

LECTURE 17

WEDNESDAY NOVEMBER 6

Review: Student Classes (with inheritance)

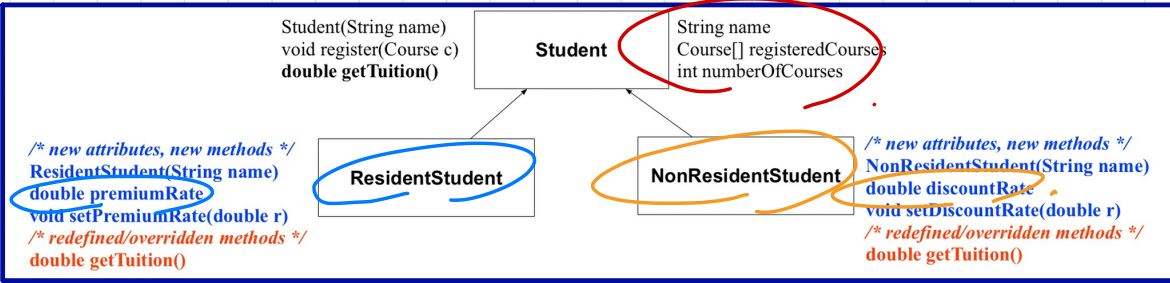
```
class Student {
    String name;
    Course[] registeredCourses;
    int numberOfCourses;
    Student (String name) {
        this.name = name;
        registeredCourses = new Course[10];
    }
    void register(Course c) {
        registeredCourses[numberOfCourses] = c;
        numberOfCourses ++;
    }
    double getTuition() {
        double tuition = 0;
        for(int i = 0; i < numberOfCourses; i++) {
            tuition += registeredCourses[i].fee;
        }
        return tuition; /* base amount only */
    }
}
```

base version inherited
function overriding
new attribute/method

```
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

```

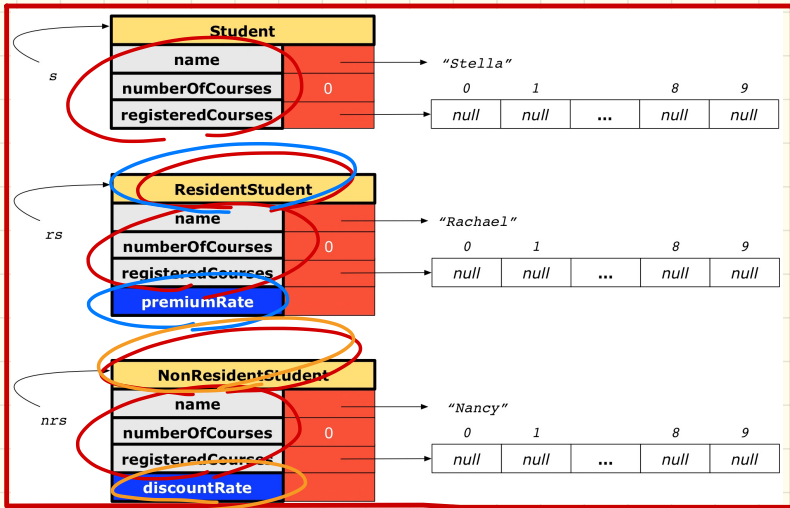
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
  
```

Declaring Static Types

Static types

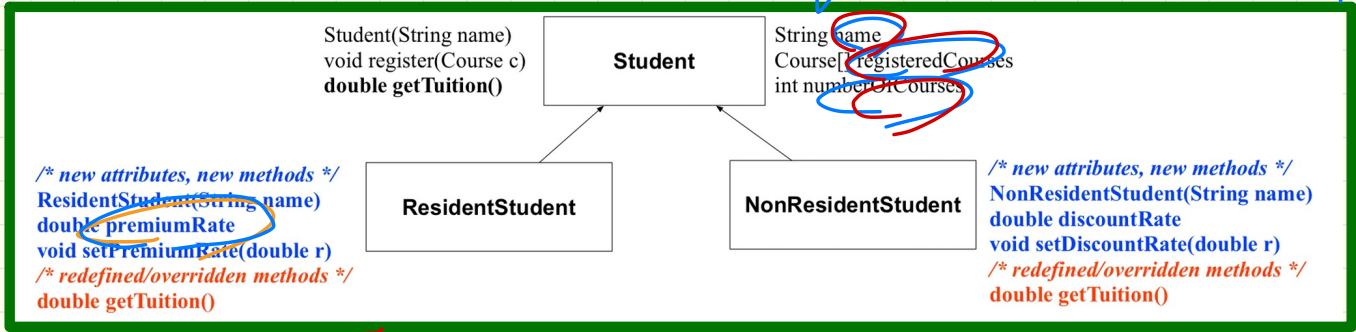
dynamic types

Runtime Object Structure



Intuition: Polymorphism

assignments in context of inheritance



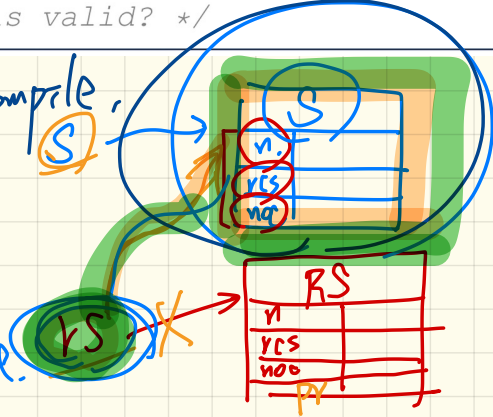
```

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? */
    
```

Assume LS should compile.

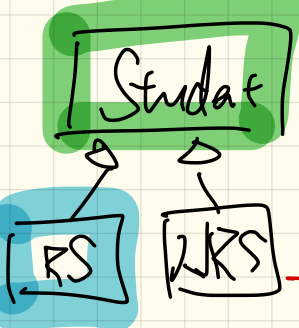
Expect $rs = s$;

Crash! LS should not compile.



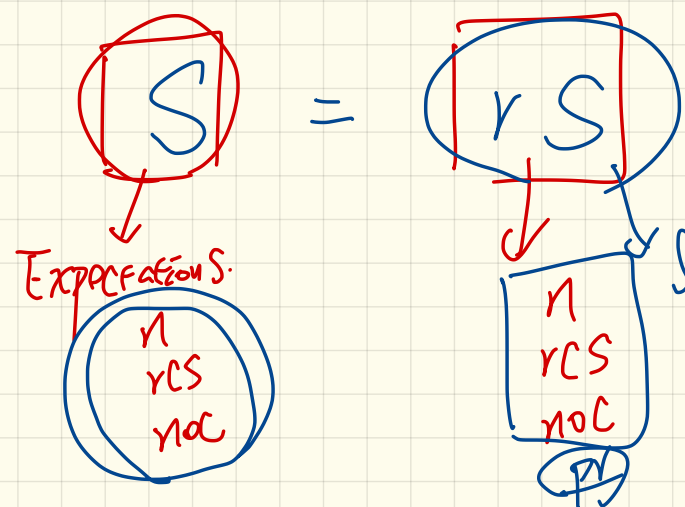
Expectations on rs.

$rs.n$
 $rs.rcs$
 $rs.noc$
 $rs.pr$



Student $S = \dots$

Resident Student $rs = \dots$

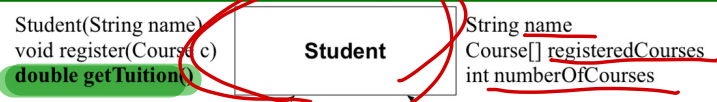


valid
 Feature Type of rs
 (RS)
 a child class
 of ST of S
 (Student)

Intuition: Dynamic Binding

runtime behaviour depends on

dynamic type.



```

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

```

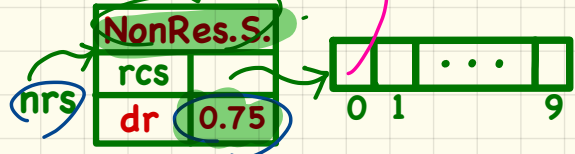
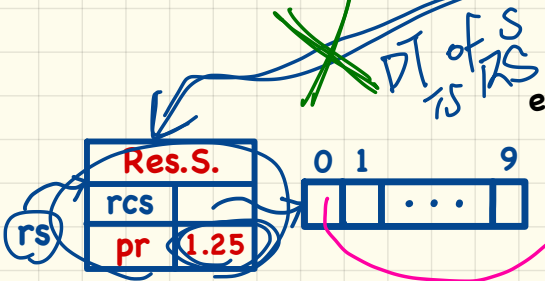
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()
  
```

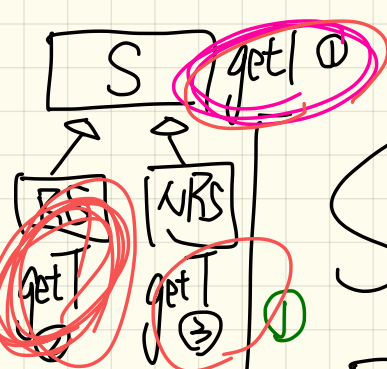
```

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 */
  
```

S (Pr) X
ST: Student

DT of S to NRS





```

Student S = new Student();
S.getTuition();

```

```

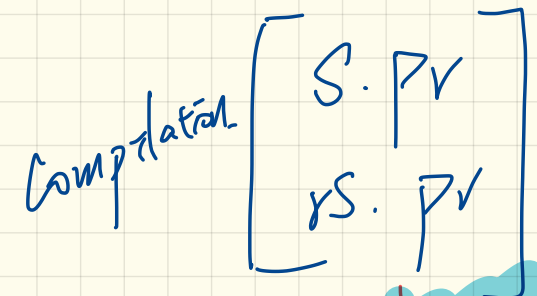
RS rs = new RS();

```

```

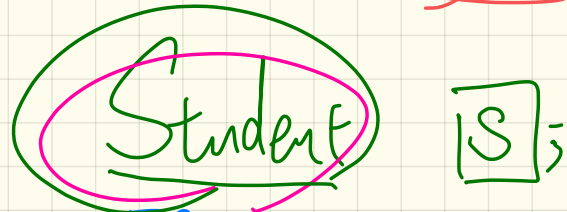
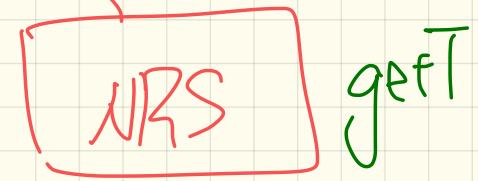
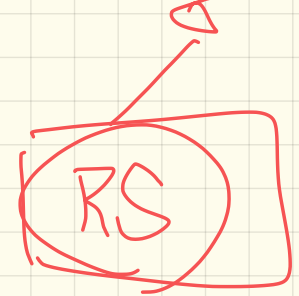
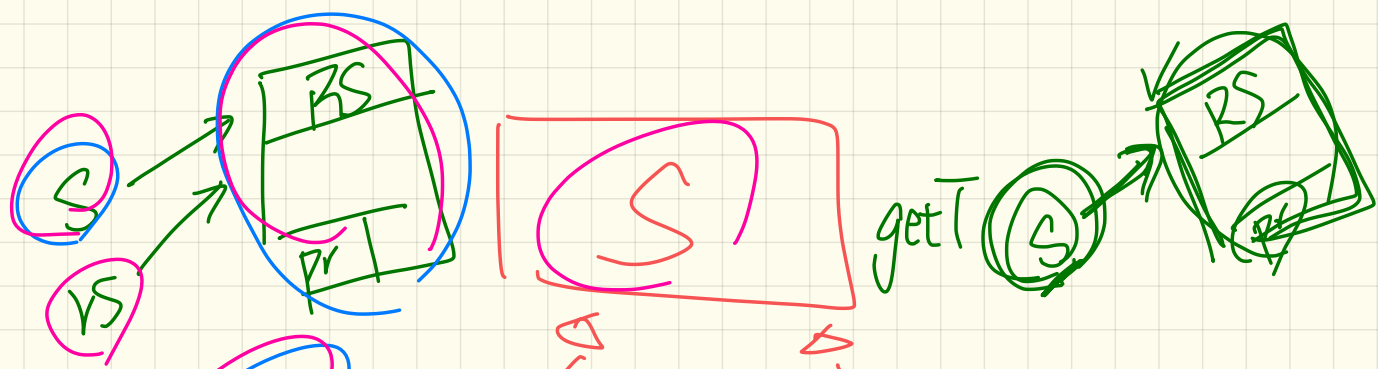
rs.setPr(1.5);
S = rs;

```



- ② $S = rs$
- ③.1 $S.Pr \times$ ∵ ST of S doesn't support
- ③.2 $S.getTuition();$

⑥ Which version of getT is called?
 ∵ DT of S is RS ∴ call RS version.
 It Computer ∵ ST of S (find) supports that



RS

rs = new RS (..);

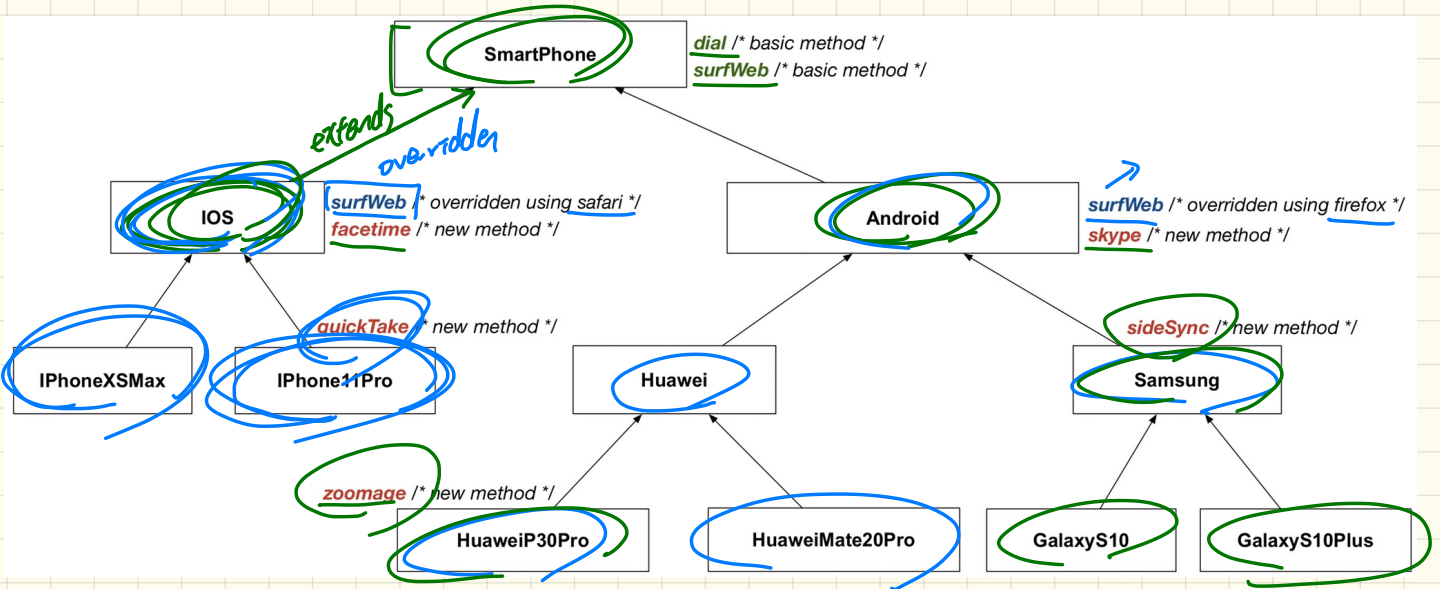
[rs]. setPr (..);

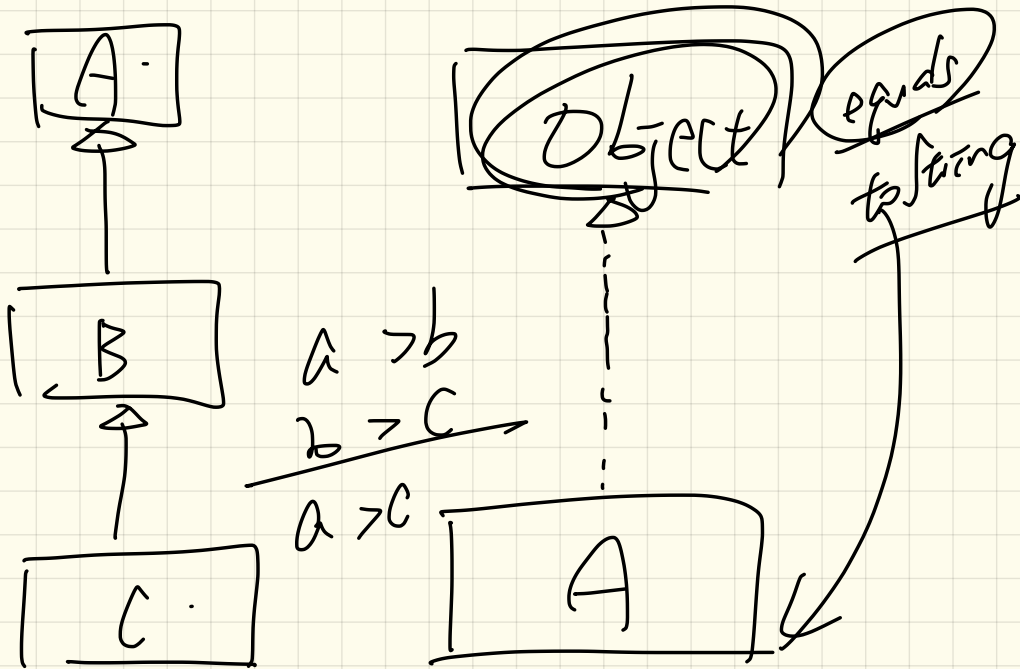
S = rs;

[S]

setPr (1.5);
rs.setPr (2.7); ? X

Multi-Level Inheritance Hierarchy of Smartphones

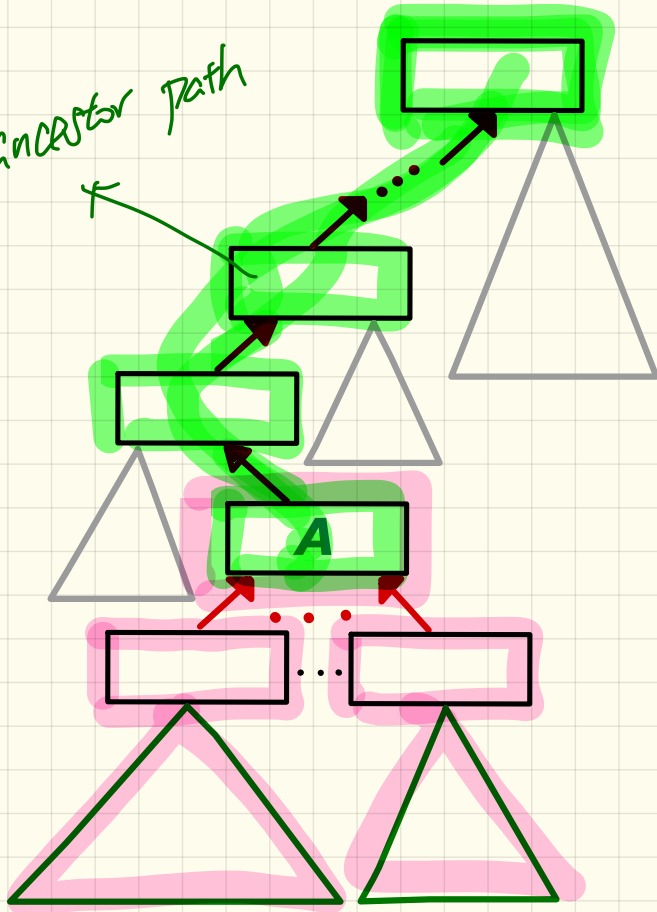




Inheritance Forms a Type Hierarchy

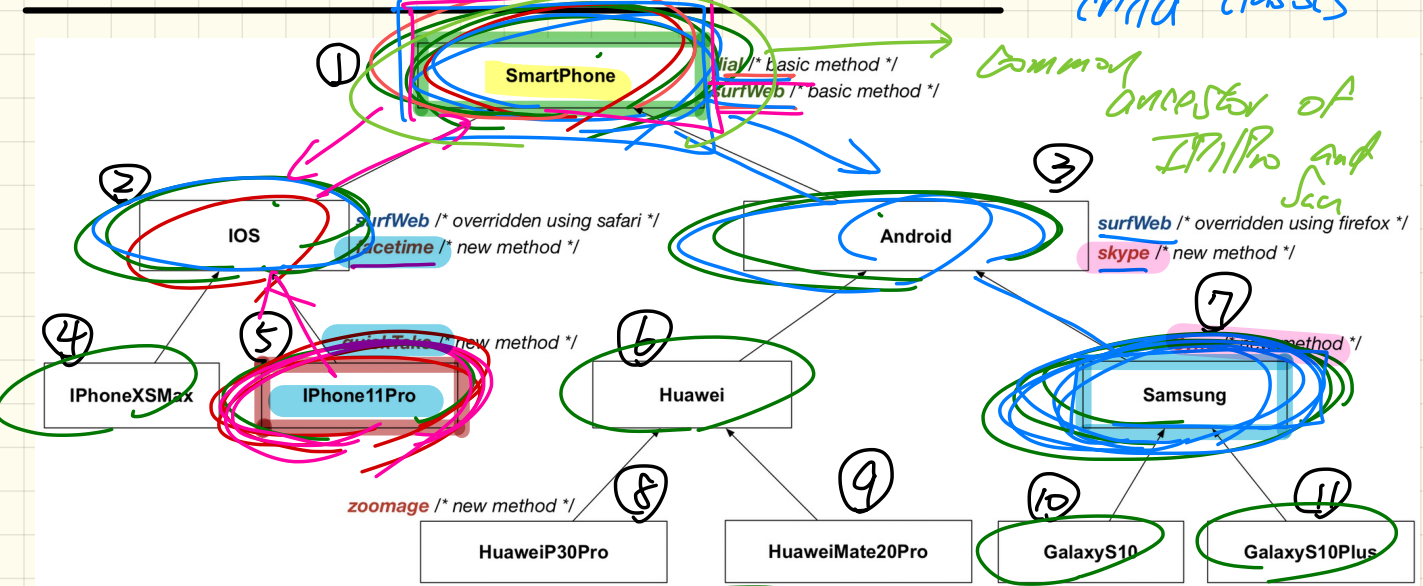
ancestors of A

dependants of A
ancestors path



Inheritance Accumulates Code for Reuse

child classes



Common ancestor of IP11Pro and S10

	ancestors	expectations	descendants
	SmartPhone	dial, surfWeb	① ~ ⑪.
	Samsung, Android	dial, surfWeb, skype, zoomage	⑦, ⑩, ⑪
	IP11Pro, IOS, iPhoneXSMax	dial, surfWeb, facetime, zoomage	⑤

Static Types determine Expectations

Inheritance Hierarchy: Students

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



```
String name
Course[] registeredCourses
int numberOfCourses
```

```
Declare:
Student jim;
...
jim.??
```

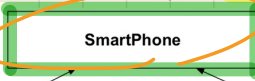
```
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
```



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

Inheritance Hierarchy: Smart Phones

(Android) pl = ...
 pl.sideSync X
 pl.quickTake



```
dial /* basic method */
surfWeb /* basic method */
```

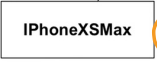
```
Declare:
SmartPhone myPhone;
...
myPhone.??
```



```
surfWeb /* overridden using safari */
facetime /* new method */
```



```
surfWeb /* overridden using firefox */
skype /* new method */
```

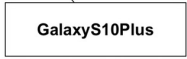


```
quickTake /* new method */
```



```
sideSync /* new method */
```

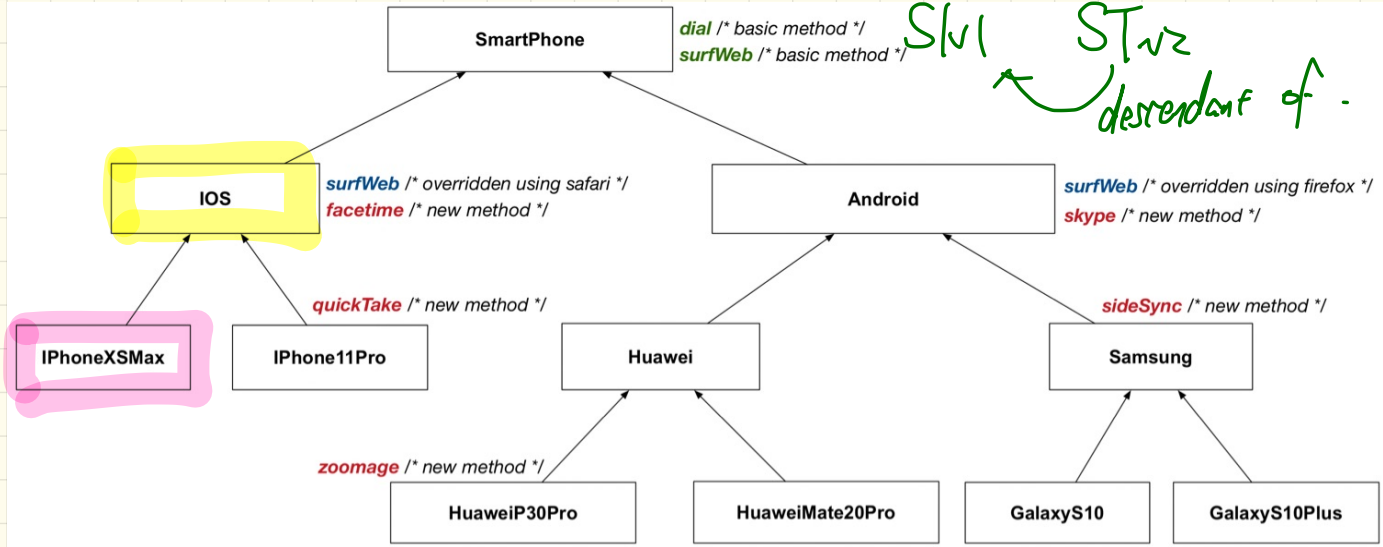
```
zoomage /* new method */
```



Rules of Substitutions (1)

$$\frac{v1}{Sv1} = \frac{v2}{Sv2}$$

STv1 STv2
↙ ↘
dependant of .

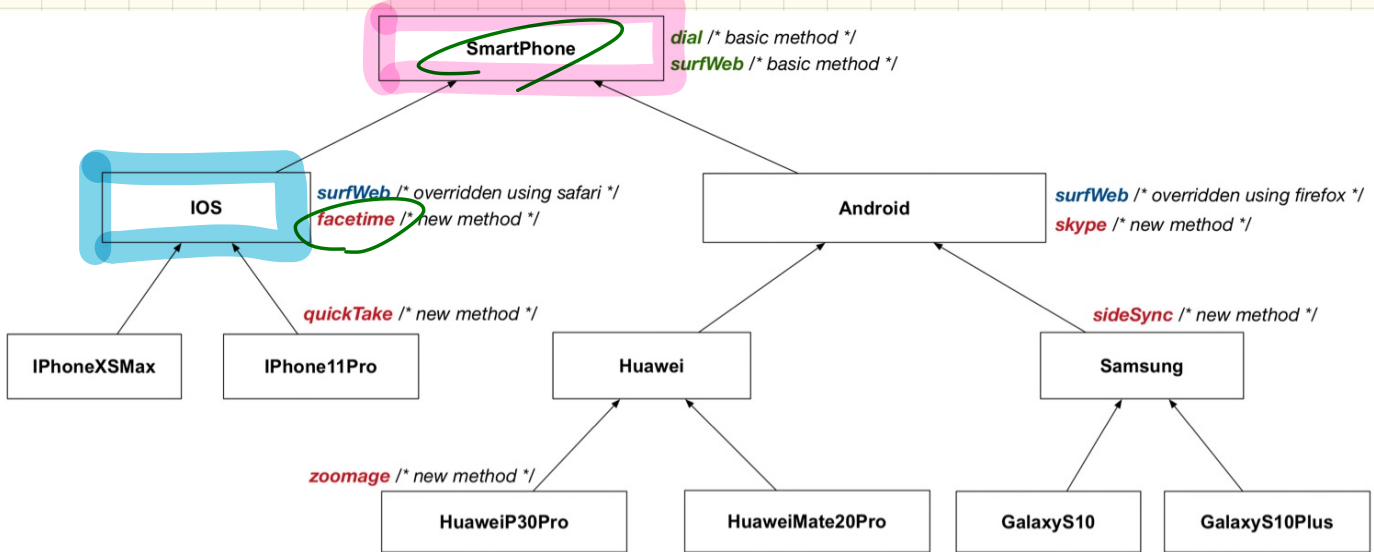


Declarations:
IOS sp1;
iPhoneXSMaX sp2;
iPhonePro11 sp3;

Substitutions:
sp1 = sp2;
sp1 = sp3;

can the ST of sp2 full fill expectations of ST of sp1

Rules of Substitutions (2)



Declarations:

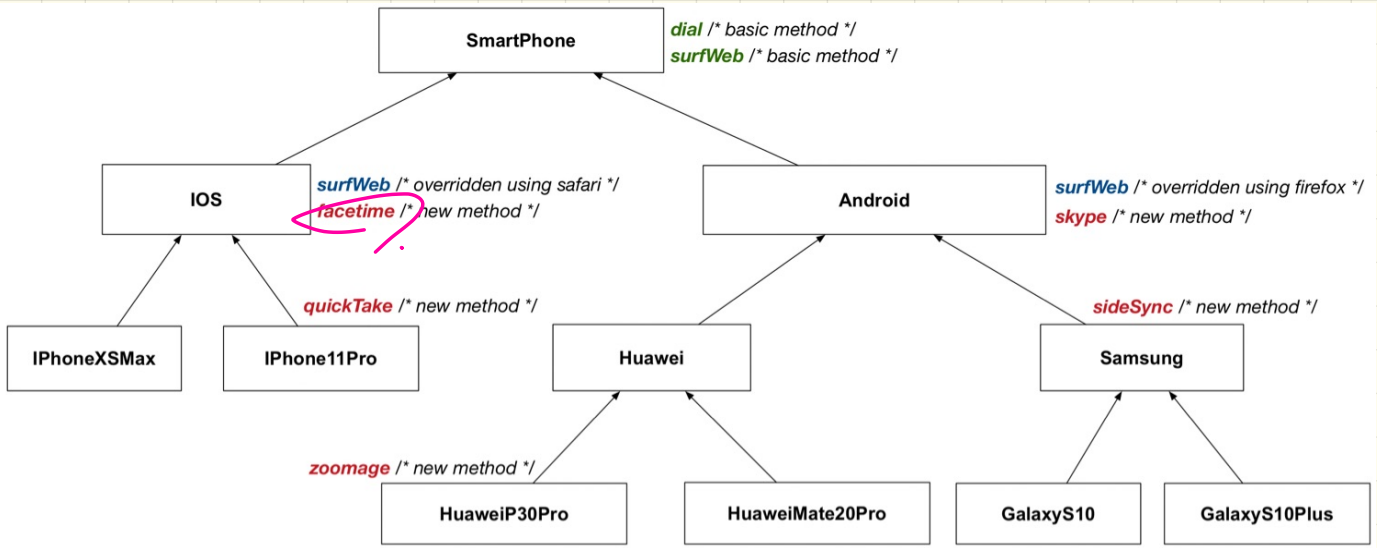
IOS sp1;
SmartPhone sp2;

Substitutions:

sp1 = sp2;

ST: IOS ST: SP

Rules of Substitutions (3)



Declarations:

IOS sp1;
HuaweiP30Pro sp2;

Substitutions:

sp1 = sp2;

p.g. facetime cannot be full filled by

sp: IOS

sp: HuaweiP30Pro

Visualization: Static Type vs. Dynamic Type

Declaration:

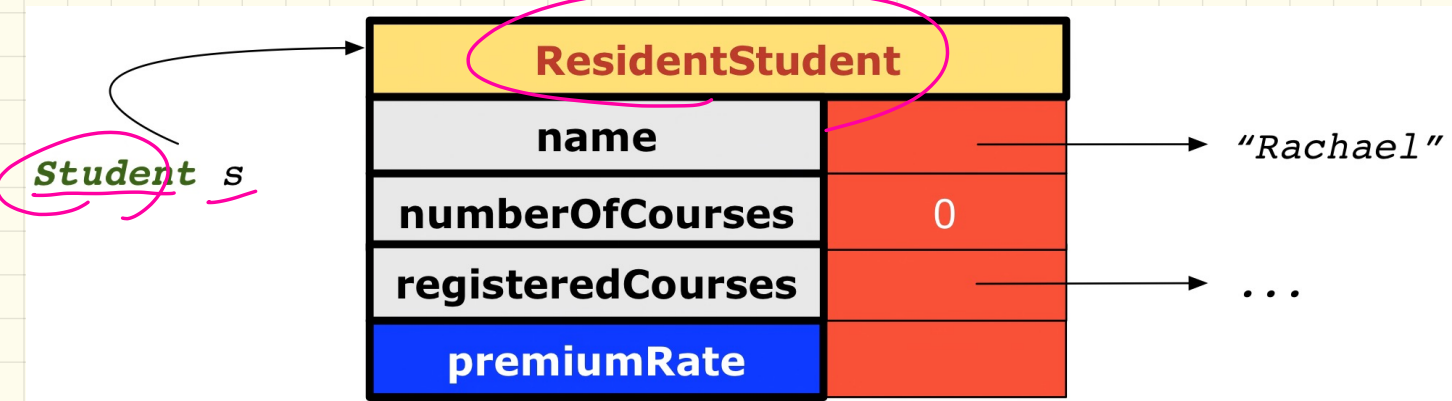
Student s;

ST

DT

Substitution:

s = **new ResidentStudent**("Rachael");



Change of **Dynamic** Type (1.1)

```
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

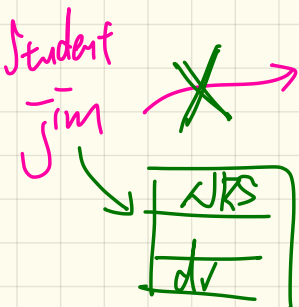
```
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()
```

ST

Example 1:

```
Student jim = new ResidentStudent(...);
jim = new NonResidentStudent(...);
```

DT of jim: RS
NRS



can RS (the attempted DT) fulfill the expectation (defined by ST Student) of jim.

Change of **Dynamic** Type (1.2)

```
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()
```

Example 2:

```
ResidentStudent jeremy = new Student(...);
```

∴ Students cannot fulfill expectations of **RS**