Lecture 17 - Nov. 7

Inheritance

Implementing a Child Class: Principles
Visibility: Class, Attribute/Method
Static Types: Expectations
Polymorphism: Intuition

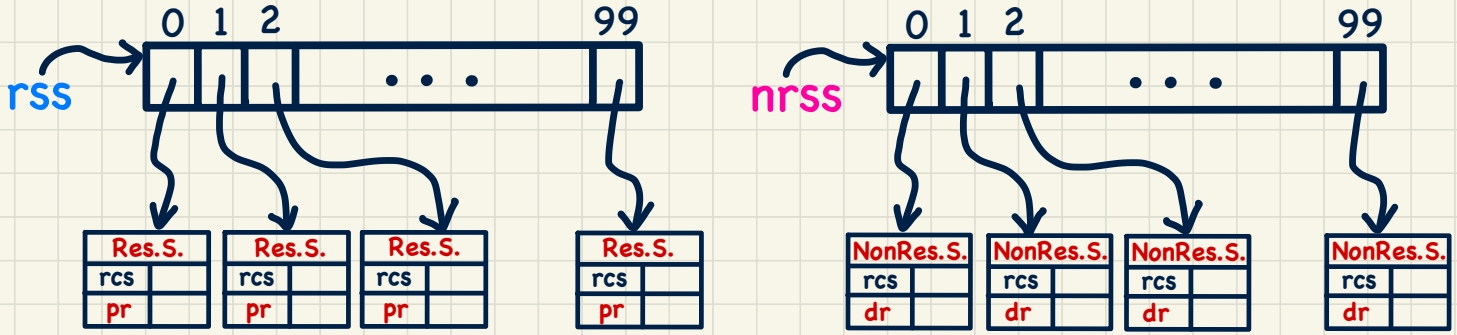
Announcements/Reminders

- Lab4 released (ProgTest3 on November 20)
- Guide & Questions for WrittenTest2 released
- Materials for In-Lab Demo on Inheritance released
- ProgTest2 results & feedback Monday Nov 11

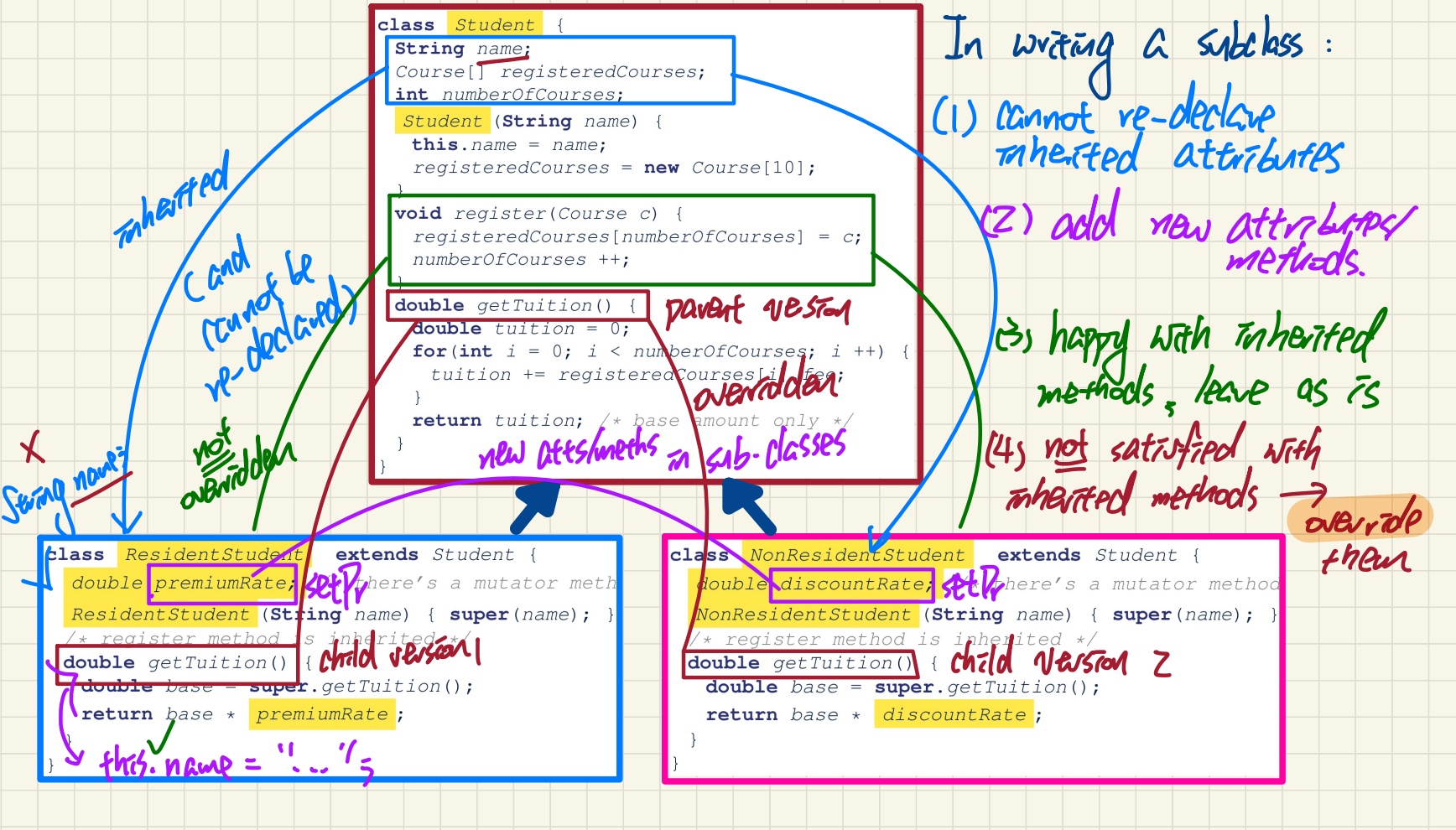
A Collection of Students (without inheritance)

```
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); }
  }
}
```

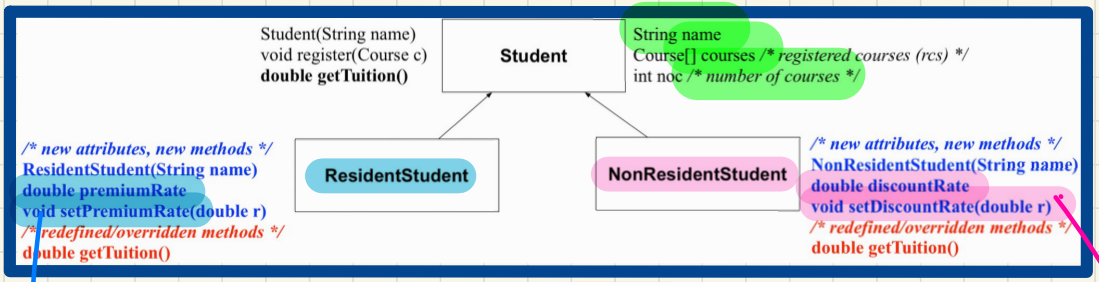
for loops correspond to # kinds of students



Recall: Student Classes (with inheritance)



Visualizing Parent and Child Objects

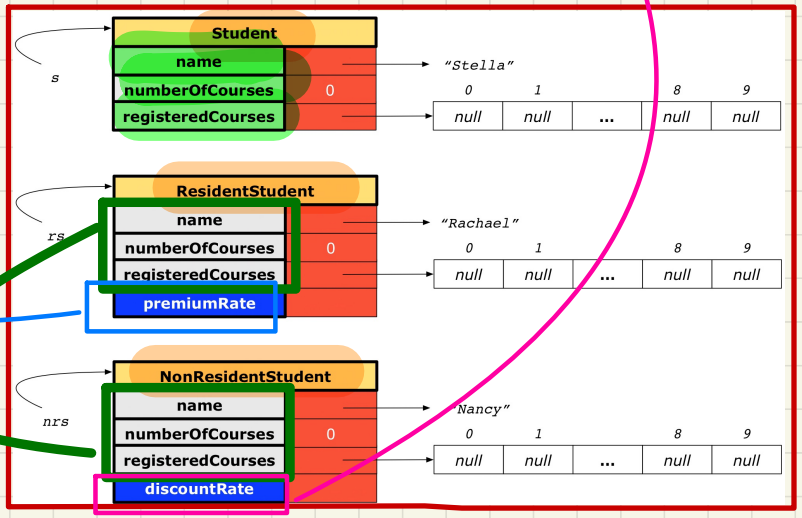


Inheritance Hirarchy

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

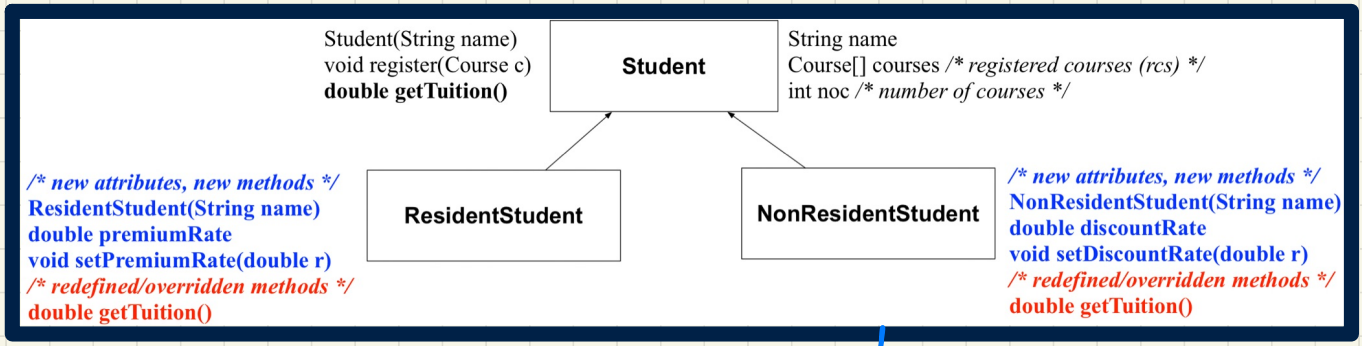
Declaring Static Types

Runtime Object Structure



inherited from Student

Testing Student Classes (with inheritance)



in-lab demo

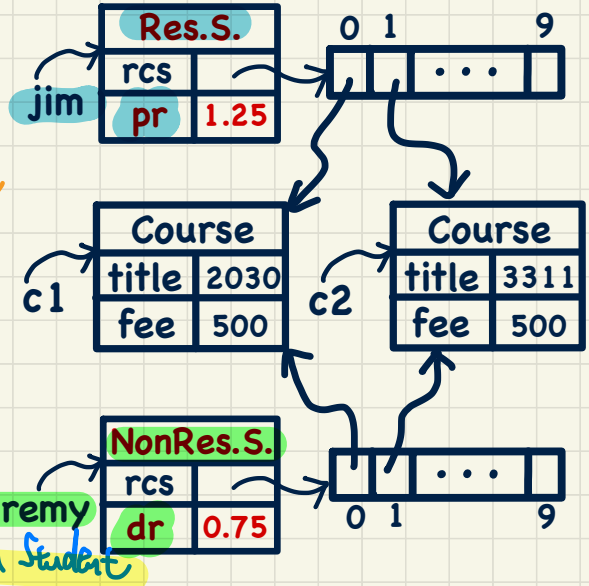
```
public class StudentTester {
  public static void main(String[] args) {
    Course c1 = new Course("EECS2030", 500.00); /* title and fee */
    Course c2 = new Course("EECS3311", 500.00); /* title and fee */
    ResidentStudent jim = new ResidentStudent("J. Davis");
    jim.setPremiumRate(1.25);
    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());
  }
}
```

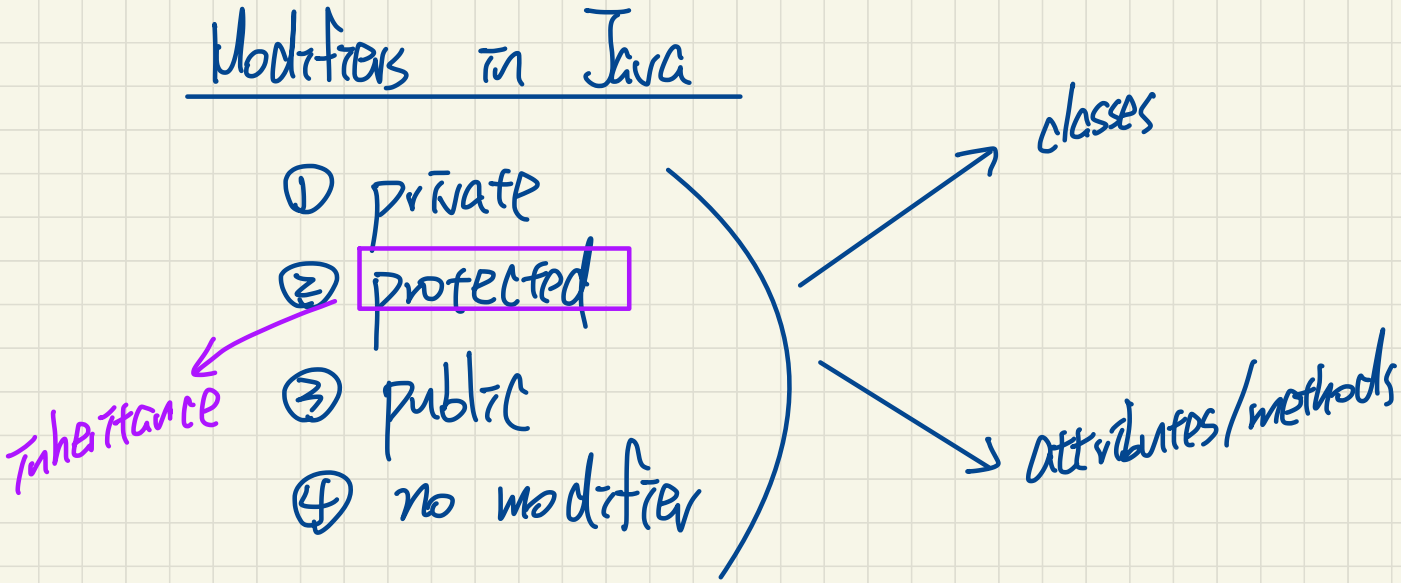
→ invoke Student constructor

version in Student invoked

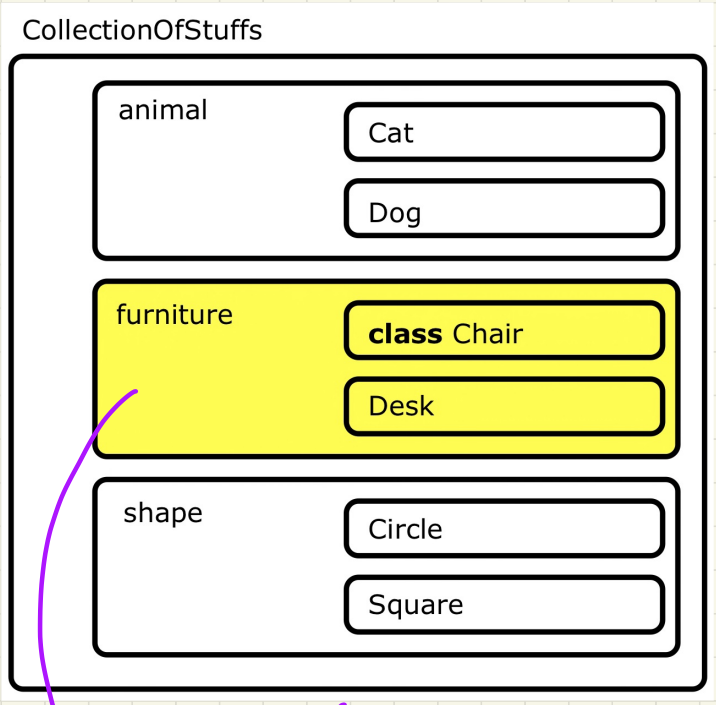
invoke ① the specific child version

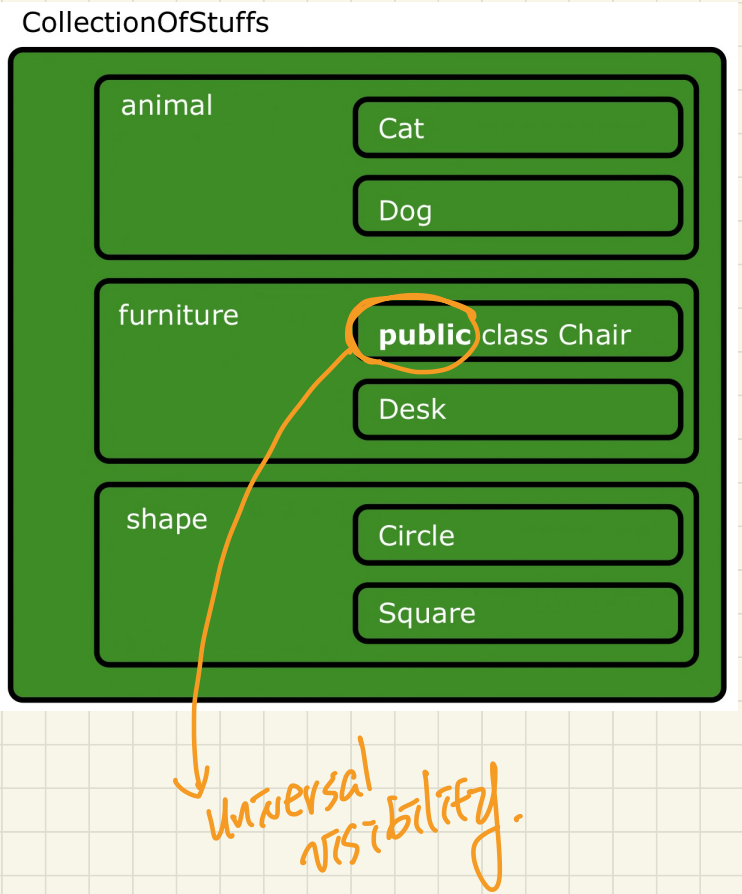
② both first invoke getTuition from Student



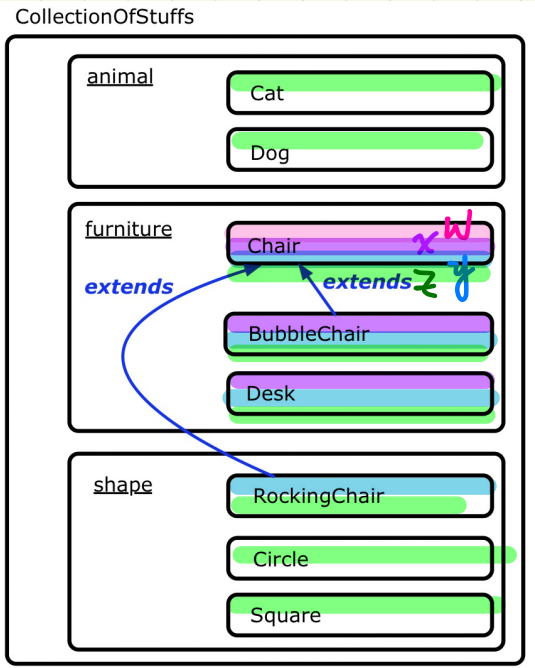

Modifiers in Java
1 private
2 protected
3 public
4 no modifier
Inheritance
classes
attributes/methods

Visibility: Classes





Visibility: Attributes and Methods

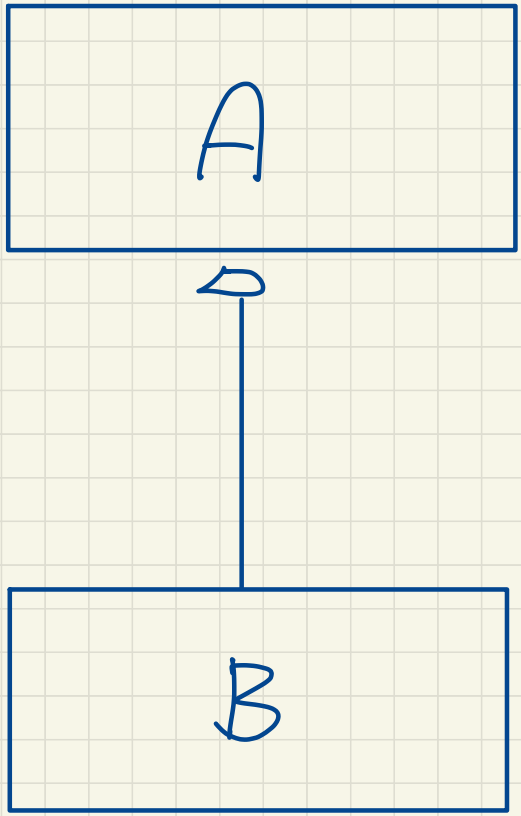



```
public class Chair {
   private int w;
   int x;
   protected int y;
   public int z;
}
```

not inherited to subclasses

as if:
(1) pkg visib.
(2) subclass (in diff. pkg.)

	Class	Package	Subclass (same pkg)	Subclass (different pkg)	Non-Subclass (across Project)
public					
protected					
no modifier					
private					

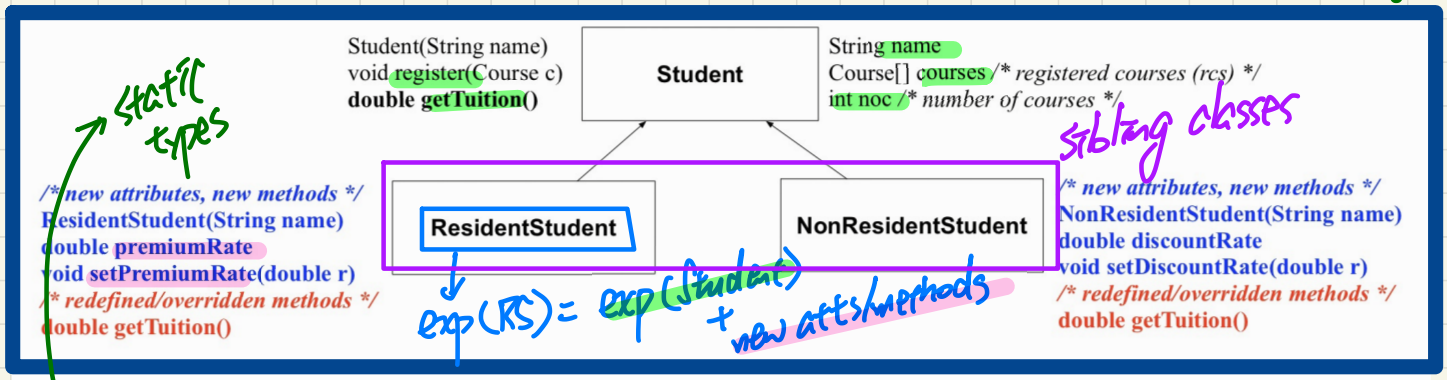


(1) All attributes private.

(2) as you discover necessary, make attrs. protected

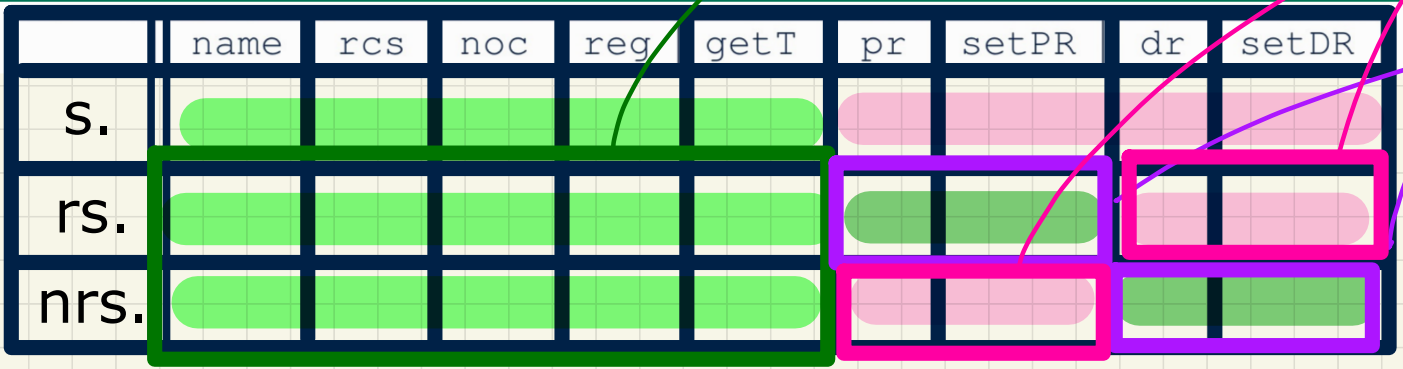
Student Classes (with inheritance): Expectations

exp. of a ref. variable determines the range of attrs/methods that can be called on that variable




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

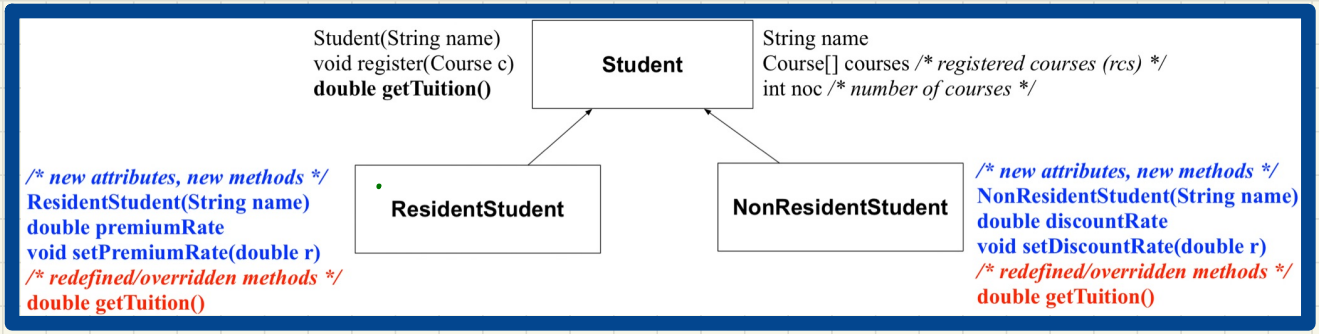
inherited exp from parent



sibling exp.

new exp introduced in subclasses

Intuition: Polymorphism




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

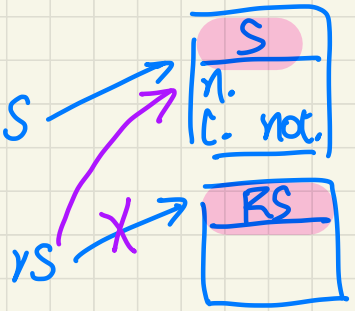
ST of subclass

ST: parent class

1. Assume rs = s compiles.

2. Execute the re-assignment.

3. Exp. of rs: *

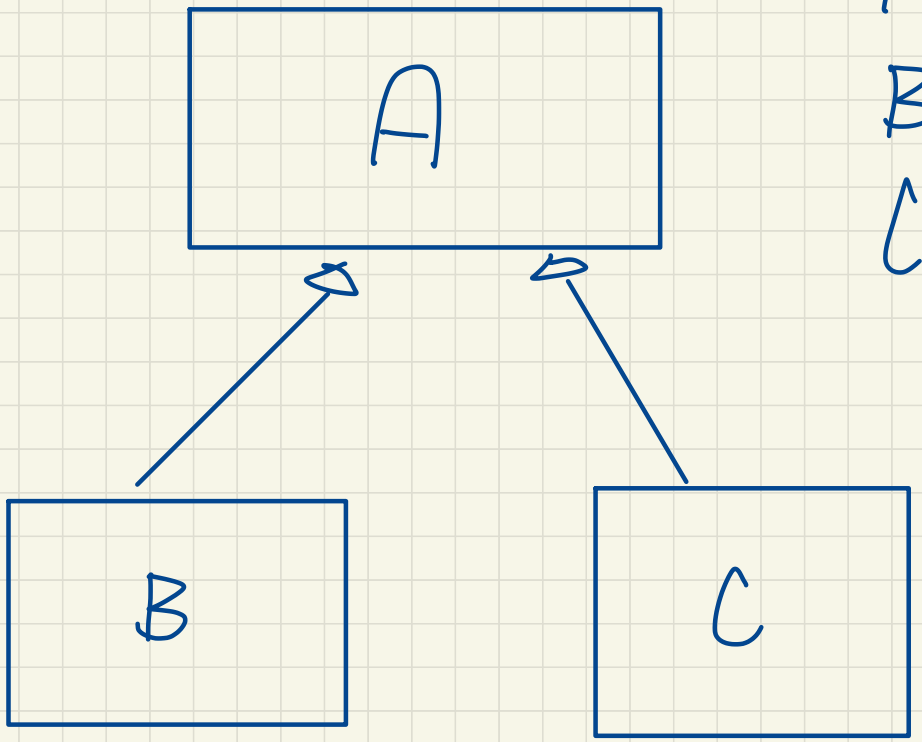


4. Assumption was wrong:

rs = s not compilable

rs.name
rs.noc
rs.courses
rs.pr

Student obj does not have pr




```
A o1 = new A();
B o2 = new B();
C o3 = new C();

o1 = o2;
o1 = o3;
o2 = o3;
o2 = o1;
o3 = o1;
o3 = o2;
```