Lecture 18 - Nov 14

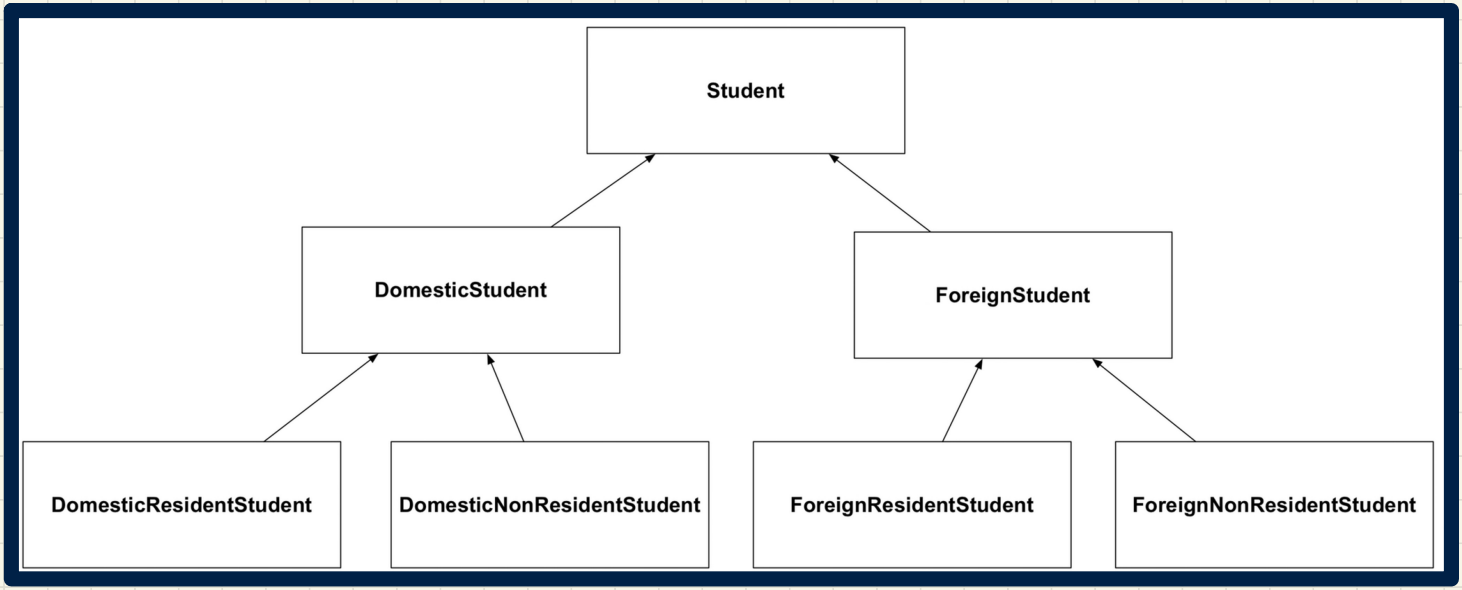
Inheritance

Rules of Substitutions
Static Types vs. Dynamic Types

Announcements

- ProgTest2: this Tuesday, November 15
- Lab4 released
  - ↳ ProgTest3

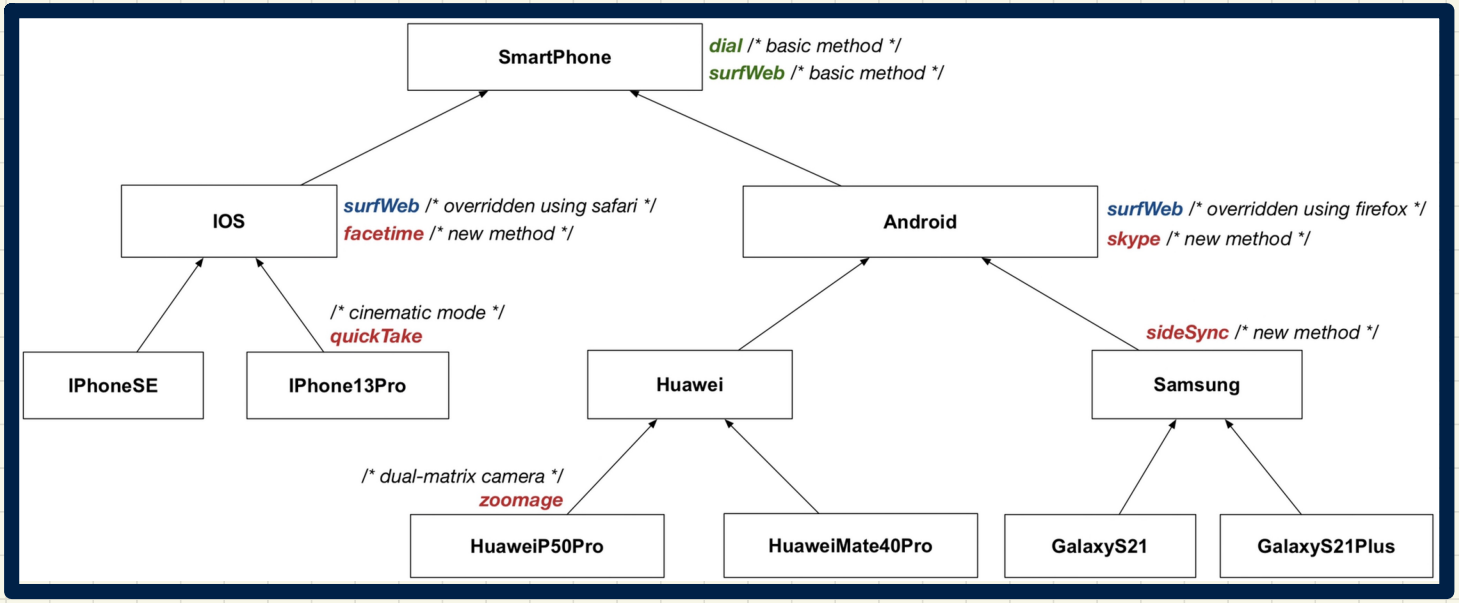
Multi-Level Inheritance Hierarchy: Students



Reflections: kind

- 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?

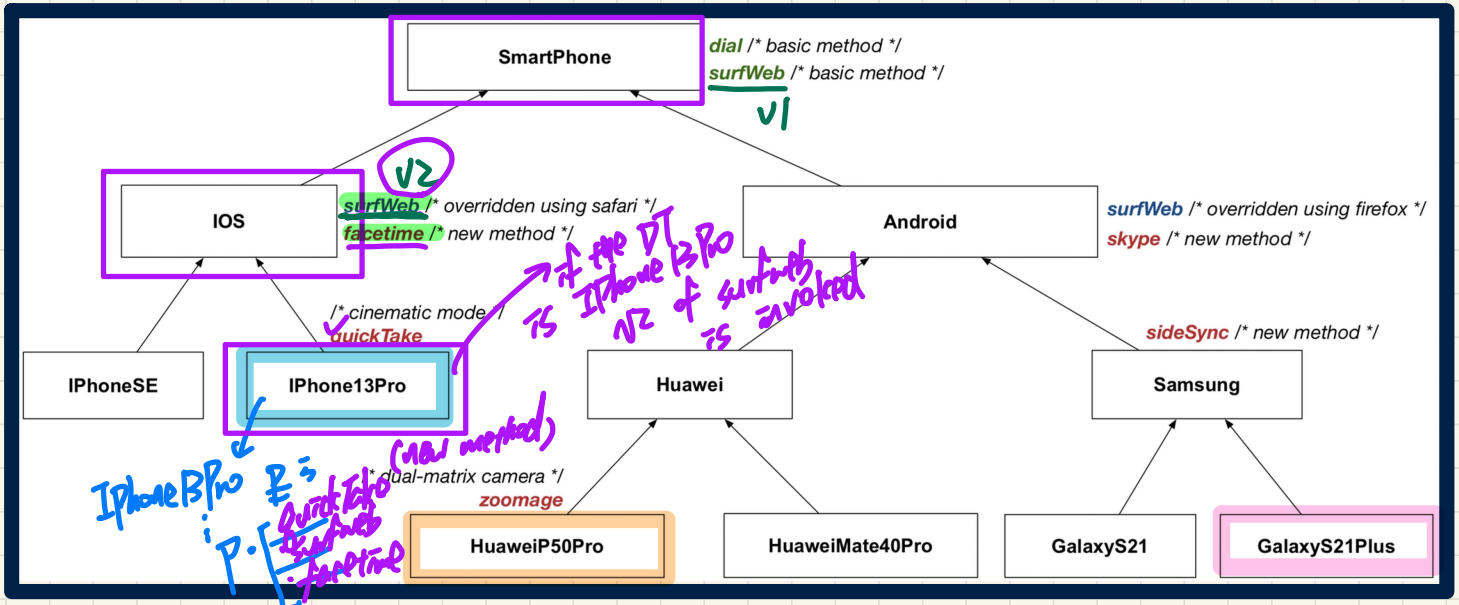
Multi-Level Inheritance Hierarchy: Smartphones



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?

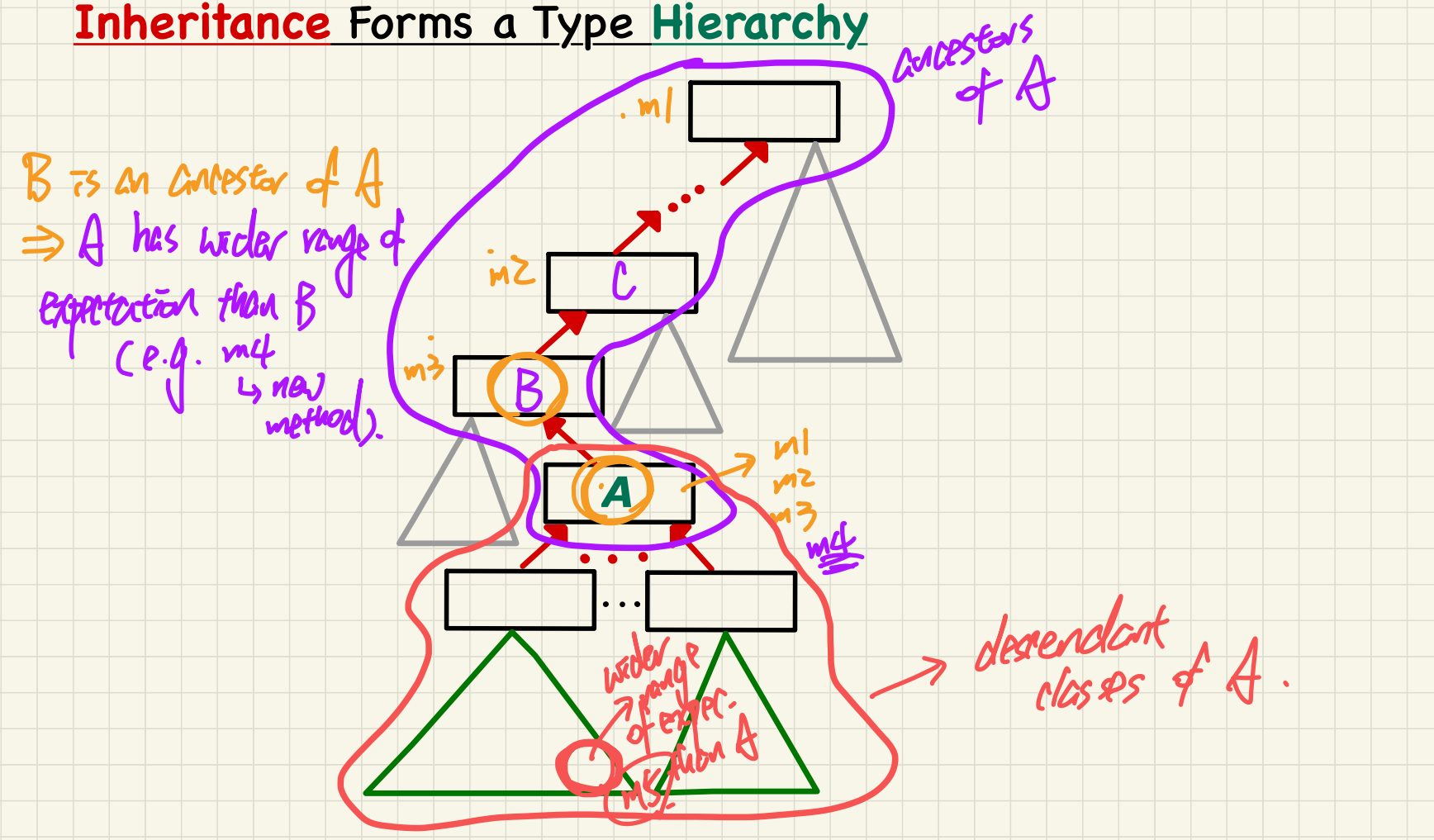
Multi-Level Inheritance Hierarchy: Smartphones



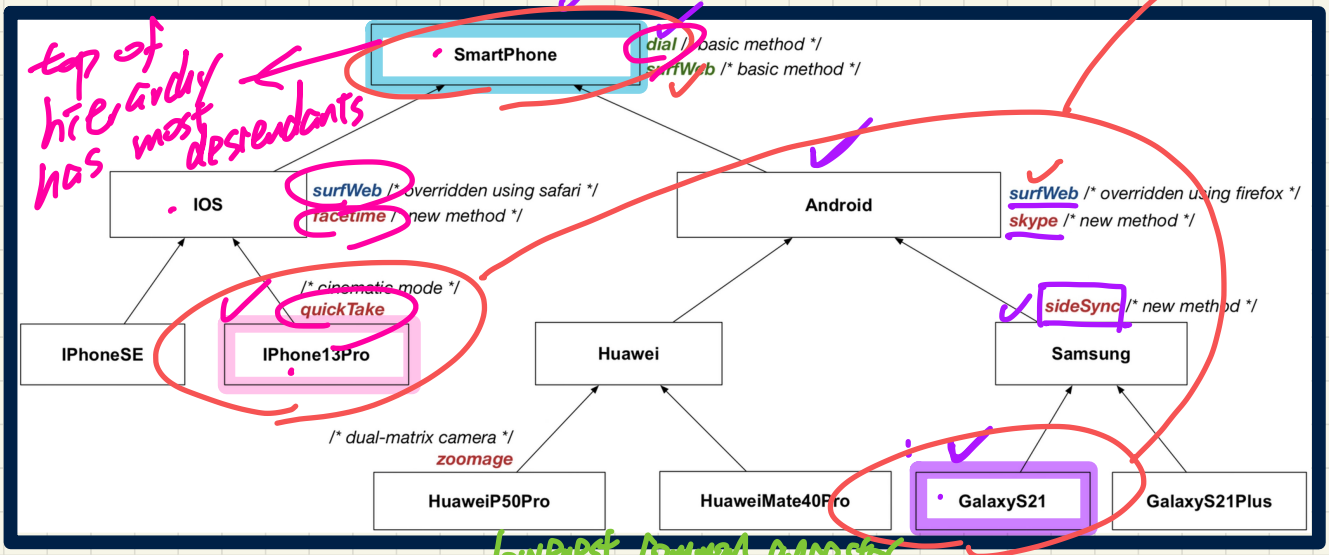
Exercise Compare the ranges of expectations of:

+ IPhone13Pro
+ HuaweiP50Pro
+ GalaxyS21Plus

Inheritance Forms a Type Hierarchy

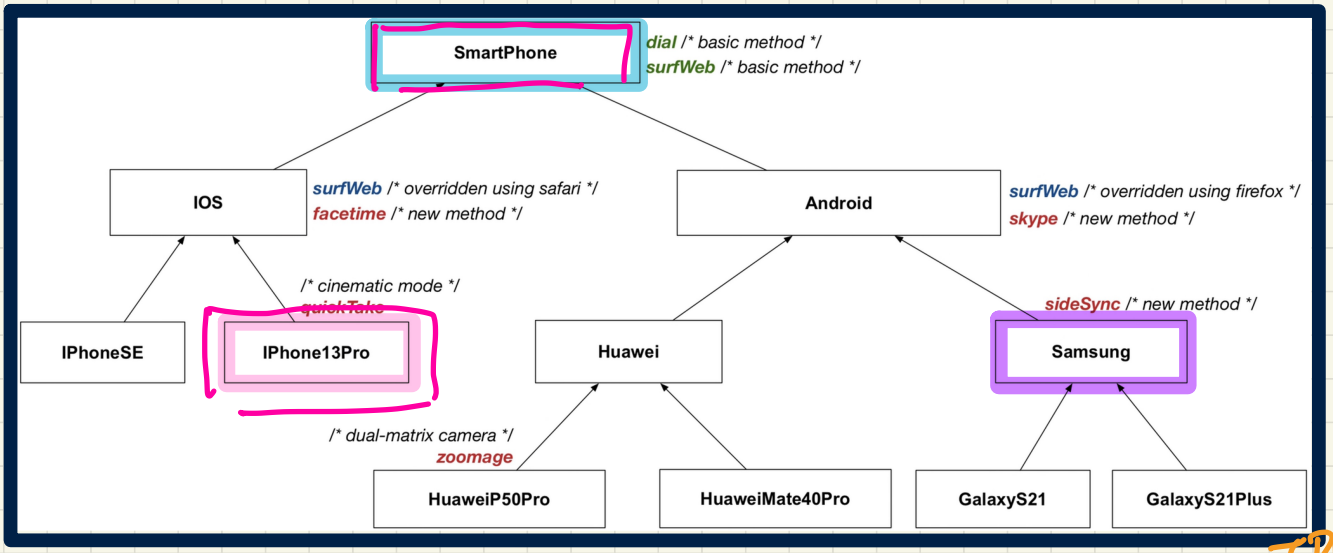


Inheritance Accumulates Code for Reuse



	ancestors	expectations	descendants
	S21, Sam., And., SP	sideSync, skype, surfWeb, dial.	
	IP13Pro, IOS, SP	quickTake, facetime, surfWeb, dial	
	exec, SP.		

Inheritance Accumulates Code for Reuse

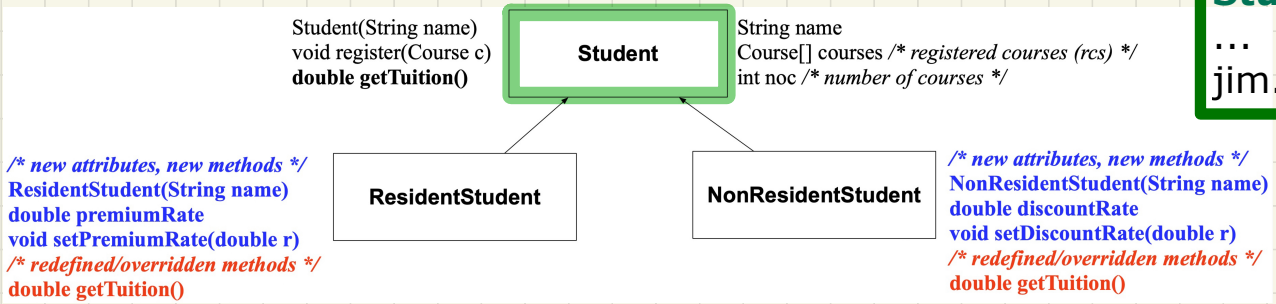



```
SmartPhone sp1;
IPhone13Pro sp2;
Samsung sp3;
```

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

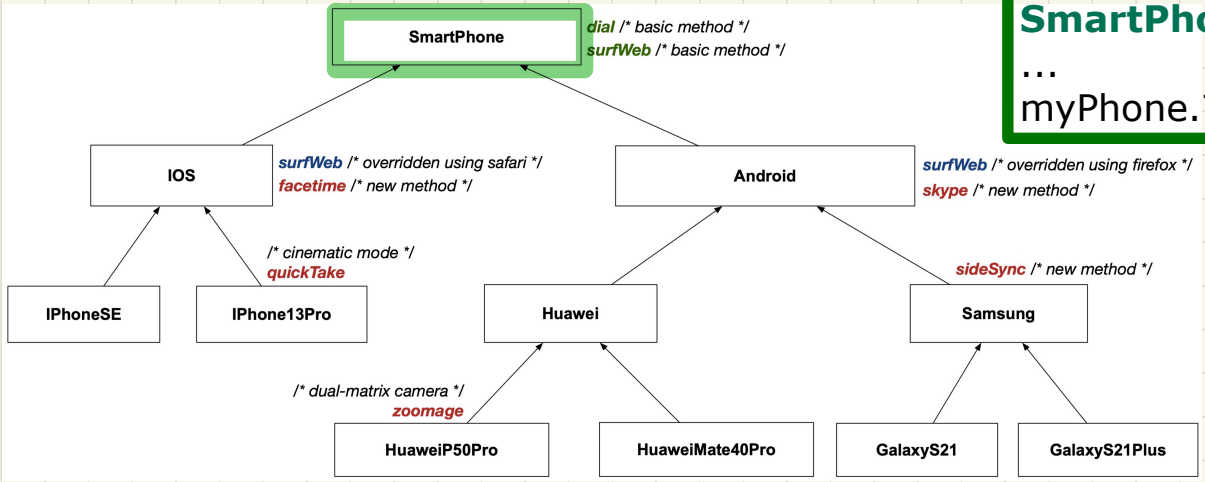
Static Types determine Expectations

Inheritance Herarchy: Students



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

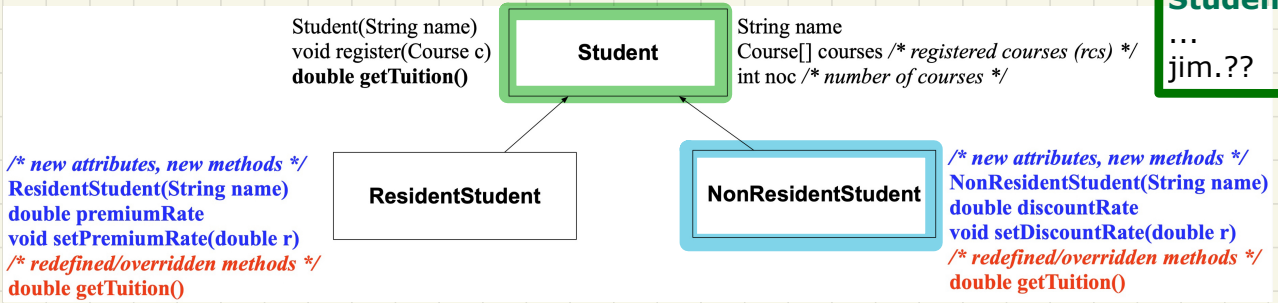
Inheritance Herarchy: Smart Phones



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

Static Types determine Expectations

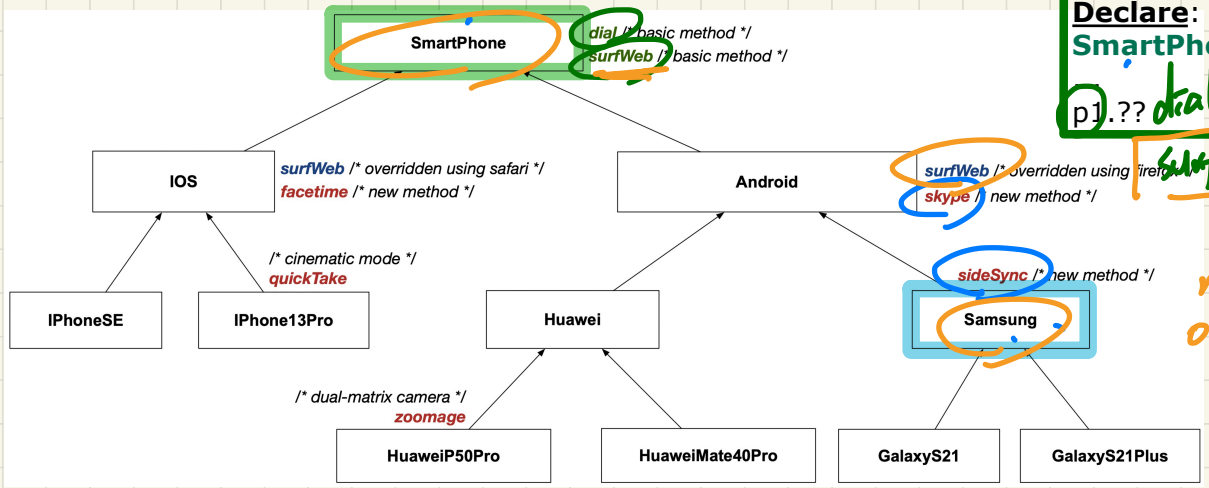
Inheritance Herarchy: Students



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

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

Inheritance Herarchy: Smart Phones

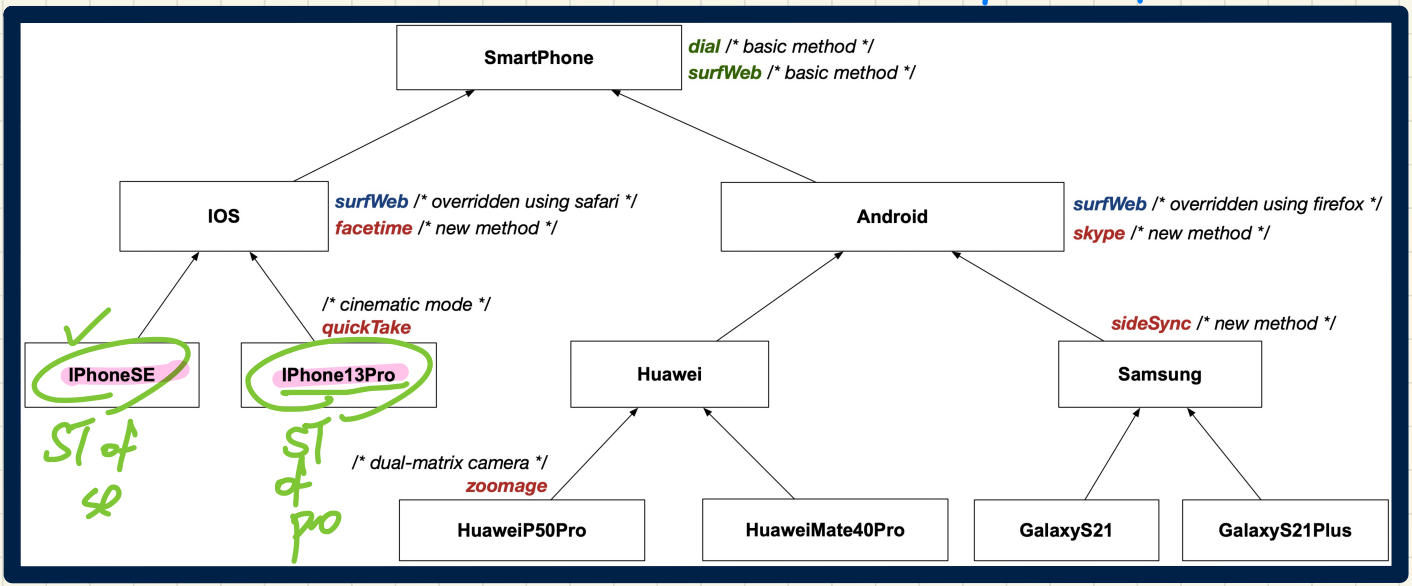


Declare:
SmartPhone p1;
...
p1.?? dial
surfWeb
↓
not overridden

Declare:
Samsung p2;
...
p2.?? dial
surfWeb
skype
sideSync
↓
overridden version

Rules of Substitutions (1)

IPhoneSE se; ① se = pro; ✗
IPhone 13Pro pro; ② pro = se; ✗ Is the ST of pro (IP13Pro) a descendant of the ST of se (IPhoneSE) ?



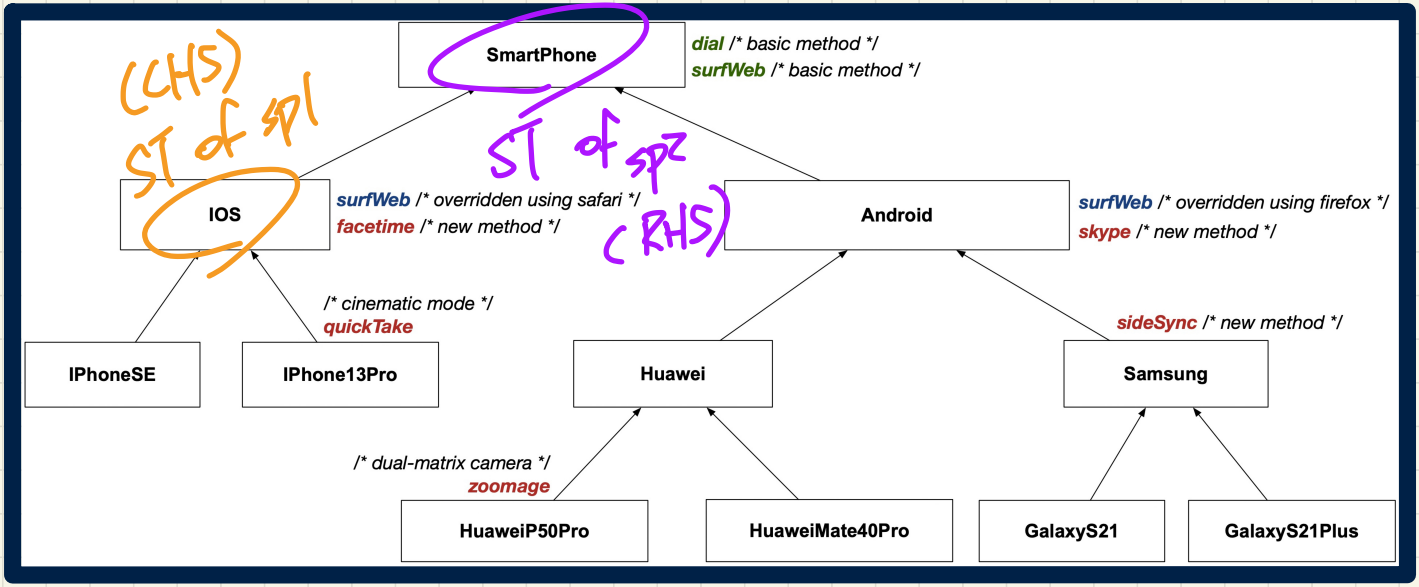
Declarations:

```
IOS sp1;
IPhoneSE sp2;
IPhone13Pro sp3;
```

Substitutions:

```
sp1 = sp2; ✓
sp1 = sp3; ✓
```

Rules of Substitutions (2)



Declarations:

```
IOS sp1;
SmartPhone sp2;
```

Substitutions:

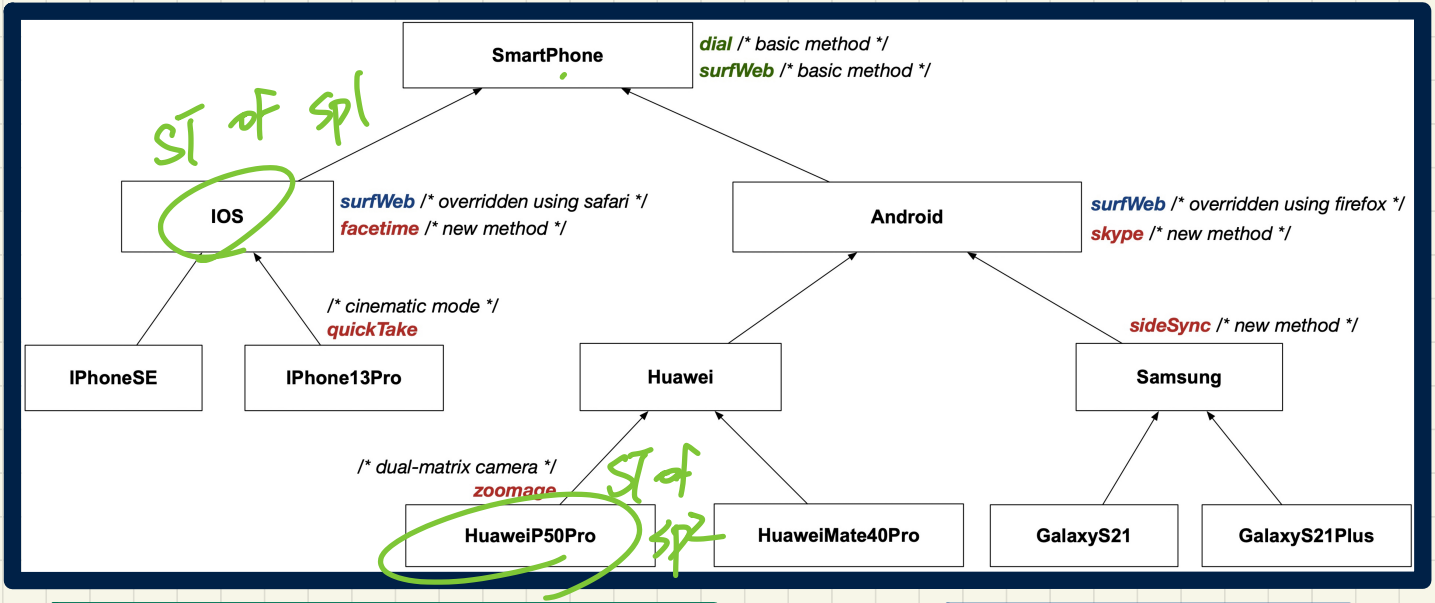
```
sp1 = sp2;
```

X

ST IOS

ST SP

Rules of Substitutions (3)



Declarations:

IOS sp1;

HuaweiP50Pro sp2;

Substitutions:

sp1 = sp2; X

Visualization: Static Type vs. Dynamic Type

Declaration:

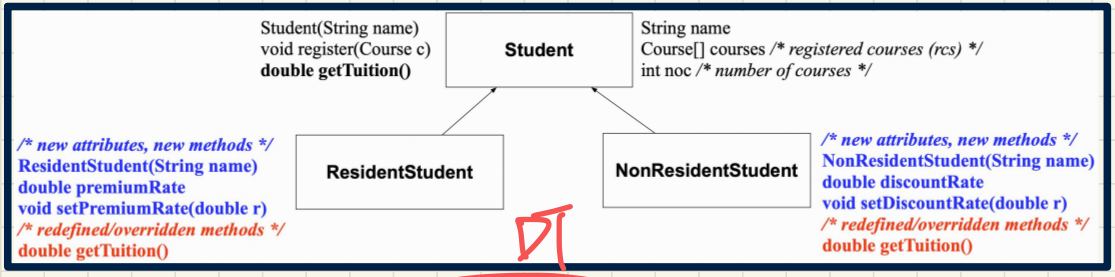
Student s;

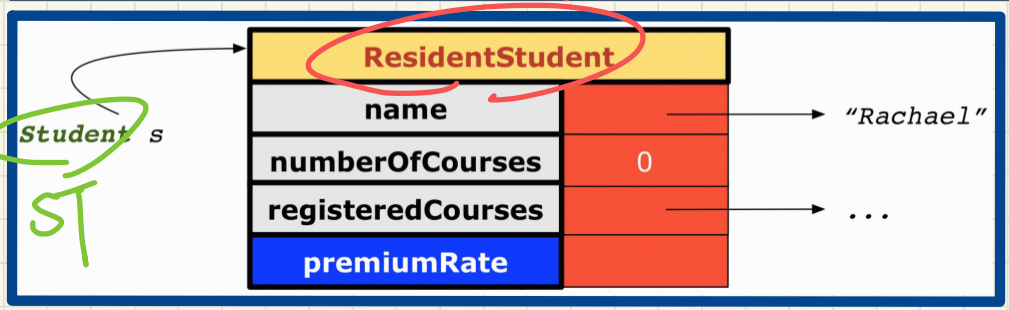
Substitution:

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

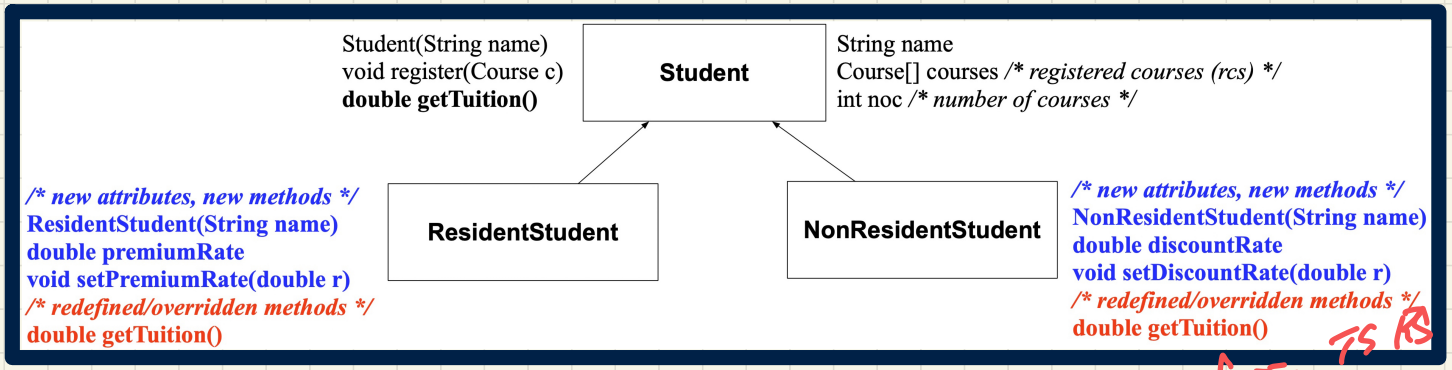
Static Type: Expectation

Dynamic Type: Accumulation of Code





Change of Dynamic Type (1.1)



Example 1:

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

DT1

DT2

DT of jim

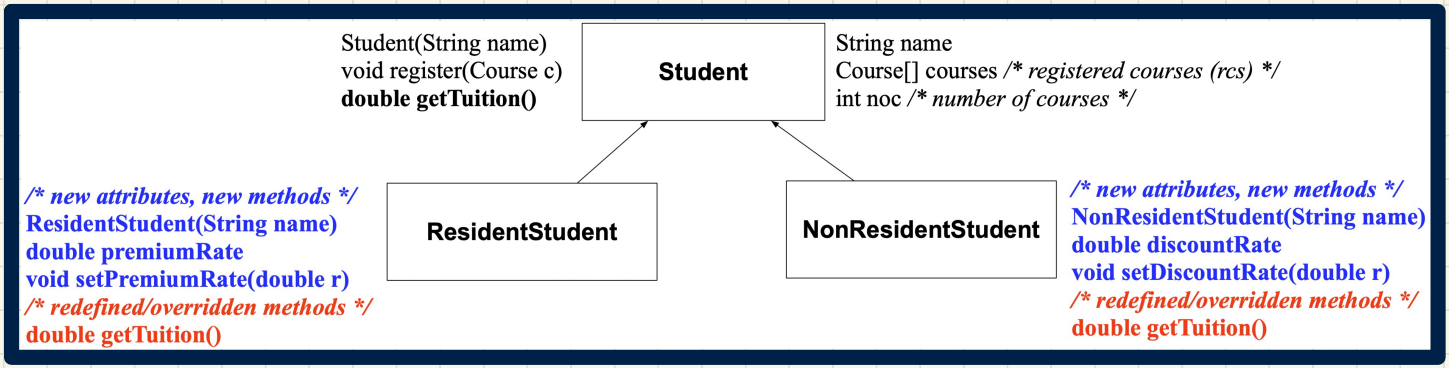
jim manages to RS

jim ✗ RS

jim manages to NRS

Rule: New DT must fulfil the expectations on the ref. var's ST ⇒ new DT is a descendant of ref var's ST.

Change of Dynamic Type (1.2)

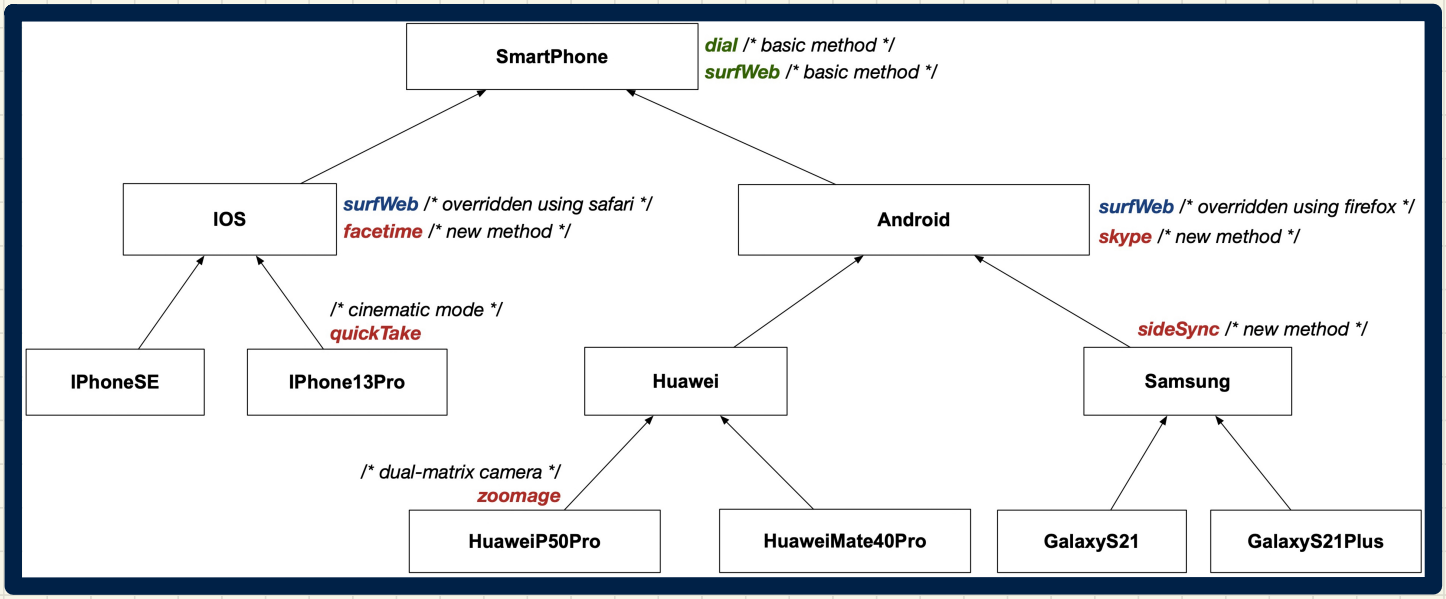


Example 2:

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

not valid
① Student is not a descendant of RS
② Student cannot fulfil exp. of RS.

Change of Dynamic Type: Exercise (1)

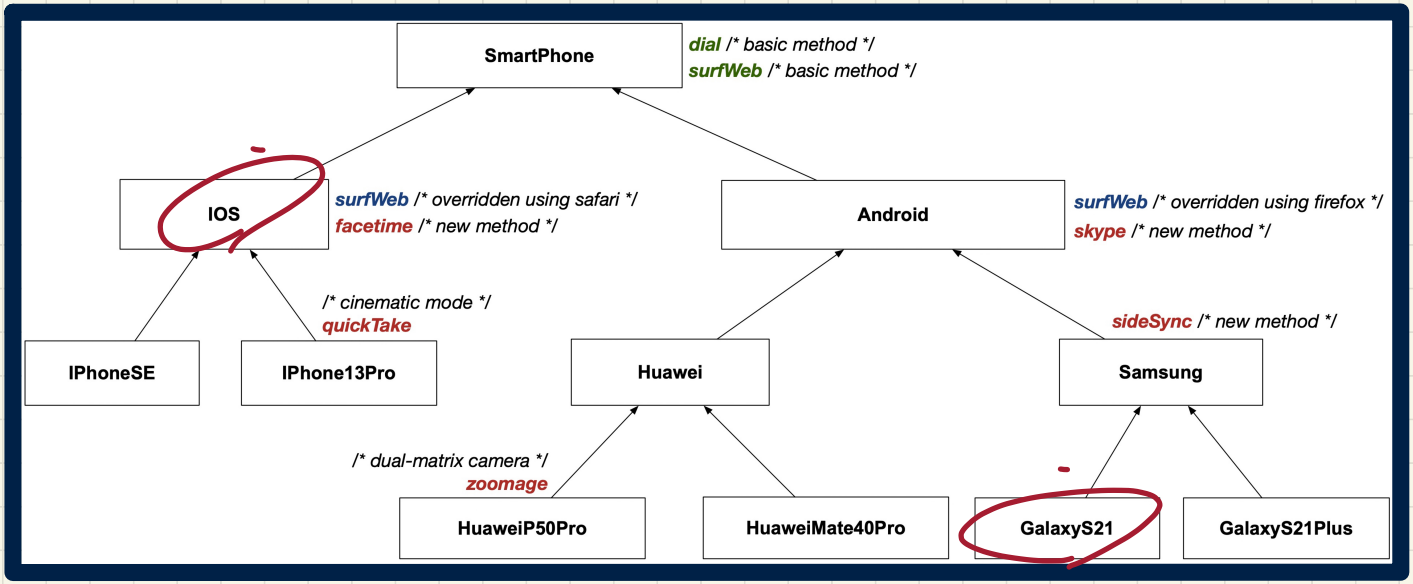


Exercise 1:

Android myPhone = **new HuaweiP50Pro**(...); ✓

myPhone = **new GalaxyS21**(...); ✓

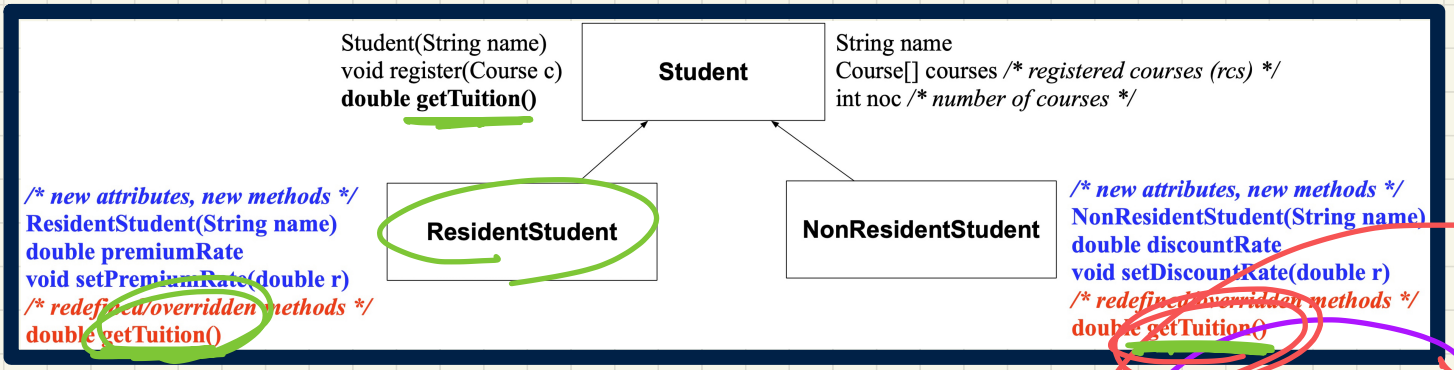
Change of Dynamic Type: Exercise (2)



Exercise 2:

```
IOS myPhone = new HuaweiP50Pro(...);
myPhone = new GalaxyS21(...);
```

Change of Dynamic Type (2.1)



Given:

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

Example 1:

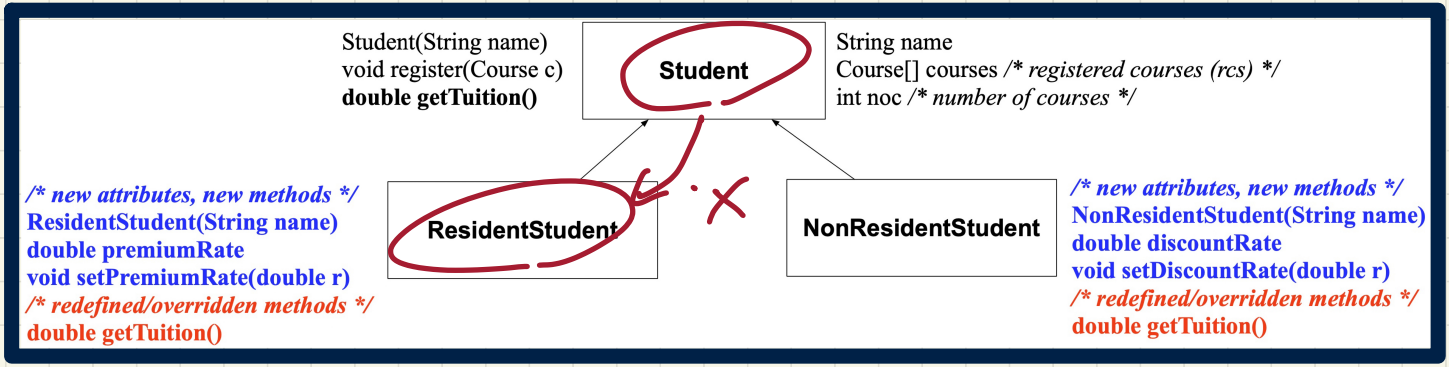
```
① jim = rs; ②
println(jim.getTuition());
jim = nrs; ③
println(jim.getTuition());
```

RS is DT

NRS is DT

	Jim's ST	Jim's DT
①	Student	Student
②		RS
③		NRS

Change of Dynamic Type (2.2)



Given:

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

Example 2:

```
rs = jim;
println(rs.getTuition());
nrs = jim;
println(nrs.getTuition());
```

which version?

which version?

ST of jim (student) not descendant of ST of rs (RS)

Polymorphism and Dynamic Binding

Polymorphism:

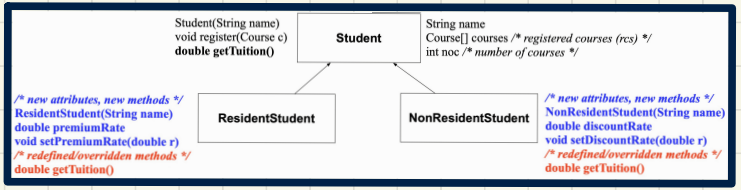
An object's **static type** may allow **multiple** possible **dynamic types**.

⇒ Each **dynamic type** has its **version** of method.

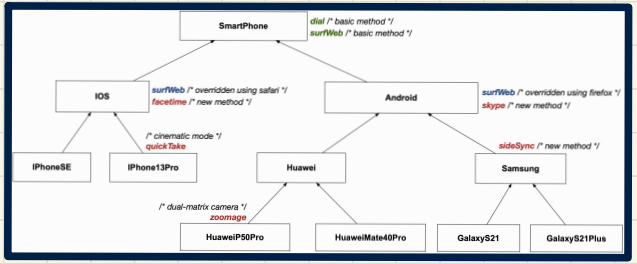
Dynamic Binding:

An object's **dynamic type** determines the **version** of method being invoked.

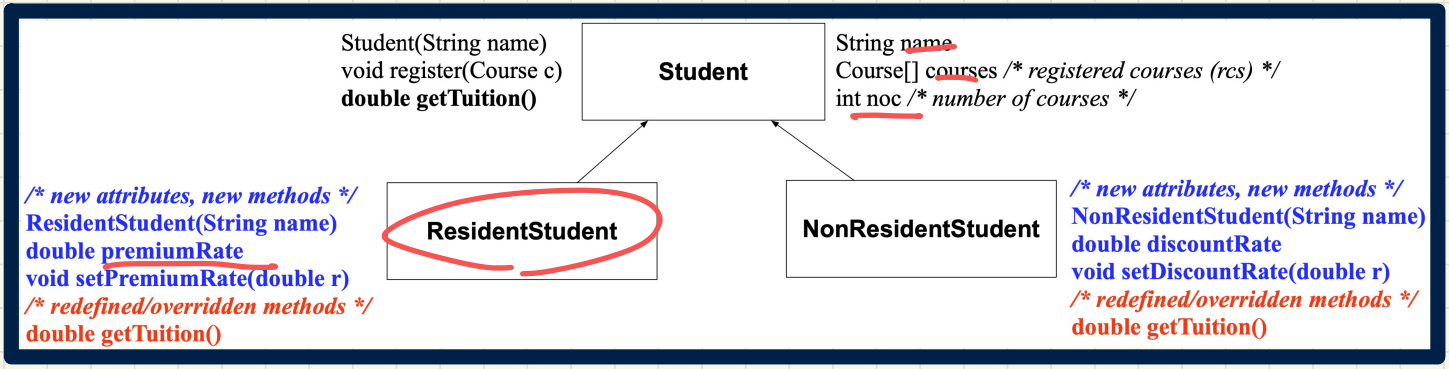
```
Student jim = new ResidentStudent(...);
jim.getTuition();
jim = new NonResidentStudent(...);
jim.getTuition();
```




```
SmartPhone sp = new IPhone13Pro(...);
sp.surfWeb();
sp = new GalaxyS21(...);
sp.getTuition();
```

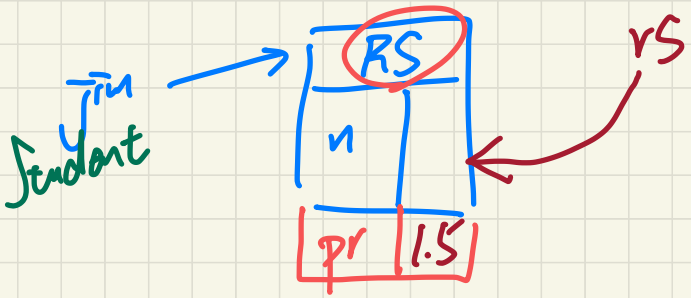


Type Cast: Motivation

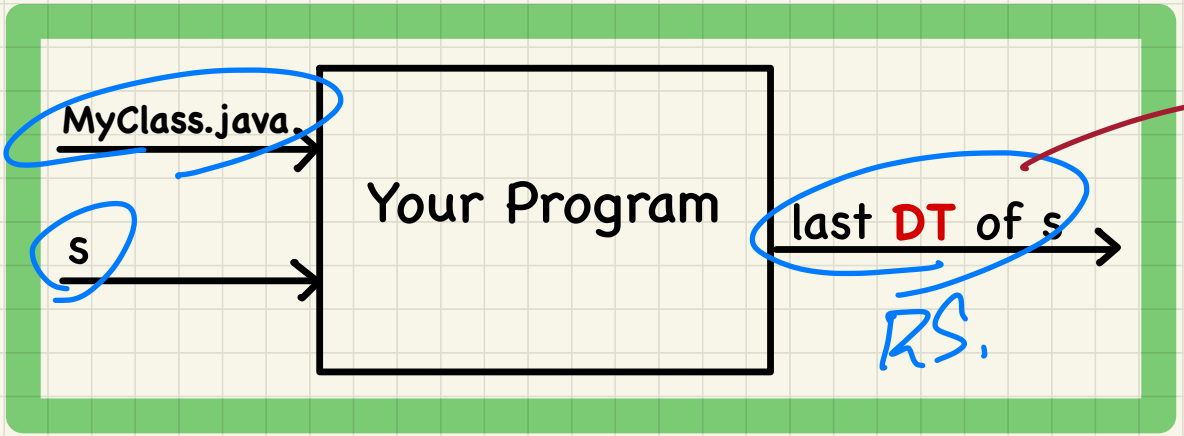



```
1  Student jim = new ResidentStudent("J. Davis");
2  ResidentStudent rs = jim;
3  rs.setPremiumRate(1.5);
```

ST: Student



An A+ Challenge: Inferring the DT of a Variable



undecidable to figure out the dynamic behaviour of a program

```
class MyClass {
   main (...)
      Student s = ...;
      ...
      s = new ResidentStudent(...);
   }
}
```

last DT

EECS 200!