

Lecture 17 - Nov 5

Inheritance

Modifiers in Java
Static Types and Expectations

Announcements/Reminders

- Today's class: notes template posted
- **Lab4** to be released
- Tracing Exercises: assertEquals and Person, PersonCollector

Student Classes (with inheritance)

thrs vs. super
 ↳ current class
 ↳ common / parent
 ↳ super / parent
 ↳ class
 ↳ follow
 ↳ extends

class
Common
super/parent
class

follow
extends to parent
class

A Student is the class
Common Parent of RS and NRS

base
class.

back
and -

overriden
versions

child/sub class $\xrightarrow{\text{extends}}$ parent/super class

child/sub class
specific attributes

no need
to redeclare
attrs / methods
from parent
class (inheritance)

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

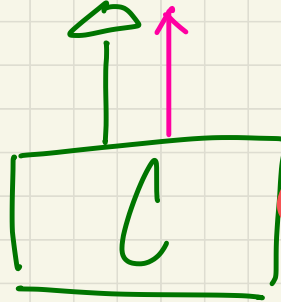
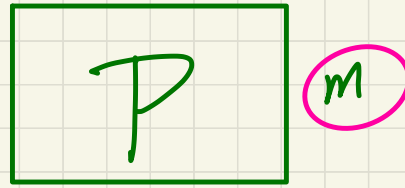
Student
super (name);

v2 calling v1.

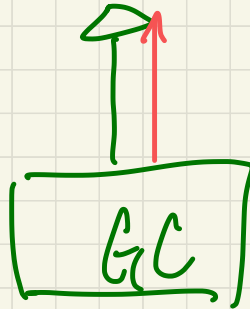
apply pr to base cnt

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

appx dr to
3000
4m

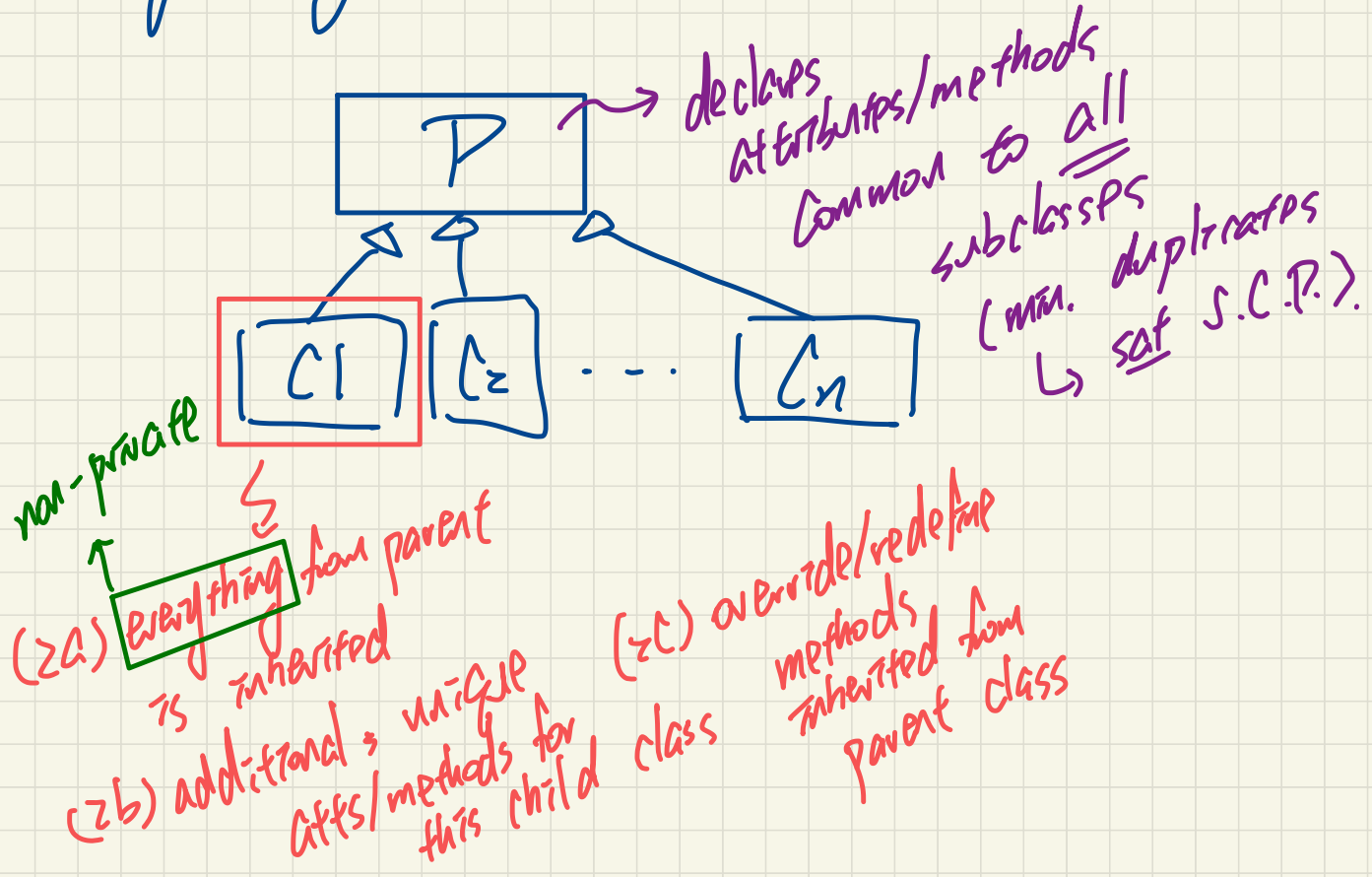


m^* overrides
↳ $\text{super.m}()$ version in P.



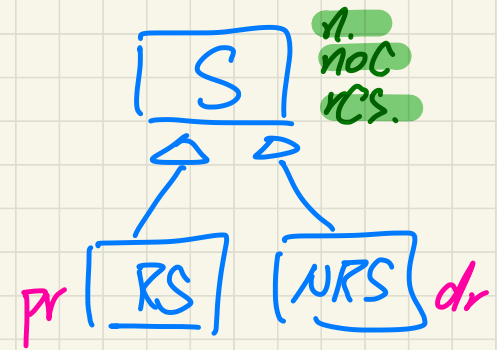
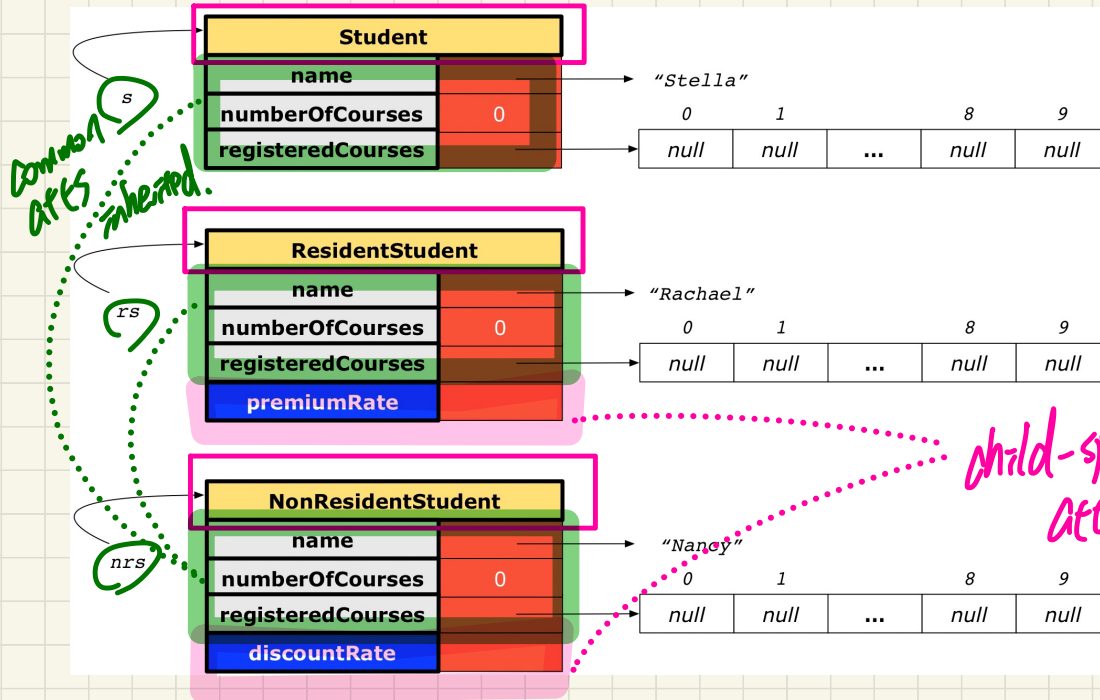
m^* overrides
↳ $\text{super.m}()$ version in C.
↳ ~~$\text{super.super.m}()$~~

When programming with inheritance:



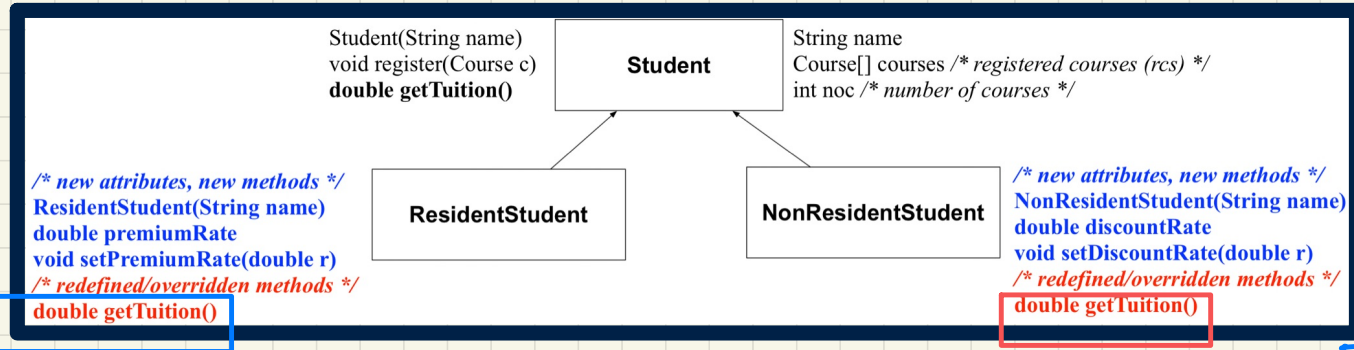
Visualizing Parent and Child Objects D.T.s

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



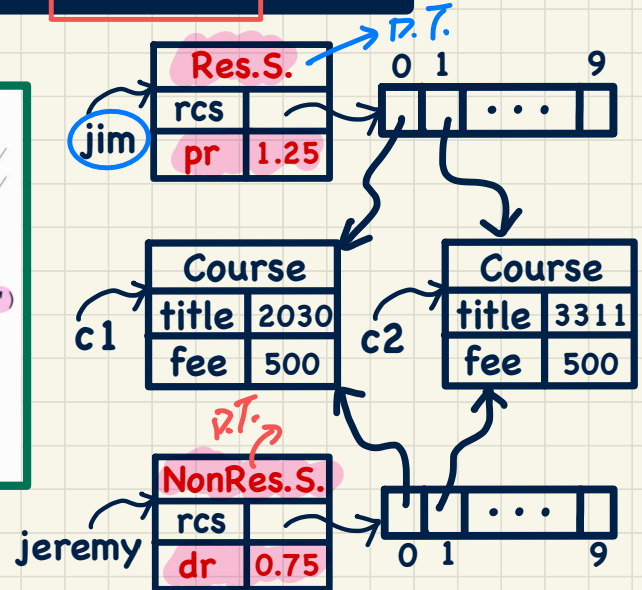
child-specific attributes

Testing Student Classes (with inheritance)



```

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());
    }
}
  
```



Modifiers in Java

visibility.

① private

② protected

③ no modifier

④ public

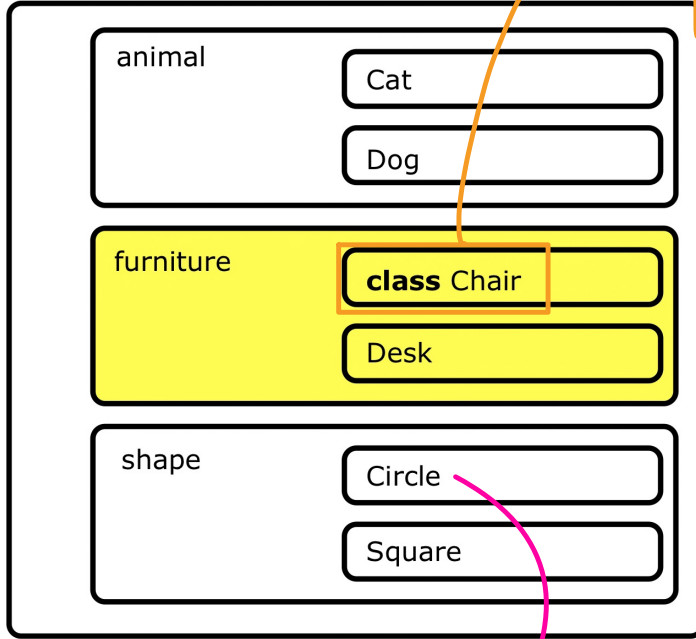
inheritance
(e.g. Lab 4)

↳
classes.

attributes / methods

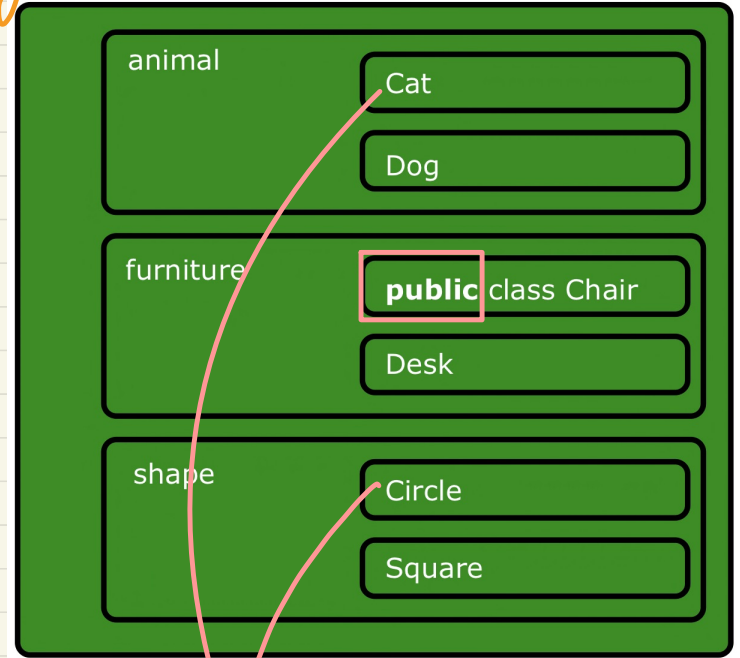
Visibility: Classes

CollectionOfStuffs



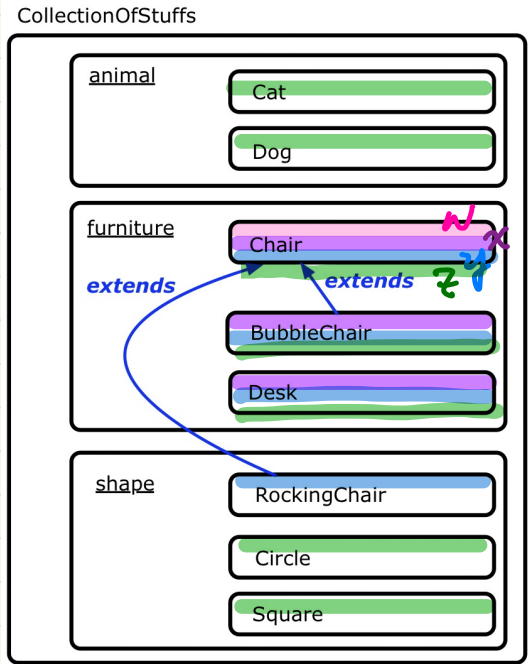
X Chair C's

CollectionOfStuffs



Chair C's ✓

Visibility: Attributes and Methods

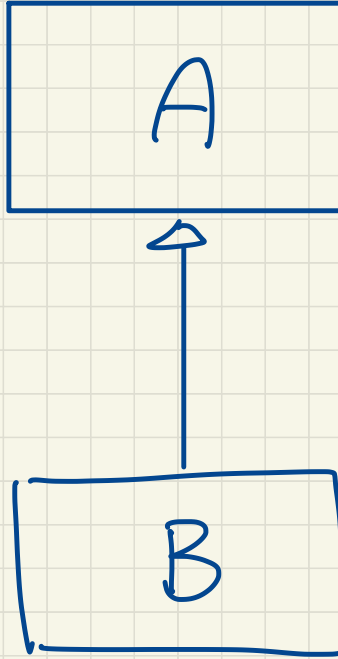


```

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

package level + inheritance (same or not package)

	CLASS	PACKAGE	SUBCLASS (same pkg)	SUBCLASS (different pkg)	NON-SUBCLASS (across Project)
public	Green	Green	Green	Green	Green
protected	Green	Green	Green	Green	Red
no modifier	Green	Green	Green	Red	Red
private	Green	Red	Red	Red	Red



(1) declare all attributes to be private

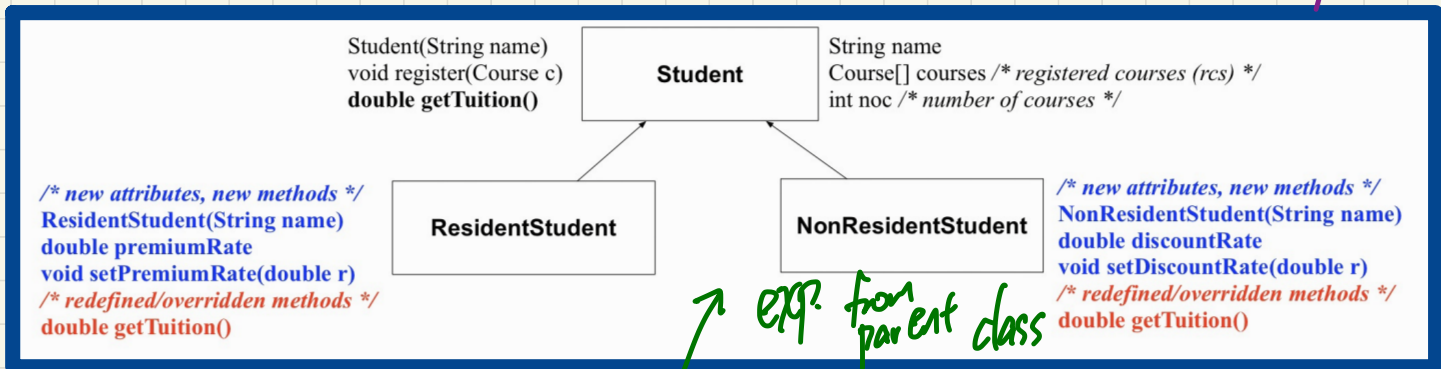
↳ not inherited.

(2) those attributes that should be inherited to B, make them protected.

↓
or no modifiers.

* each child class inherits exp. of its parent. range of attr/method callable static/declared type

Student Classes (with inheritance): Expectations



expectations of a ref. var. are determined by its static type

```

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

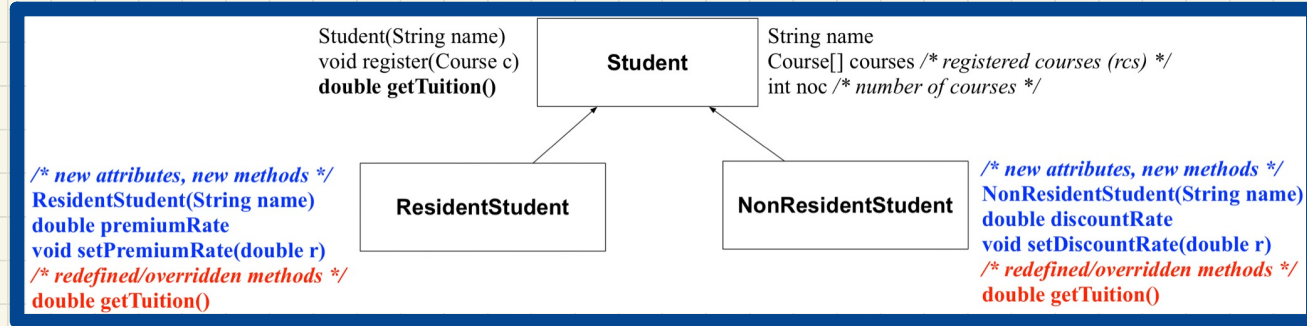
st: S.	name	rcs	noc	reg	getT	pr	setPR	dr	setDR
S.									
rs.									
nrs.									

st: S.
rs: RS
nrs: NRS

child-specific exp.

Intuition: Polymorphism

4. assumption is wrong
! $rs = s$ should not compile.

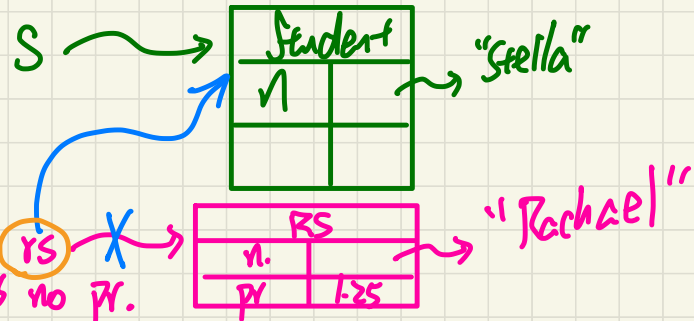


```
1 Student s = new Student("Stella");
2 ResidentStudent rs = new ResidentStudent("Rachael");
3 rs.setPremiumRate(1.25);
4 s ✓ rs; /* Is this valid? */
5 rs ✗ s; /* Is this valid? */
```

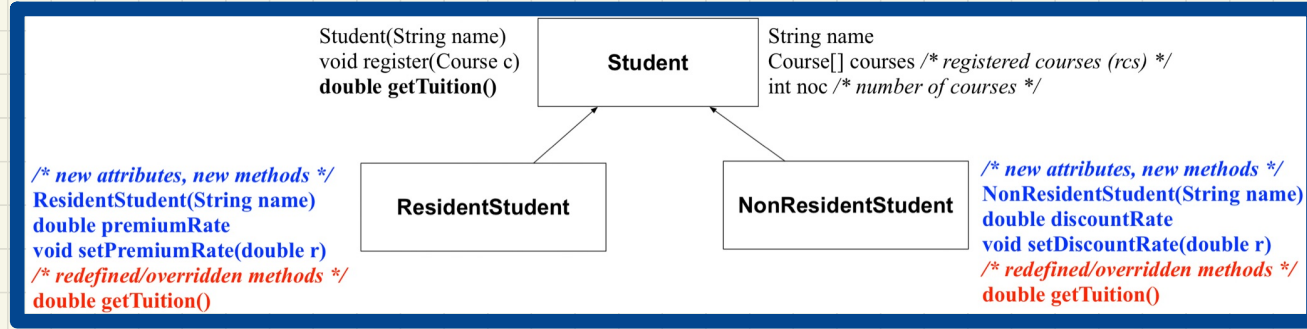
1. Assume $rs = s$ compiles.

2. Execute the var assignment.

3. Exp of rs ' ST (RS) includes: pr
 $rs.pr \rightarrow$ crash ! Student has no pr.



Intuition: Dynamic Binding

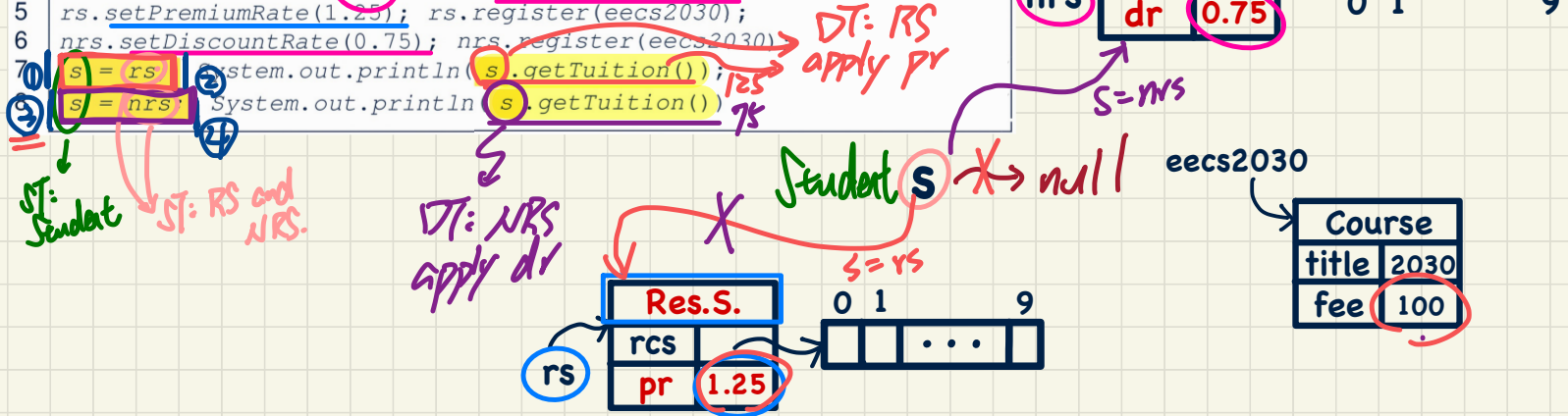


var s

step	ST	DT
①	Student s	null
②		RS
③		RS
④		NRS

```

1 Course eecs2030 = new Course("EECS2030", 100.0);
2 Student s;
3 ResidentStudent rs = new ResidentStudent("Rachael");
4 NonResidentStudent nrs = new NonResidentStudent("Nancy");
5 rs.setPremiumRate(1.25); rs.register(eecs2030);
6 nrs.setDiscountRate(0.75); nrs.register(eecs2030);
7 s = rs; System.out.println(s.getTuition());
8 s = nrs; System.out.println(s.getTuition());
    
```



Compilation: static type

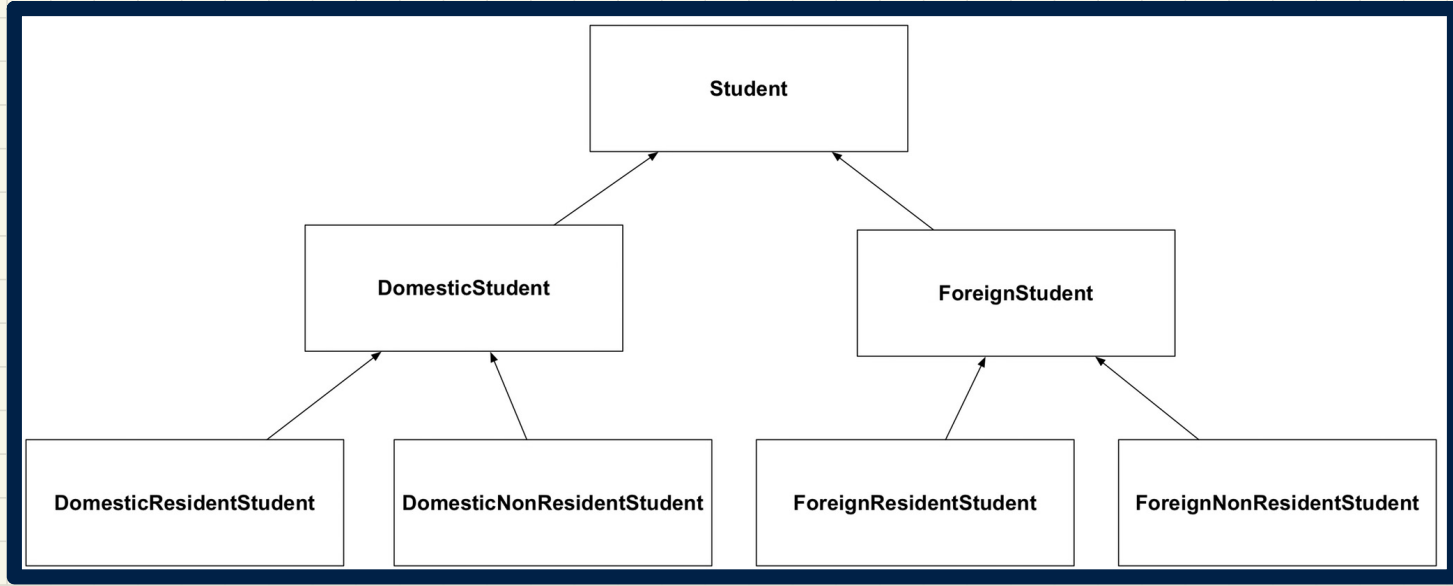
runtime: dynamic type

e.g. version
of method
called

e.g. exception

↓ `ClassNotFoundException`.

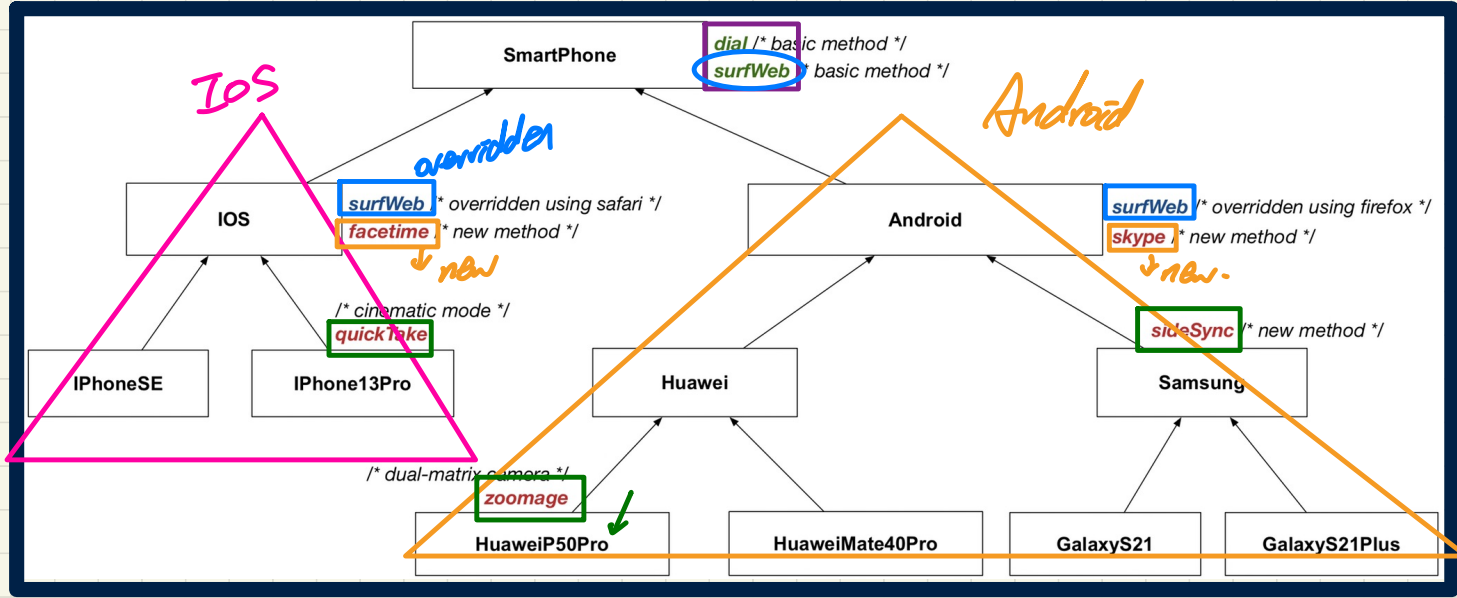
Multi-Level Inheritance Hierarchy: Students



Reflections:

- For **Design 1**, how many encodings to check for each method?
 - For **Design 2**, how many arrays to store for SMS?
 - For **Design 3**, where are common attributes/methods stored?
- kind var? how many classes*
- extends!*

Multi-Level Inheritance Hierarchy: Smartphones



Reflections:

- For Design 1, how many encodings to check for each method?
- For Design 2, how many arrays to store for SMS?
- For Design 3, where are common attributes/methods stored?