Lecture 18 - Nov. 12

Inheritance

Dynamic Binding: Intuition
Ancestors vs Descendants, Expectation
Variable Substitutions

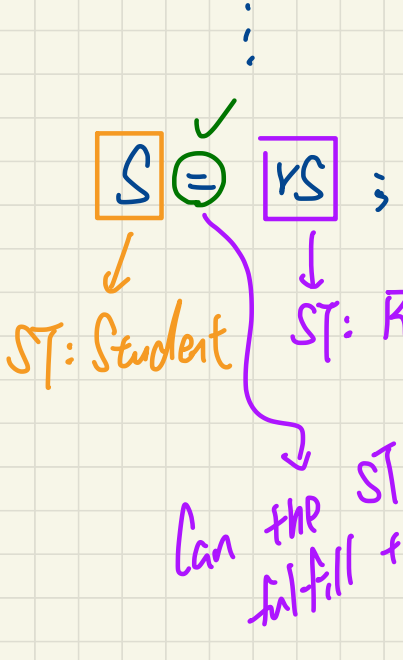
Announcements/Reminders

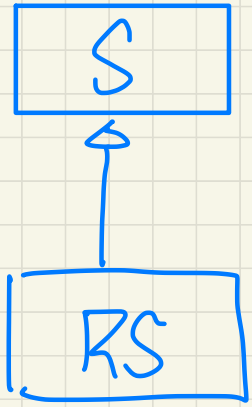
- **WrittenTest2** tomorrow
- **Lab4** due this Friday at noon
- **ProgTest3** next Wednesday, November 20
- **ProgTest2** results & feedback released

Student S = ...

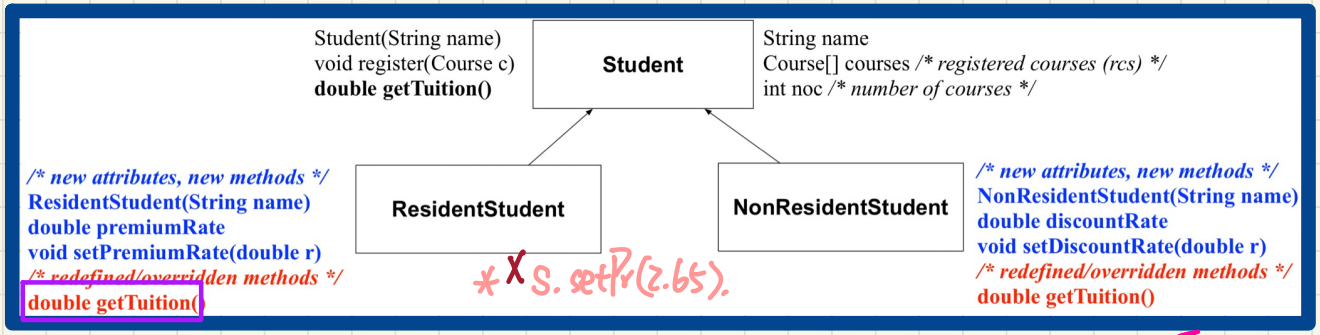
ResidentStudent RS = ...

RS = S; ✗



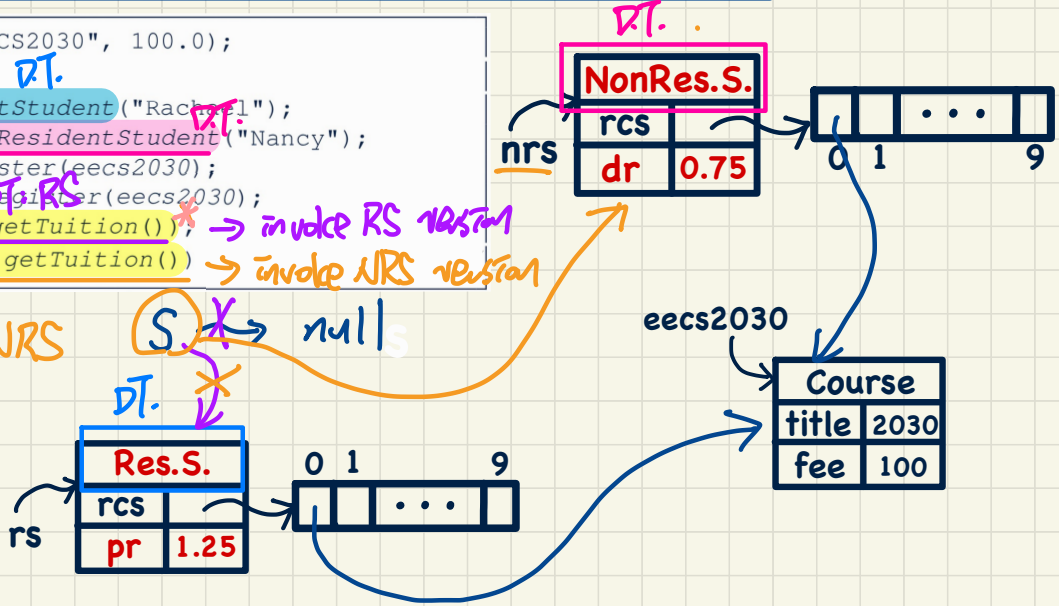


Intuition: Dynamic Binding




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

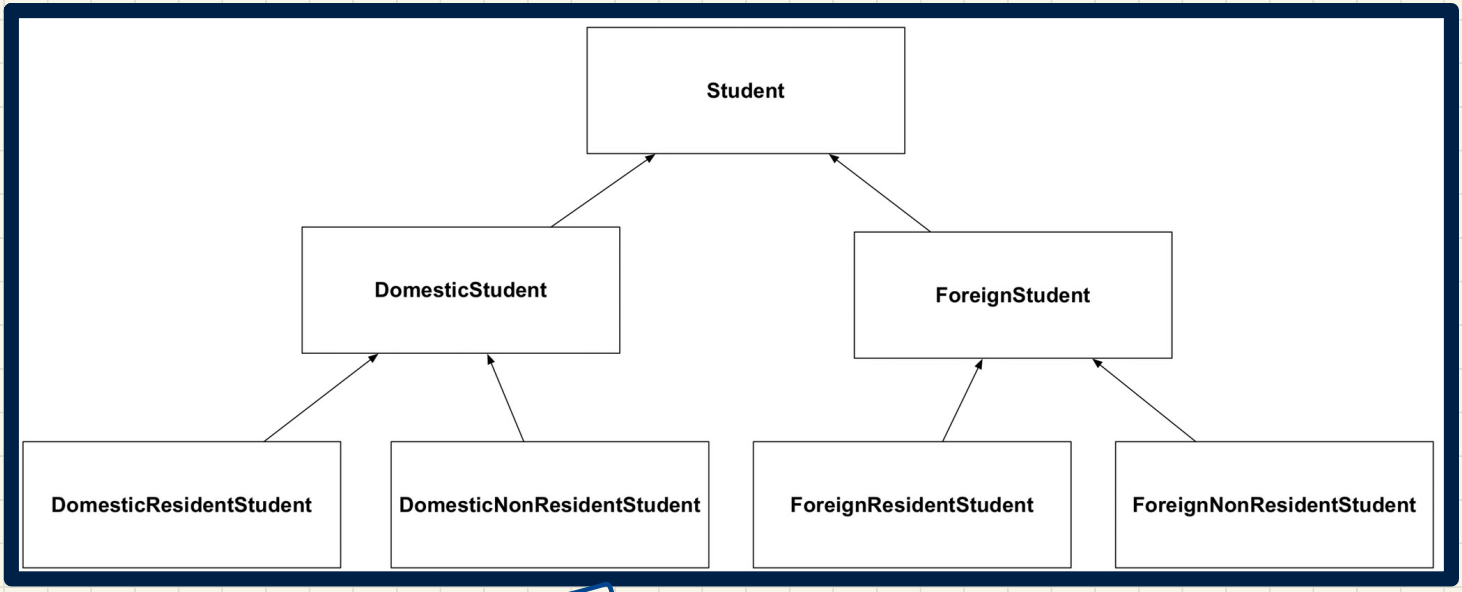
	ST of s	DT of s
①	Student	null
②		RS
③		RS
④		NRS



Compilation : Static types

runtime : dynamic types

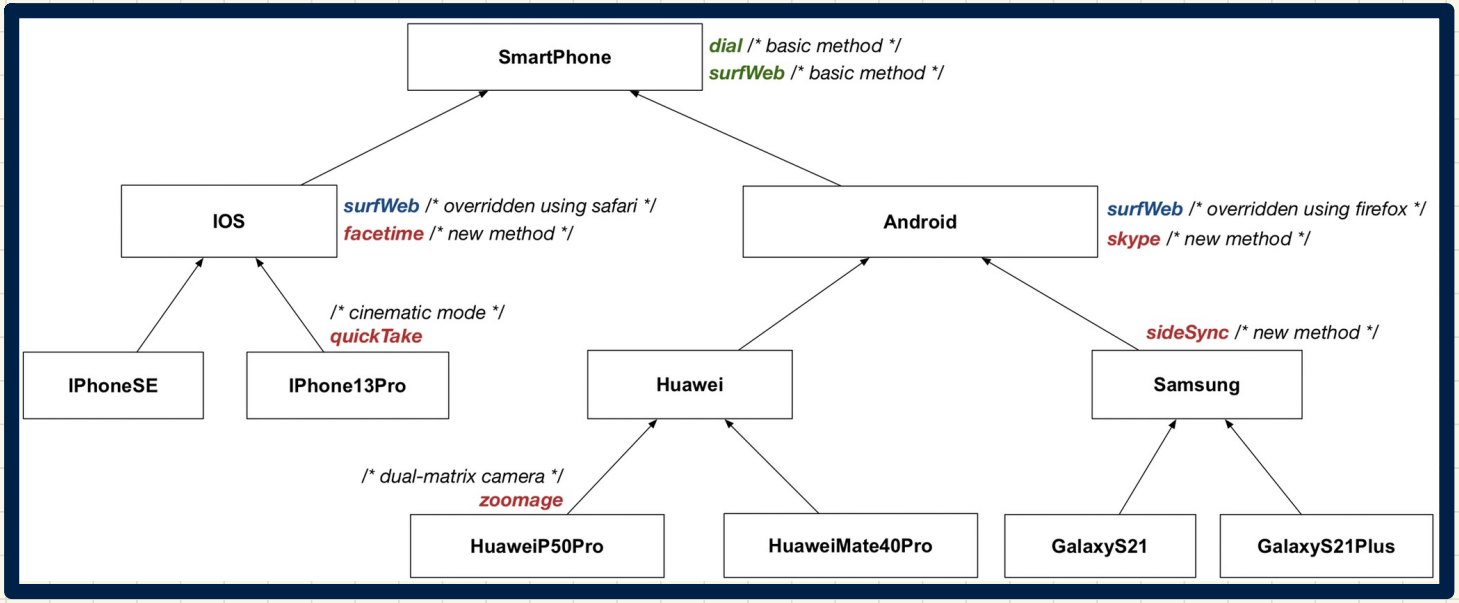
Multi-Level Inheritance Hierarchy: Students



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?

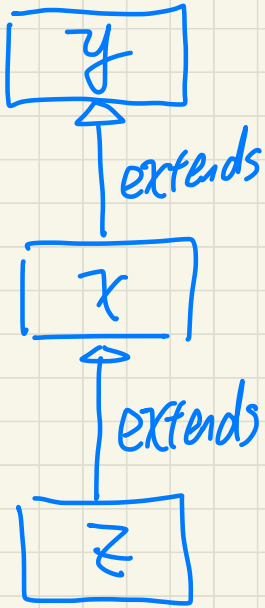
Multi-Level Inheritance Hierarchy: Smartphones



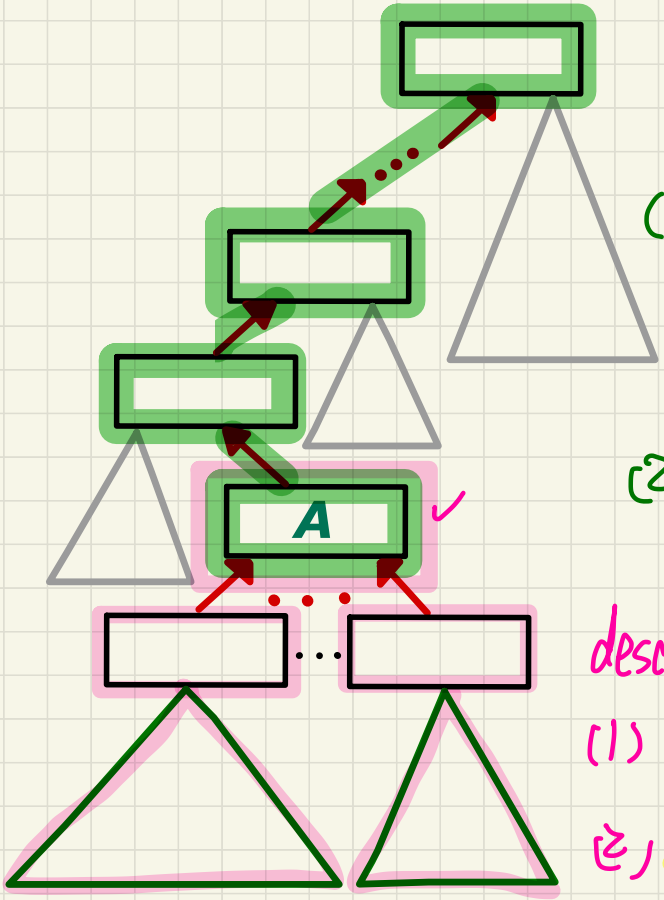
Reflections:

- 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?

Inheritance Forms a Type Hierarchy



Z indirectly extends X



ancestors of A:

(1) A accumulates the attrs/methods from its ancestors.

(2) exp(A) ≥ exp of any ancestor

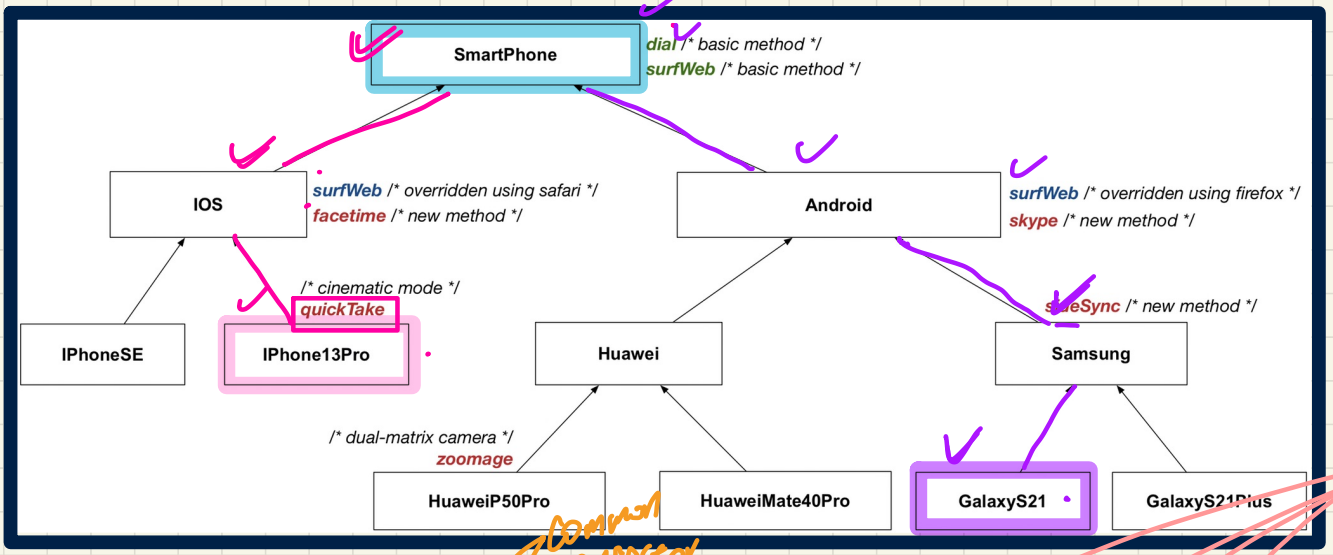
descendants of A:

(1) exp(A) is inherited to all descendants

(2) exp(A) ≤ exp of any descendant

Inheritance Accumulates Code for Reuse

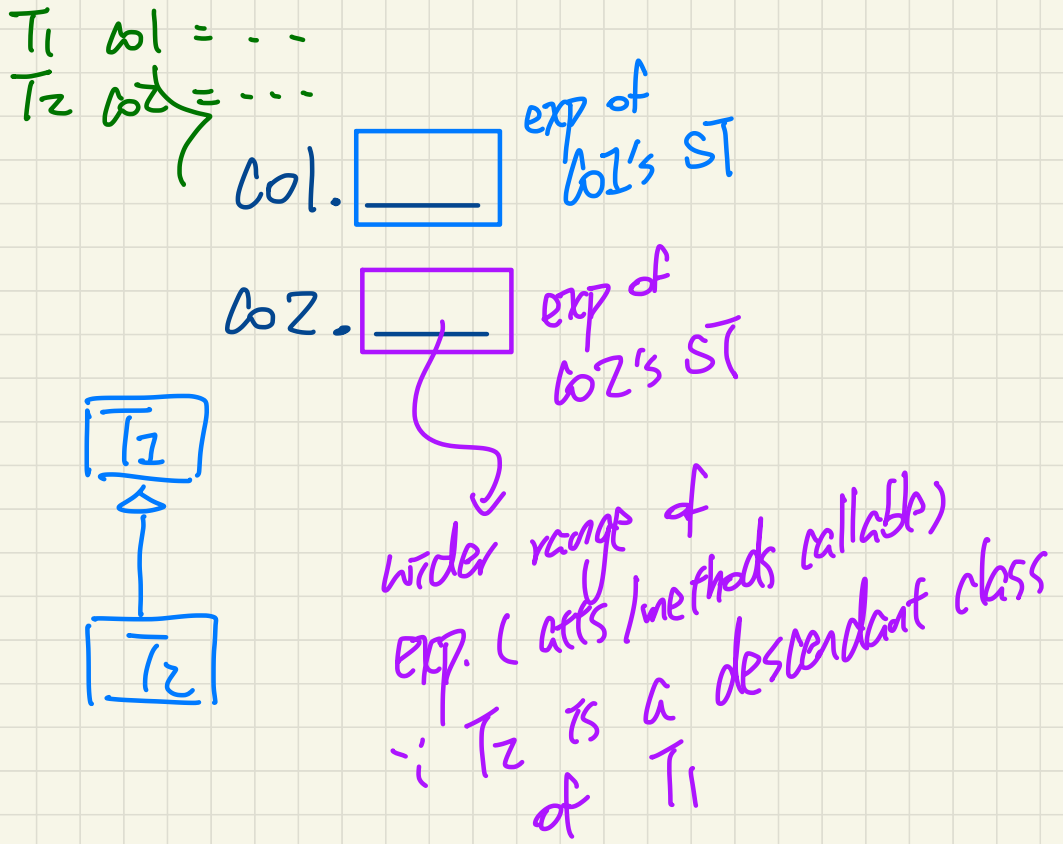
Common Ancestor



exp of common ancestor.

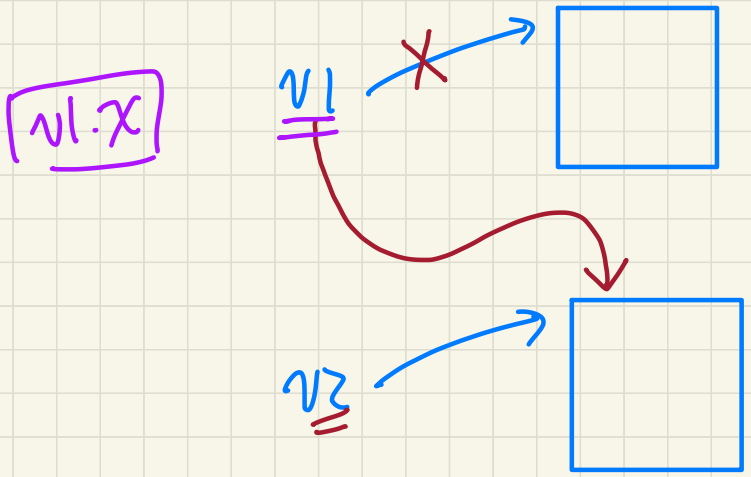
common ancestor

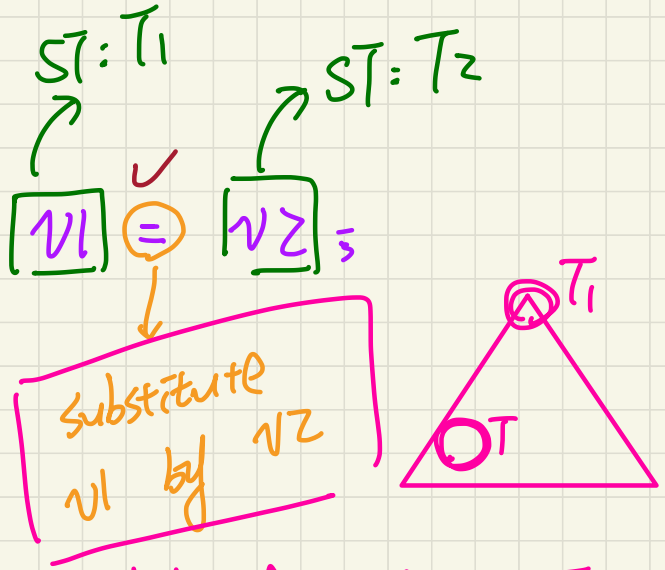
	ancestors	expectations	descendants
	GS21, SS, An., SP	sideSync, surfWeb, skype, dial	GS21
	IP13Pro, IOS, SP	quickTake, surfWeb, facetime, dial	IP13Pro
	SP	dial, surfWeb	all classes


T1 o1 = . .
T2 o2 = . . .
o1.
exp of o1's ST
o2.
exp of o2's ST
T1
T2
wider range of exp. (attrs/methods callable)
∵ T2 is a descendant class of T1

T_1 v_1 = ... ;

T_2 v_2 = ... ;



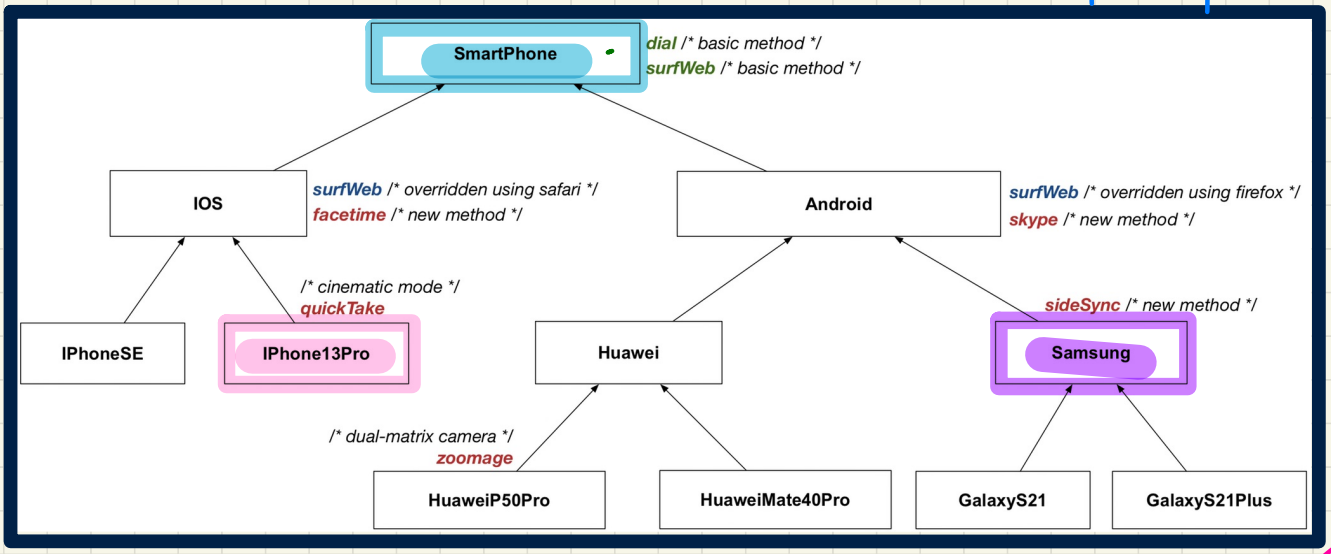


↓ valid if v_2's ST T_2

(1) can fulfill exp of v_1's ST T_1

(2) v_2's ST should be a descendant of v_1's ST

Inheritance Accumulates Code for Reuse



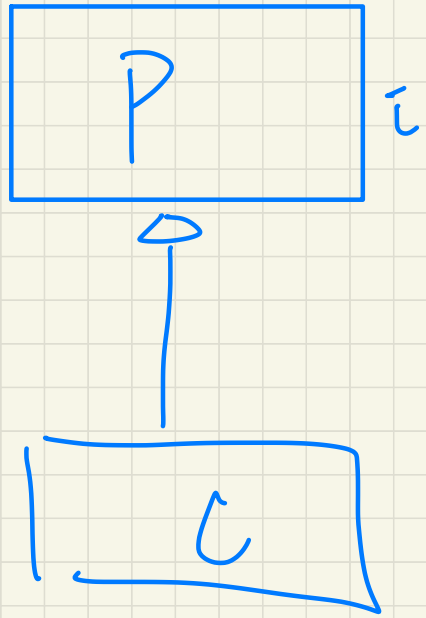
① sp1 = sp2; → compiles ∵ sp2's ST IP13Pro can fulfill the exp of sp1's ST SmartPhone

② sp2 = sp1; ✓

sp2 = sp3; fail to compile ∵ ST of sp1 cannot fulfill the exp of ST of sp2

```
SmartPhone sp1;
IPhone13Pro sp2;
Samsung sp3;
```

```
sp1 = ?;
sp2 = ?;
sp3 = ?;
```



P objP = ...

C objC = ...

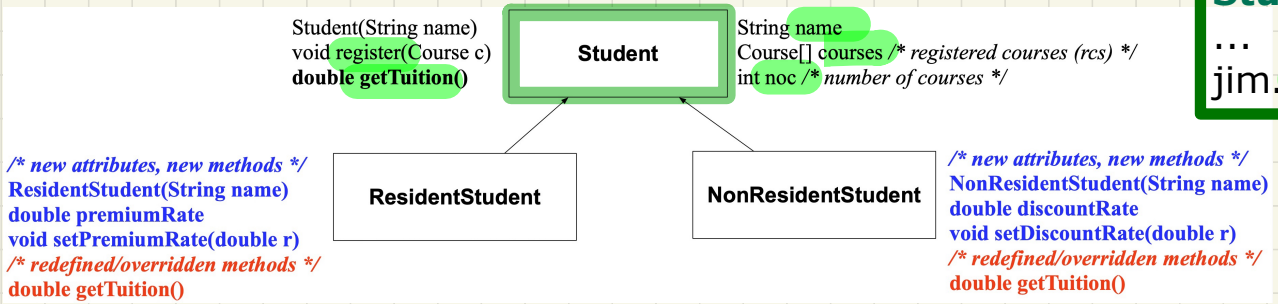
objP = objC ✓

objC = objP ✗

P is not a subclass of C.

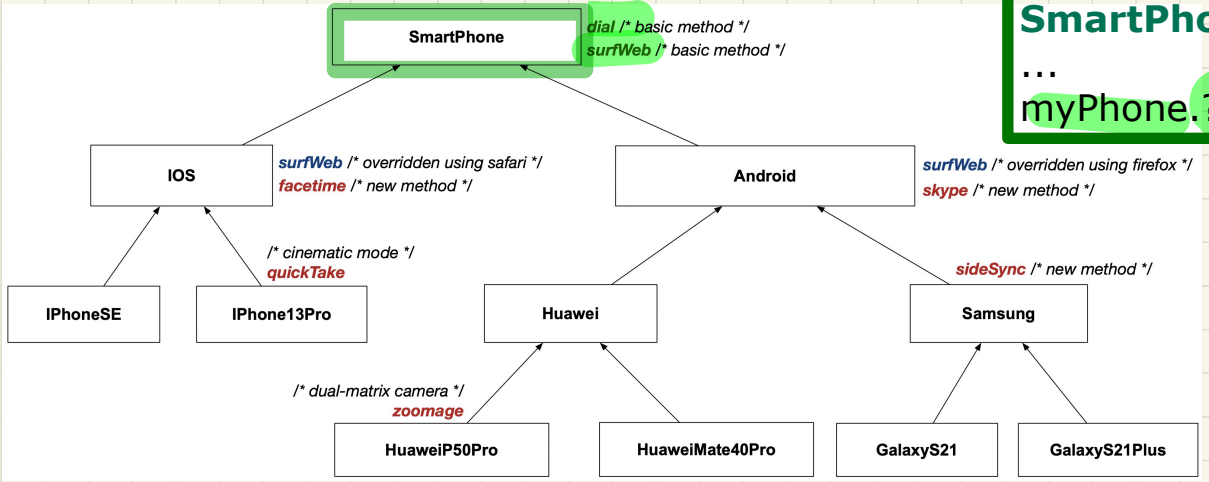
Static Types determine Expectations

Inheritance Herarchy: Students



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

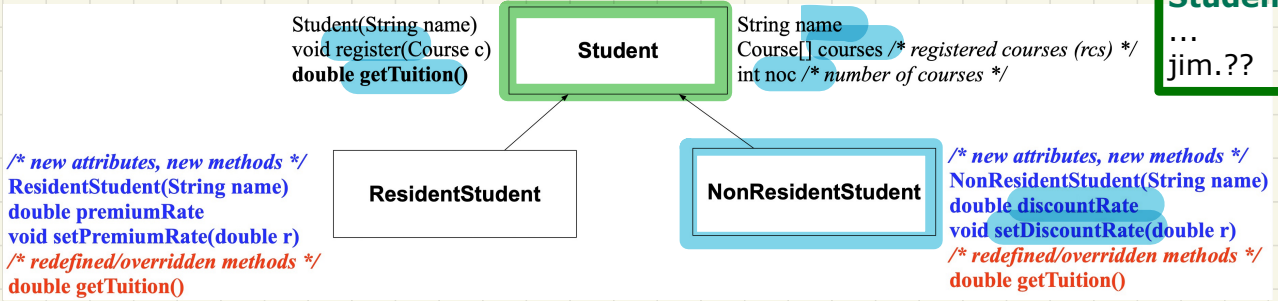
Inheritance Herarchy: Smart Phones



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

Static Types determine Expectations

Inheritance Herarchy: Students

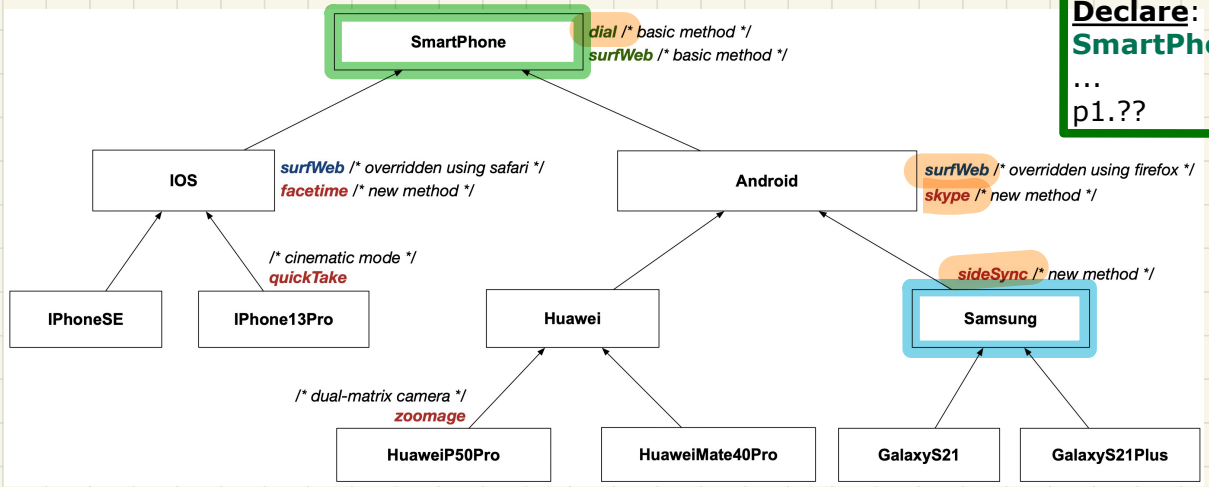


Declare:
Student jim;
...
jim.??

Declare:
NRS alan;
...
alan.??

exp(Student)
+ dR + setDr.

Inheritance Herarchy: Smart Phones



Declare:
SmartPhone p1;
...
p1.??

Declare:
Samsung p2;
...
p2.??