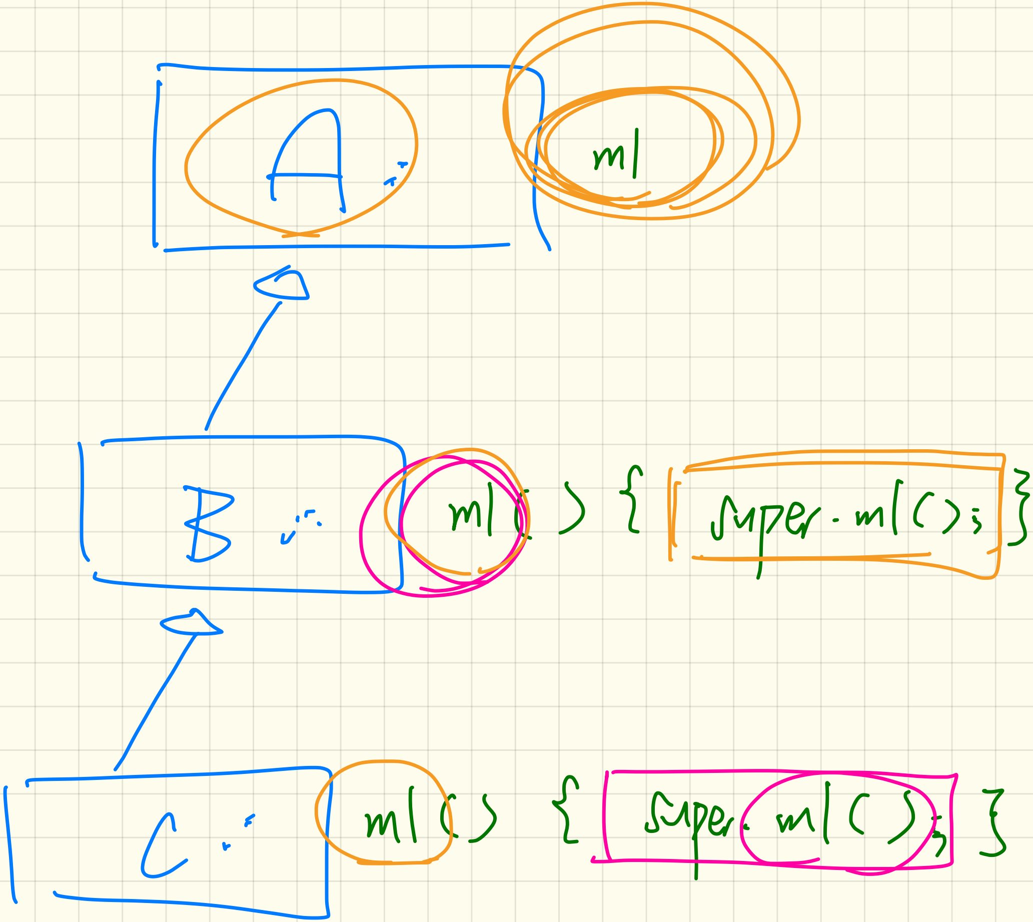


```
class Student {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  Student (String name) {
    this.name = name;
    registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition() {
    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;
  }
}
```

Student(name)

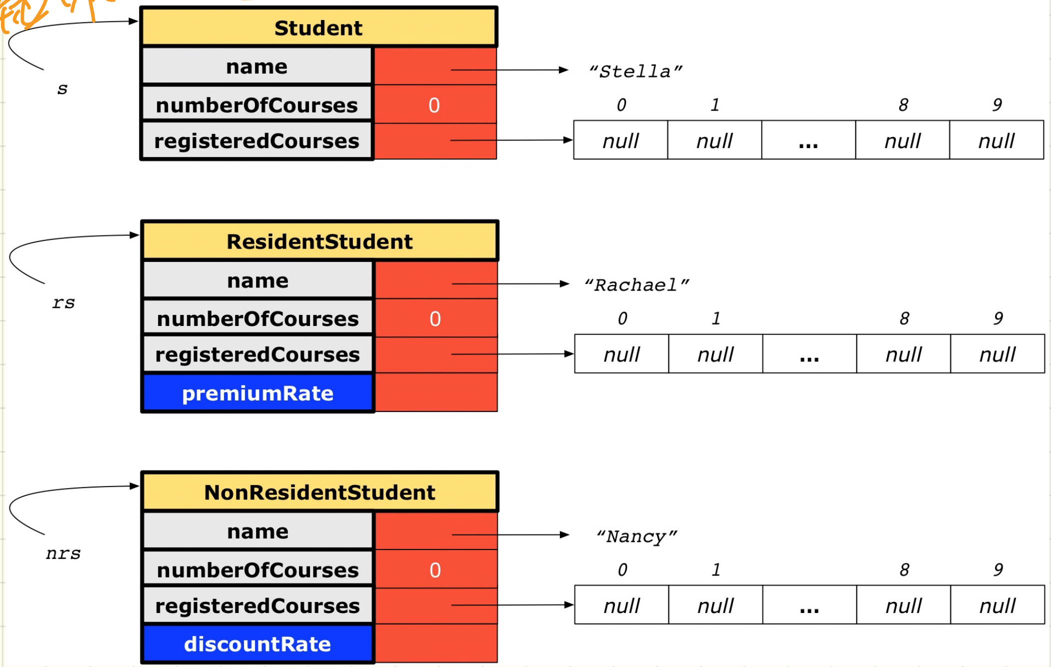
```
class NonResidentStudent extends Student {
  double discountRate; /* there's a mutator method
  NonResidentStudent (String name) { super(name); }
  /* register method is inherited */
  double getTuition() {
    double base = super.getTuition();
    return base * discountRate;
  }
}
```


A :
m1
B :
m1( ) { super.m1( ); }
C :
m1( ) { super.m1( ); }

Visualizing Parent and Child Objects

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

static type



s

Student

name	→ "Stella"
numberOfCourses	0
registeredCourses	→

0	1		8	9
null	null	...	null	null

rs

ResidentStudent

name	→ "Rachael"
numberOfCourses	0
registeredCourses	→
premiumRate	

0	1		8	9
null	null	...	null	null

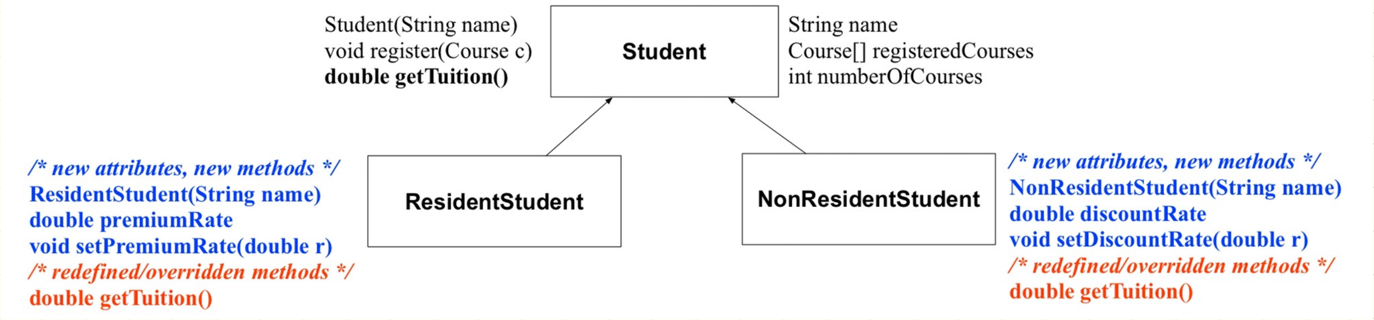
nrs

NonResidentStudent

name	→ "Nancy"
numberOfCourses	0
registeredCourses	→
discountRate	

0	1		8	9
null	null	...	null	null

Testing Student Classes (with inheritance)



Student(String name)
void register(Course c)
double getTuition()

Student

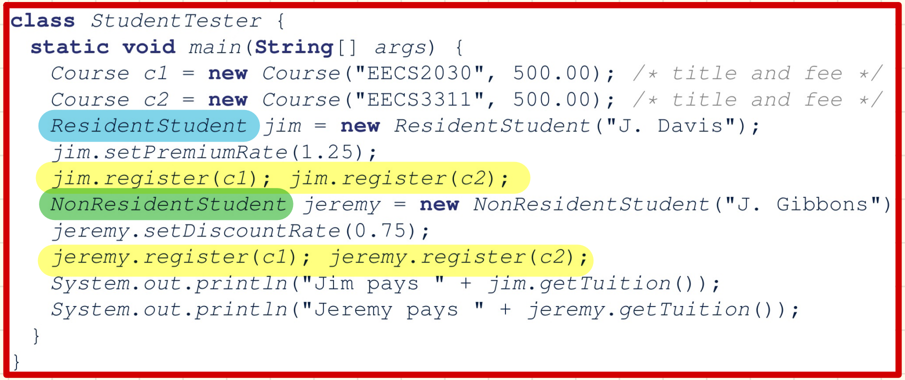
String name
Course[] registeredCourses
int numberOfCourses

/ new attributes, new methods */*
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/ redefined/overridden methods */*
double getTuition()

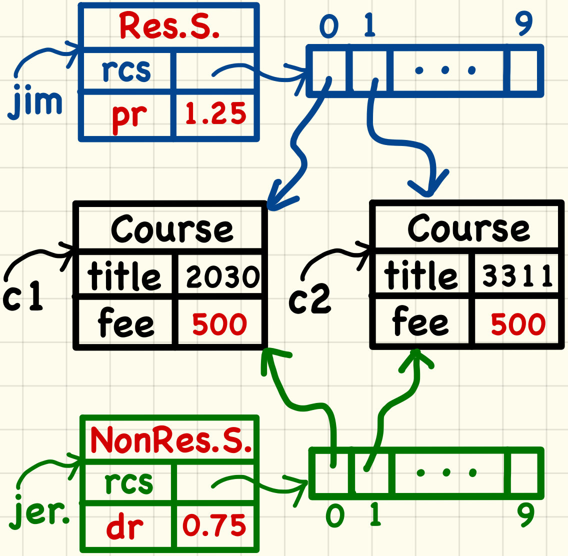
ResidentStudent

NonResidentStudent

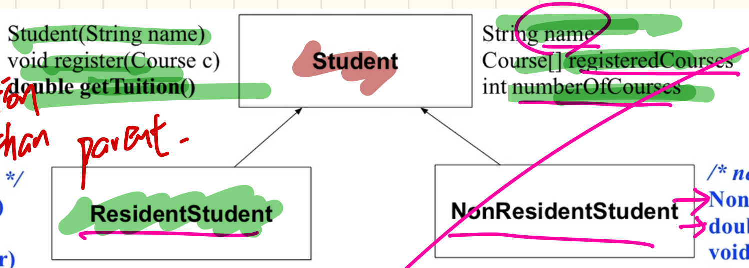
/ new attributes, new methods */*
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/ redefined/overridden methods */*
double getTuition()



```
class StudentTester {
  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): Expectations



Student(String name)
void register(Course c)
double getTuition()

String name
Course[] registeredCourses
int numberOfCourses

child class has more expectation than parent.

common attributes/meth. inherited.

/ new attributes, new methods */*
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/ redefined/overridden methods */*
double getTuition()

overriding / (redefinition).

/ new attributes, new methods */*
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/ redefined/overridden methods */*
double getTuition()

static types

s.pr ? X ∵ pr is only in RS

subclass-specific attributes

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

	name	rcs	noc	reg	getT	pr	setPR	dr	setDR
s.	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥
rs.	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥
nrs.	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟩	🟩

T

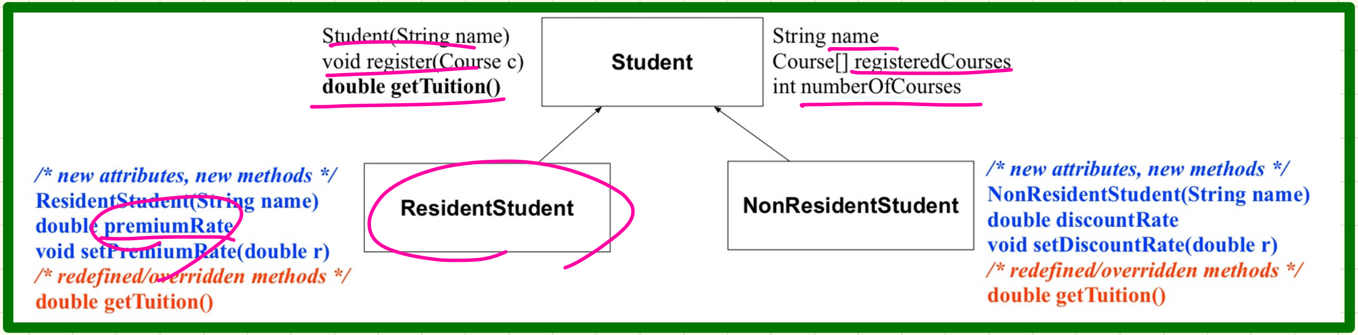
v : · — — —

??

$v \cdot 1$

expectation

Intuition: Polymorphism


Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] registeredCourses
int numberOfCourses
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

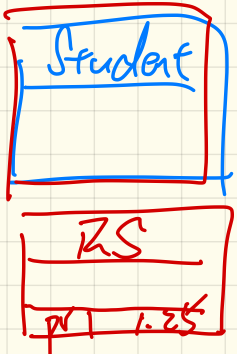
```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
rs.setPremiumRate(1.25);
s = rs; /* Is this valid? */
rs = s; /* Is this valid? */
```

Expectation on rs ? s

rs.pr

ST: RS

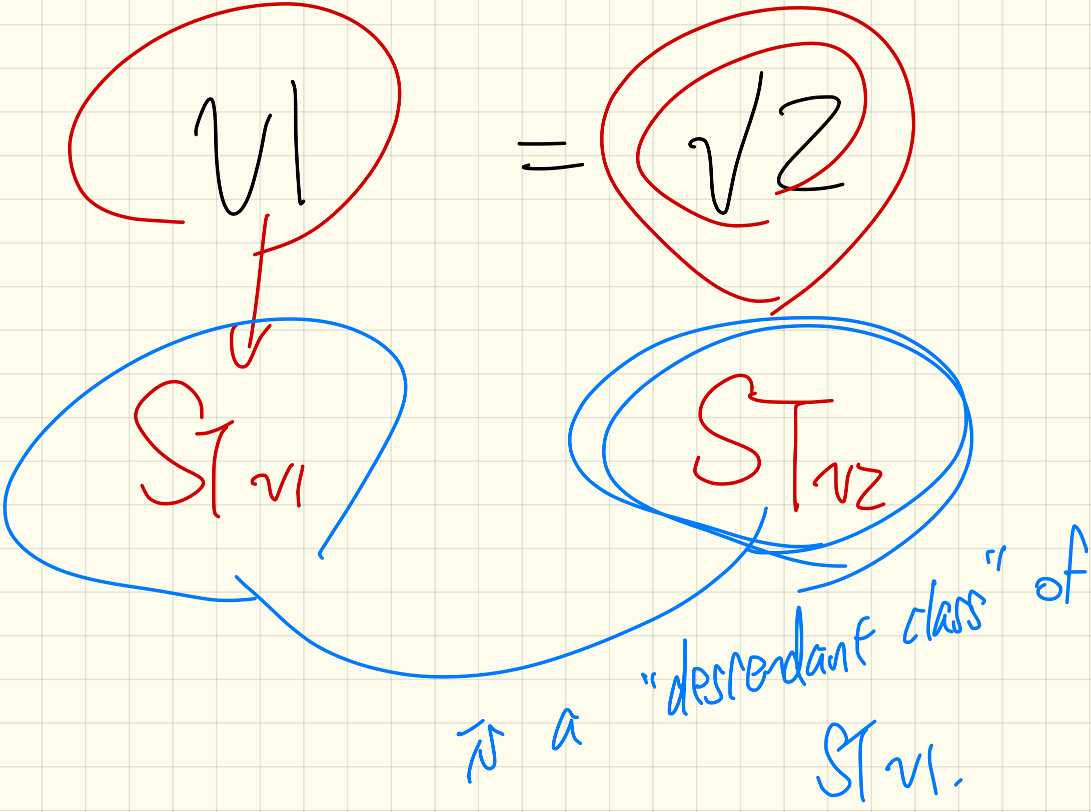
executing this AFTER: crash!



Assume rs = s compiled.

Expecting rs = s

re-direct rs.



is a "descendant class" of ST v1.