Monday Nov. 5

Lecture 16

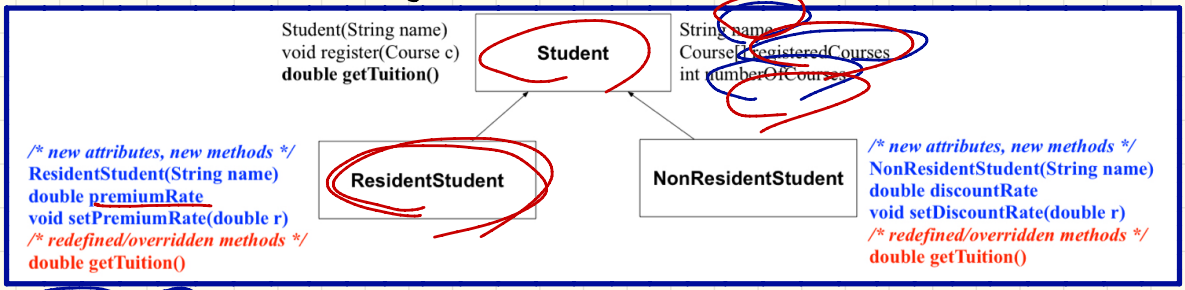
Review: Student Classes (with inheritance)

```
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 method
  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 method
  NonResidentStudent (String name) { super(name); }
  /* register method is inherited */
  double getTuition() {
    double base = super.getTuition();
    return base * discountRate;
  }
}
```

Review: Visualizing Parent and Child Objects



Inheritance Hierarchy

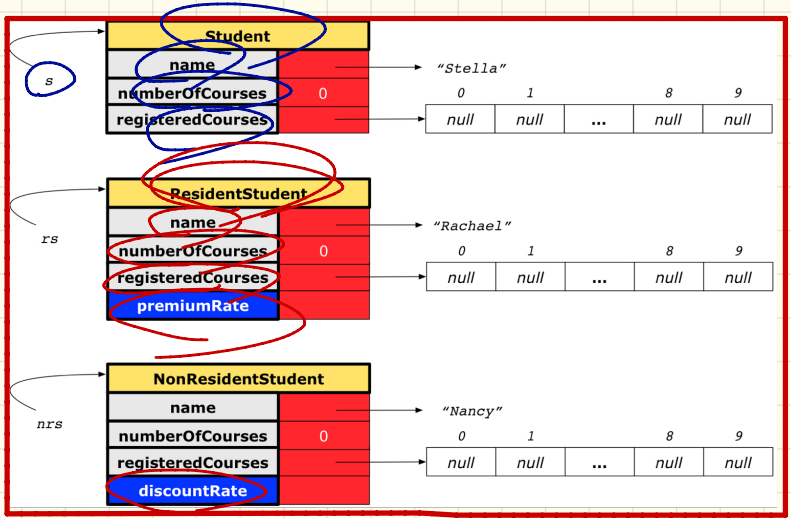
```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
```

Declaring Variable

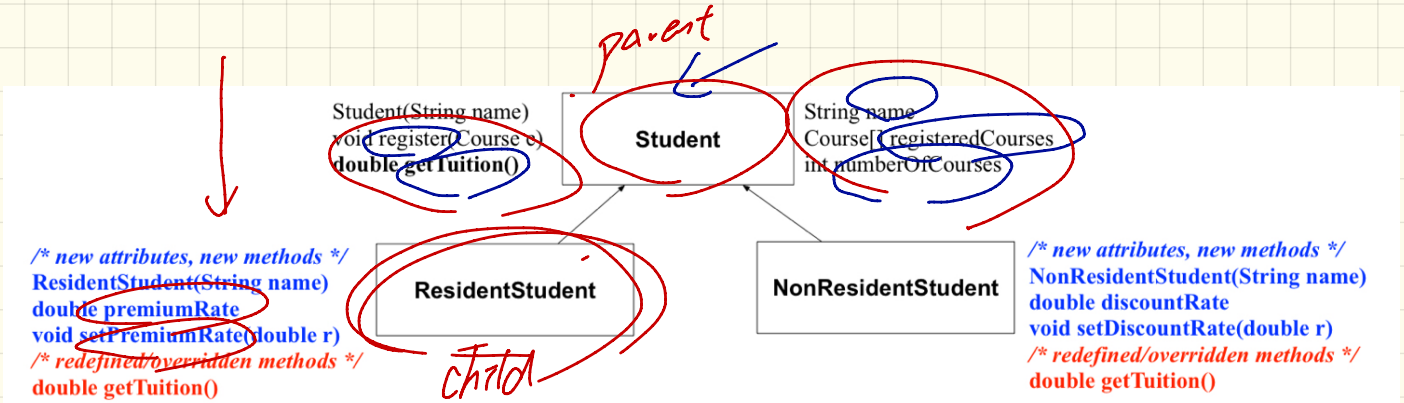
↓

static type

Runtime Object Structure



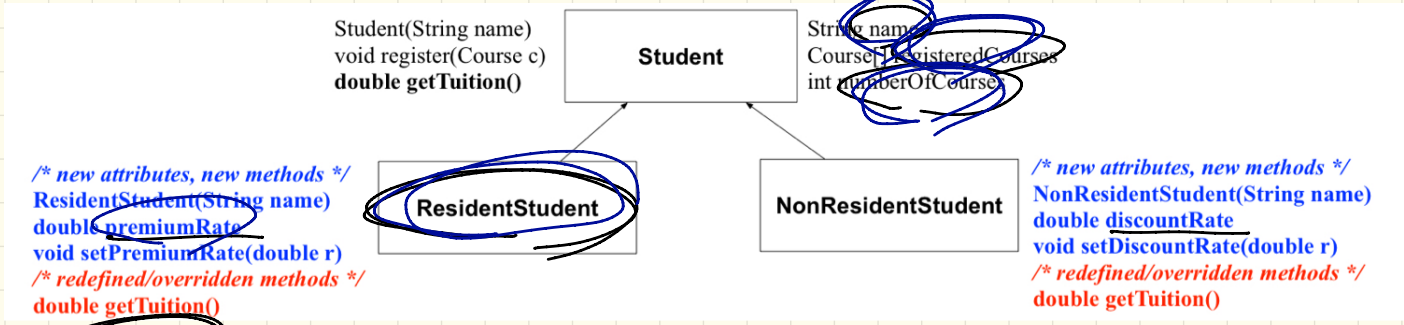
Review: Static Types and Expectations




```
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.	✓					×		✓	

Intuition: Polymorphism




```
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? */
```

Compilation

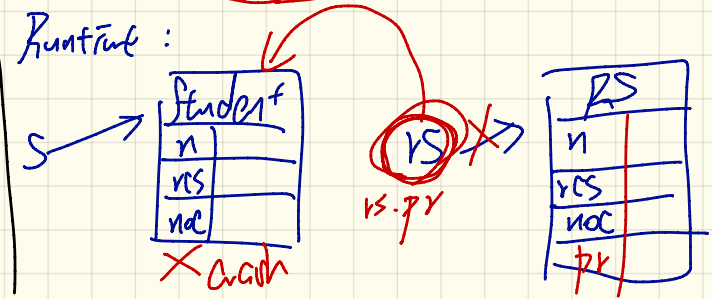
should not compile

Expectations

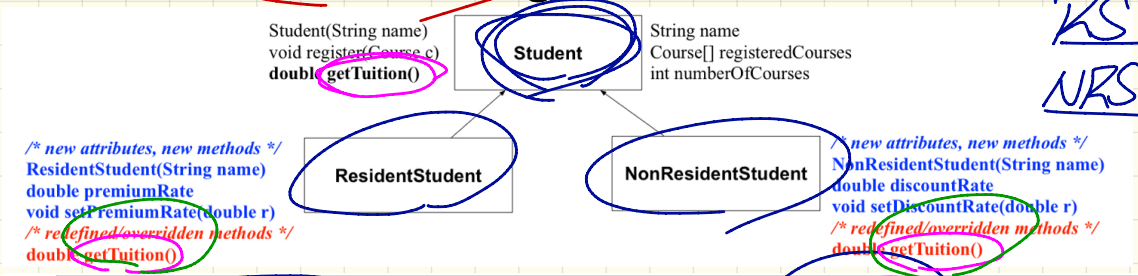
s.name	s.noc	s.pr
s.rcs		s.dr X
rs.name	rs.pr	
rs.rcs	rs.dr X	
rs.noc		

Assume rs = s compiled

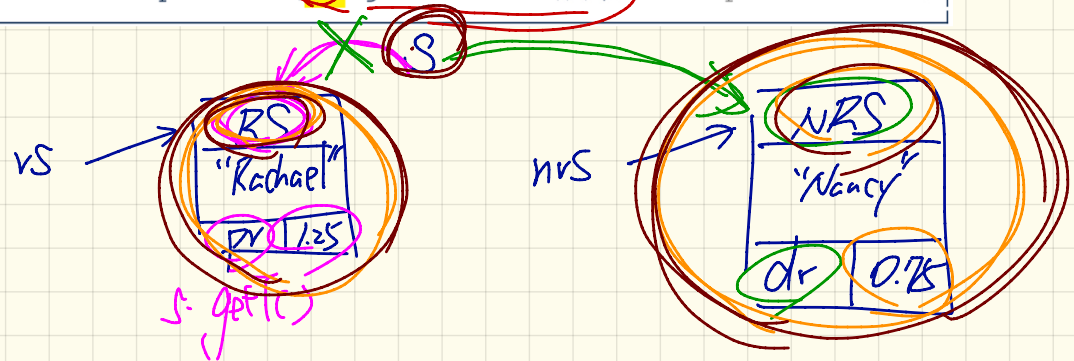
Runtime:



Intuition: Dynamic Binding

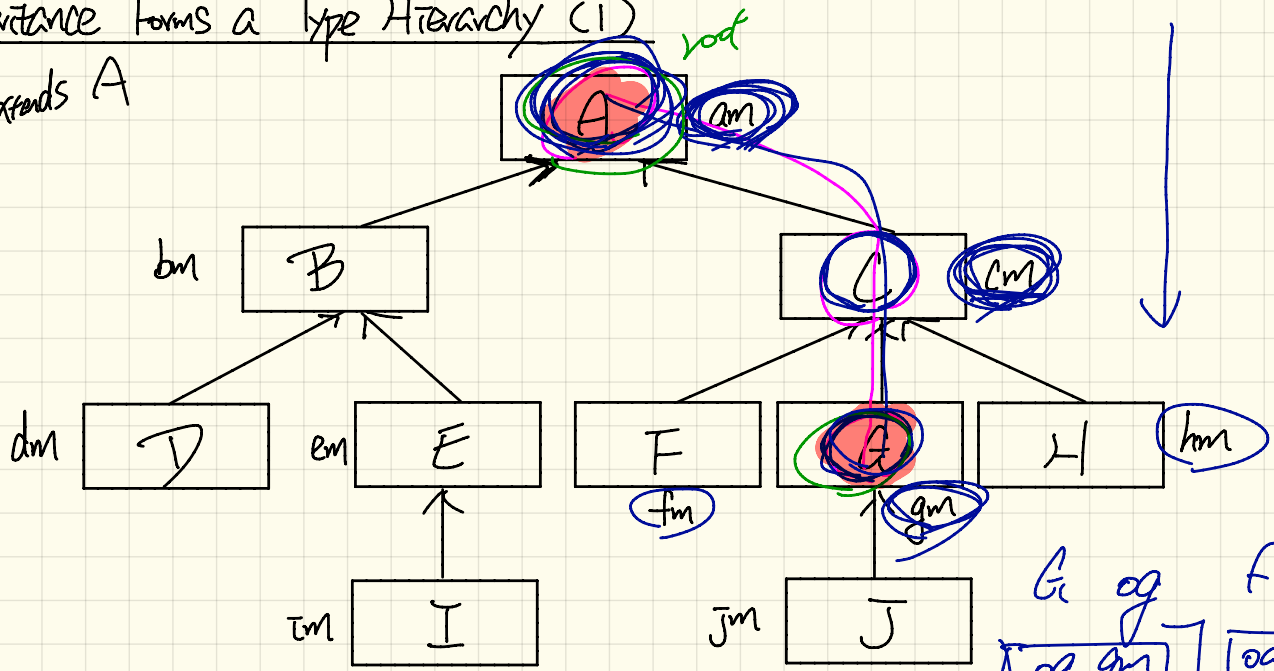



```
1  Course eecs2030 = new Course("EECS2030", 100.0);
2  Student s;
3  ResidentStudent rs = new ResidentStudent("Rachael");
4  NonResidentStudent nrs = new NonResidentStudent("Nancy");
5  rs.setPremiumRate(1.25); rs.register(eecs2030);
6  nrs.setDiscountRate(0.75); nrs.register(eecs2030);
7  s = rs; System.out.println(s.getTuition());/* output: 125.0 */
8  s = nrs; System.out.println(s.getTuition());/* output: 75.0 */
```



Inheritance Forms a Type Hierarchy (1)

B extends A



	ancestors	expectations	descendants
A	A	Gm	all classes
C	C, A	Cm, am	C, F, G, H, J
G	G, C, A	gm, cm, am	J, G

G og

og.gm

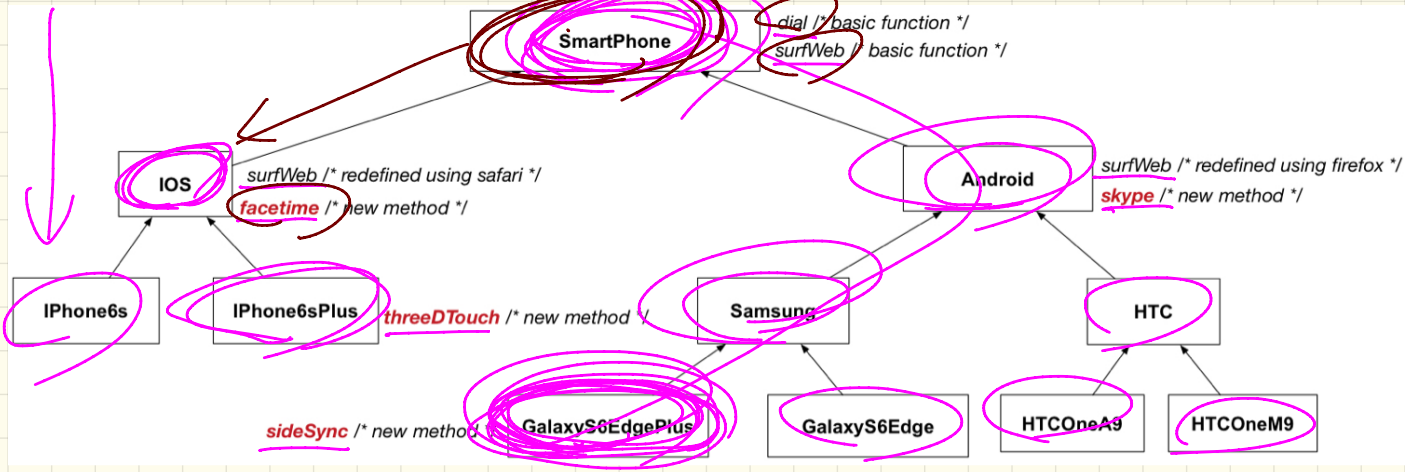
og.cm

og.am

A oa

oa.am

Inheritance Forms a Type Hierarchy (2)

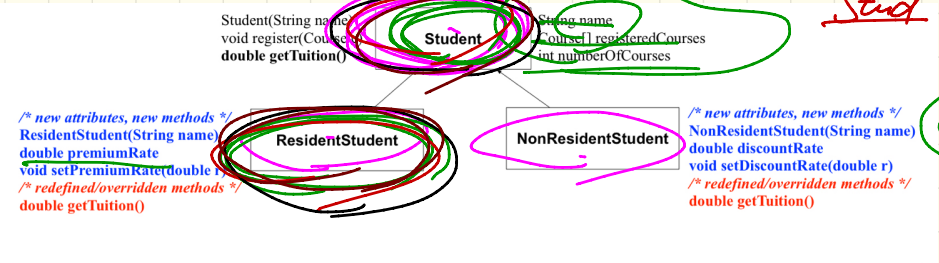


	ancestors	expectations	descendants
SmartPhone	SP		
Android	A, SP		
GS6EP	GS6EP, S, A, SP		

Substitutions ≈ Re-assignments

When considering compilation, only look at static types.

Rule: the ST of RHS is a descendant class of the ST of LHS.




```
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.			✓			×		✓	

Declarations

Student s;

ResidentStudent rs;

NonResidentStudent nrs;

Substitutions

ST: Student

ST: RS

s = rs;

valid

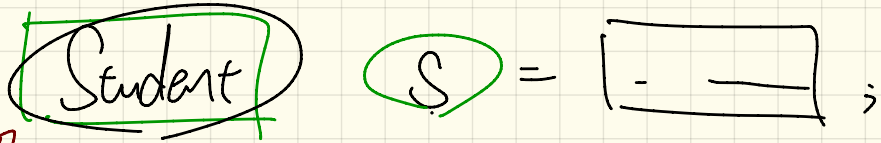
s = nrs;

s

rs = s;

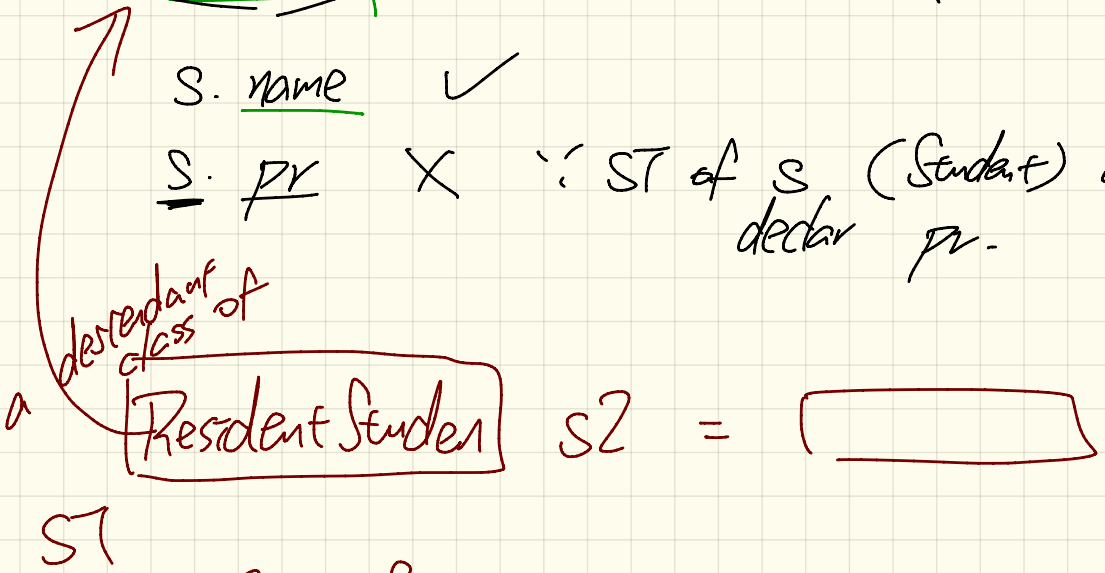
invalid

rs = nrs;



S. name ✓

S. pr ✗ ∵ ST of S (Student) doesn't declar pr.



Student S = new ResidentStudent(--);

→ SI: Student DT: RS

S = new NonResidentStudent(--);

→

SI: Student DT: NRS

S → RS

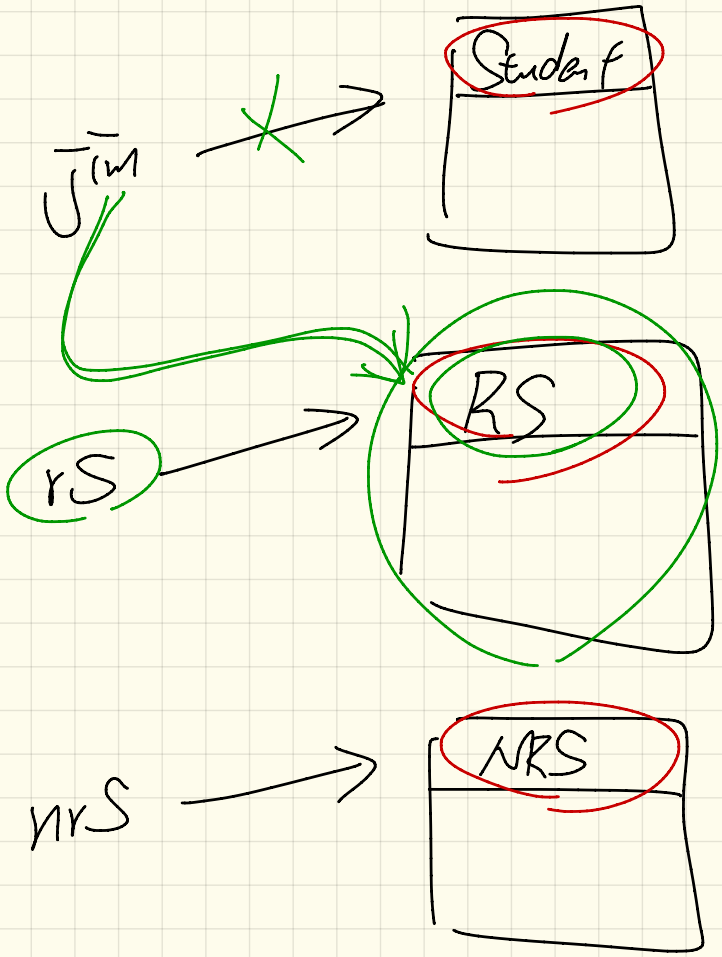
S

ResidentStudent s2 = new Student();

NRS

s2.pr

DT:



ST(Student) ST(RS)

jim = rs ✓

ST of jim : Student

DT of jim : RS