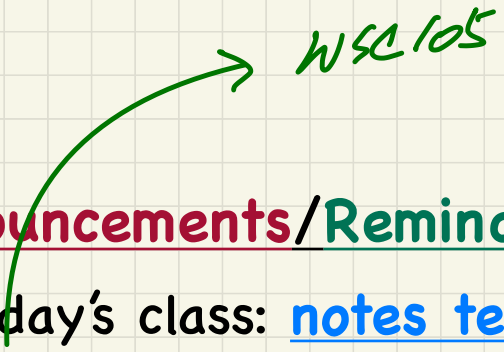
Lecture 18 - Nov 10

Inheritance

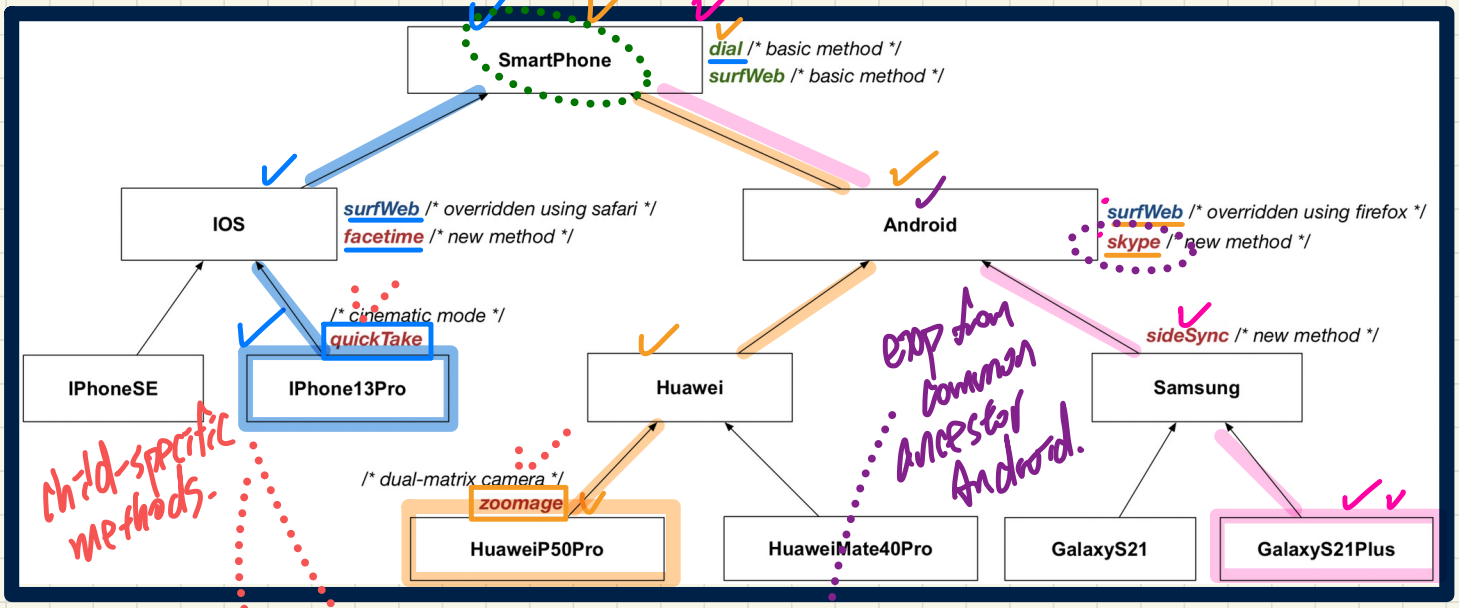
Ancestors, Descendants, Expectations
Rules of Substitutions
Polymorphism vs. Dynamic Binding



Announcements/Reminders

- Today's class: notes template posted
- Lab4 released
- WrittenTest2 guide and example questions to be released

Multi-Level Inheritance Hierarchy: Smartphones



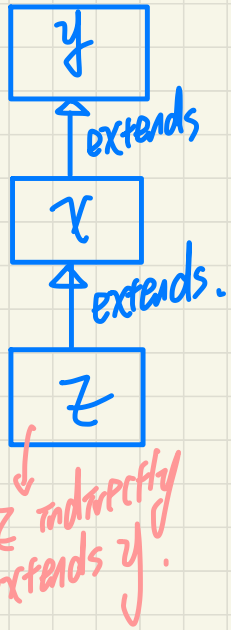
Exercise Compare the ranges of expectations of:

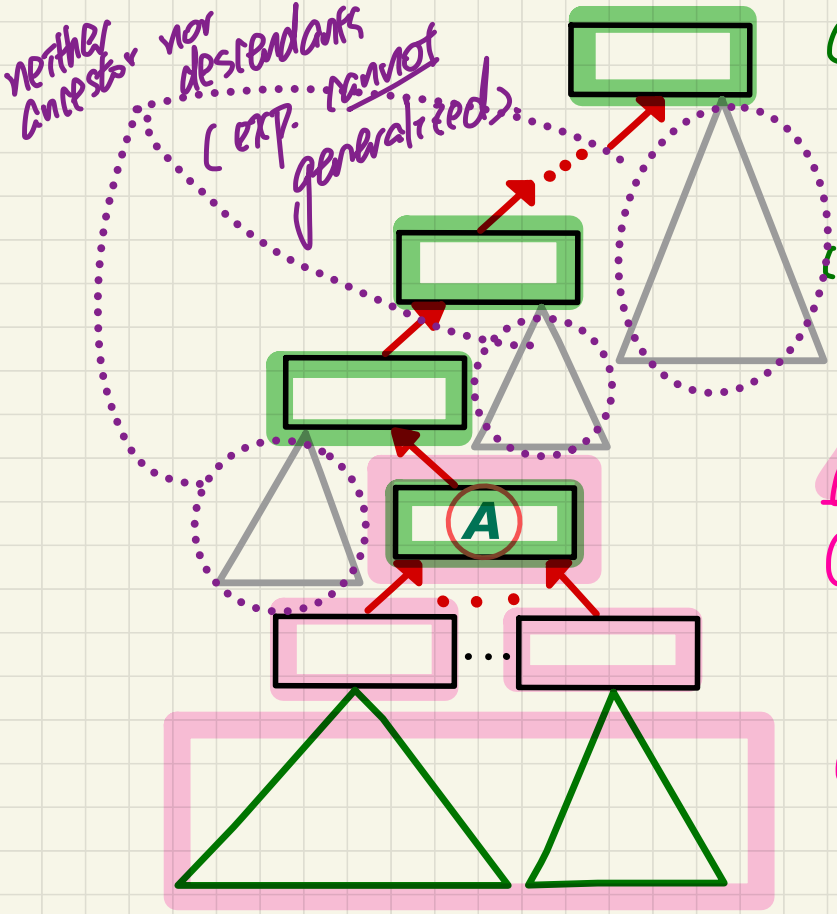
+ IPhone13Pro
+ HuaweiP50Pro
+ GalaxyS21Plus

QT, facetime, surfweb, dial
zoomage, skype, surfweb, dial
sideSync, skype, surfweb, dial

common expe. inherited from Smartphone

Inheritance Forms a Type Hierarchy





ancestors of A

exp.

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

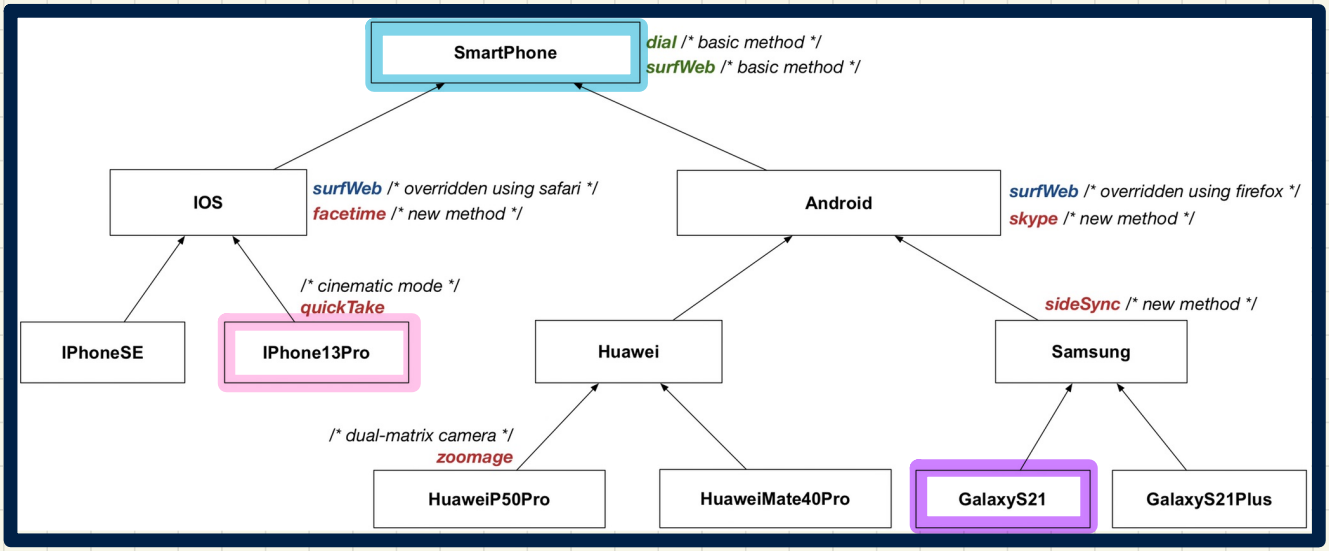
(2) exp(A) ≥ exp of any ancestor class

descendants of A

(1) exp(A) inherited to all of A's descendants

(2) exp(A) ≤ exp of any descendant class

Inheritance Accumulates Code for Reuse



	ancestors	expectations	descendants
	GS21, Sam., An, SP	Execise.	GS21
	SP.		all classes.

Rules of Substitutions

Compilation (static types).

```
A oa = ...;
? ob = ...;
oa = ob;
```

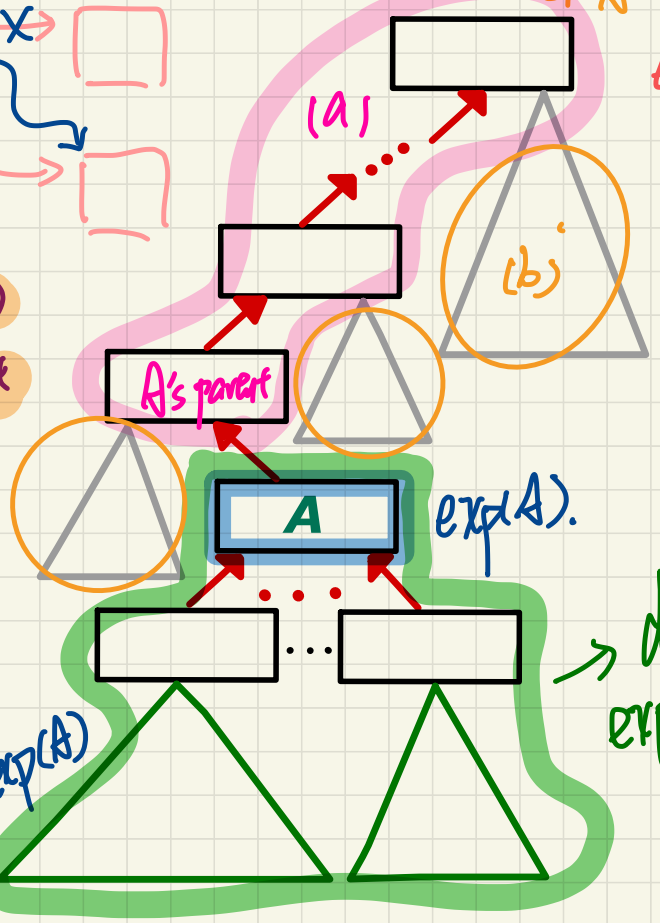
ST of oa

ST of ob.

what can ? (ST of ob) be in order for this var. assignment to compile ?

oa → X

ob →



* oa's expectation must be fulfilled by ob's

ST: A

ST: ?

exp. :

exp(?) ≥ exp(A).

↳ ? must be a descendant of A

descendants of A

exp ≥ exp(A).

if ? is one of these → fail to compile

(a) ancestors of A's parent

(b) neither an. nor des

• substitute oa by ob

• after the substitution, we expect: oa.* —— exp(A)

does this var assignment (substitution) compile?

var1 = var2

ST: T_1 ST: T_2

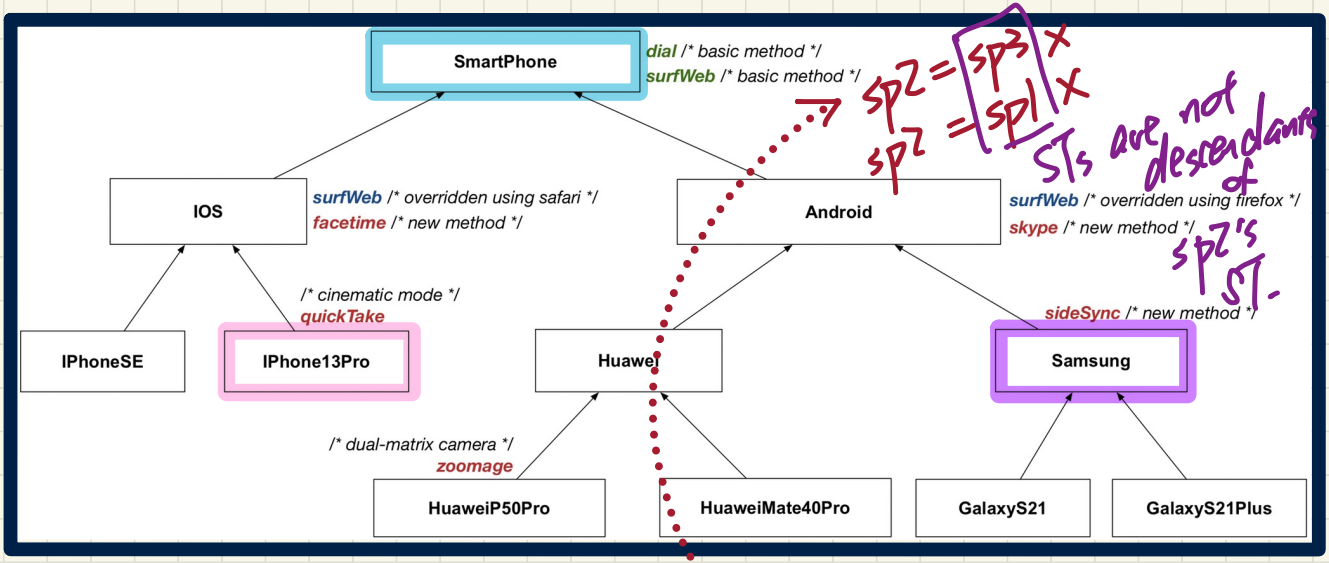
(inheritance).

Rule of substitutions

$exp(T_2) \Rightarrow exp(T_1)$

T_2 must be a descendant of T_1

Inheritance Accumulates Code for Reuse



sp2 = sp3 ✗
sp2 = sp1 ✗
STs are not descendants of sp2's ST.

SmartPhone sp1; ✓
IPhone13Pro sp2; ✓
Samsung sp3; ✓

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

sp1's ST exp.

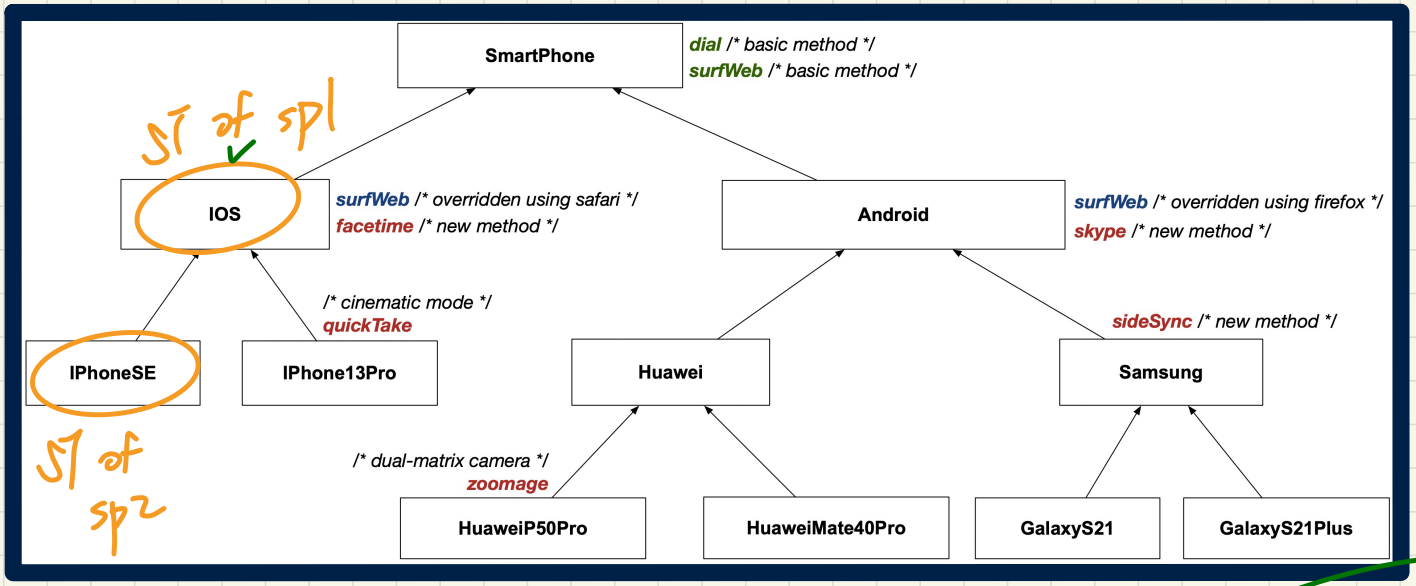
* Q. What var can appear on the RHS without compilation error?

(A1) Any var whose ST is a descendant of sp1's ST can appear on RHS.

(A2) to substitute sp1, its ST must have at least as large the exp of

sp1
sp2
sp3

Rules of Substitutions (1)



sp2's ST can fulfill the exp of sp1's ST.

Declarations:

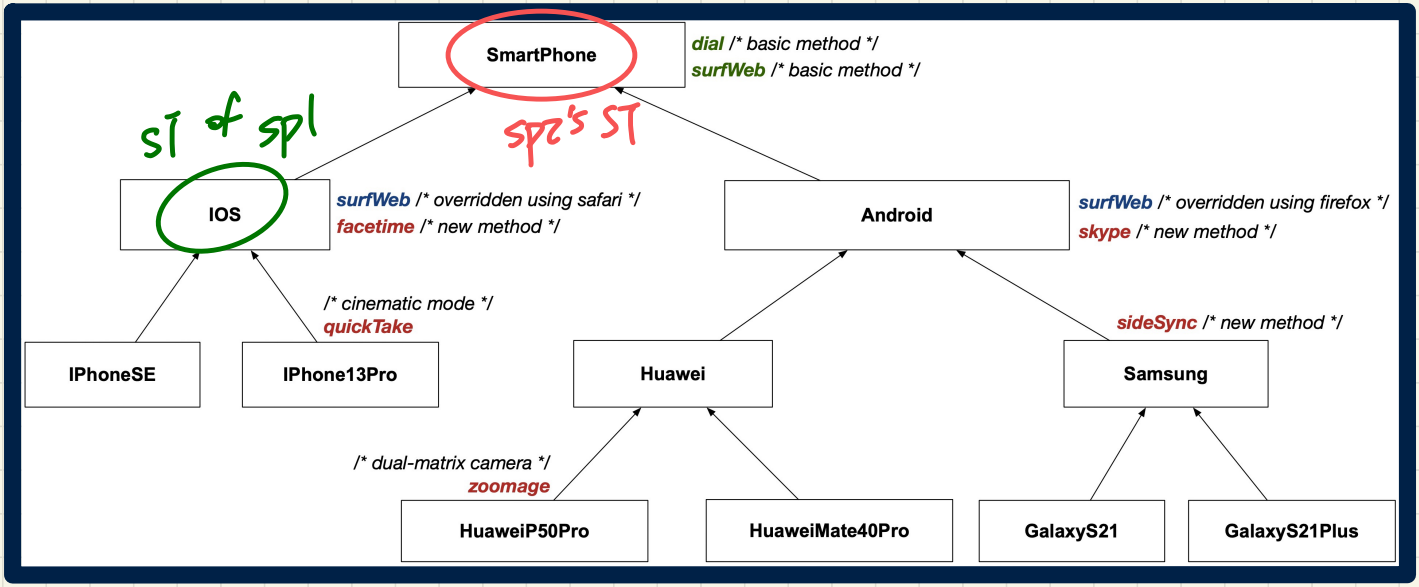
```
IOS sp1;
IPhoneSE sp2;
IPhone13Pro sp3;
```

Substitutions:

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

Compiles ∵ ST of sp2 (IPSE) is a descendant of ST of sp1 (IOS).

Rules of Substitutions (2)



Declarations:

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

Substitutions:

```
sp1 = sp2;
```

fail to compile.

Visualization: Static Type vs. Dynamic Type

Declaration:

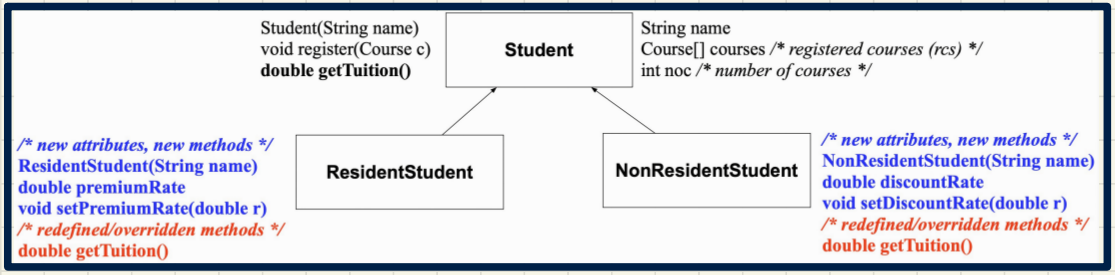
Student s;

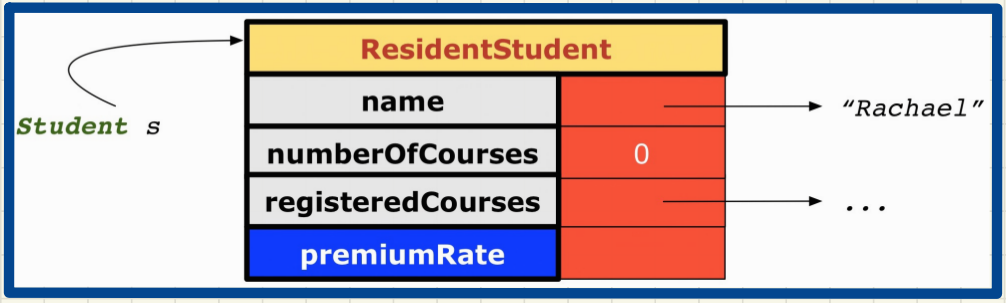
Substitution:

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

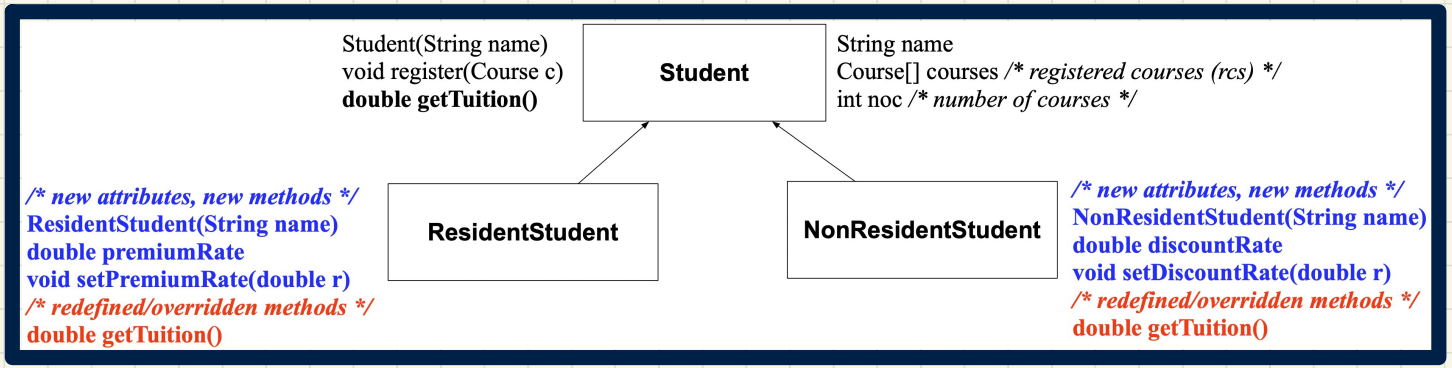
Static Type: Expectation

Dynamic Type: Accumulation of Code





Change of Dynamic Type (1.1)



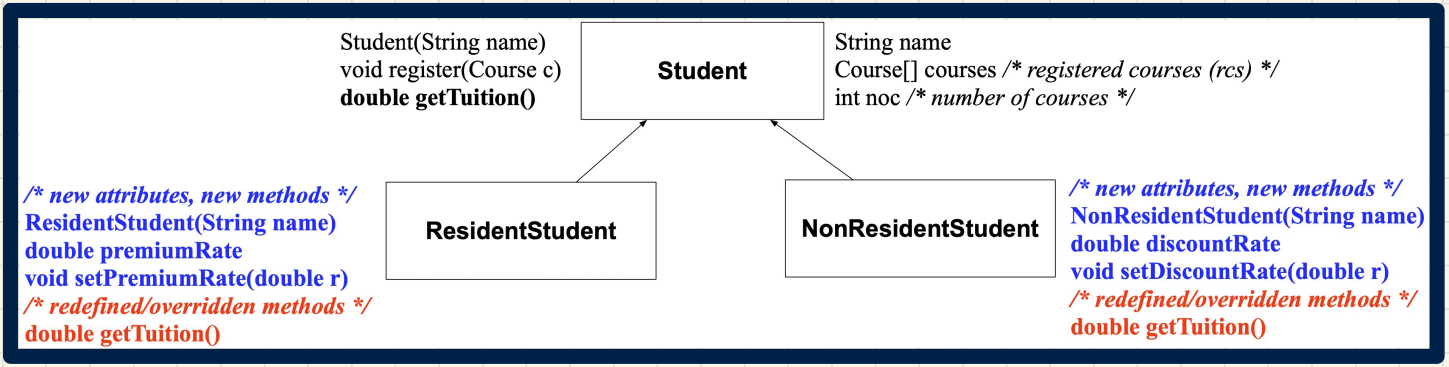
Example 1:

✓ compiles ∵ RS is a descendant of Student

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

compiles ∵ NRS is a descendant of Student

Change of Dynamic Type (1.2)

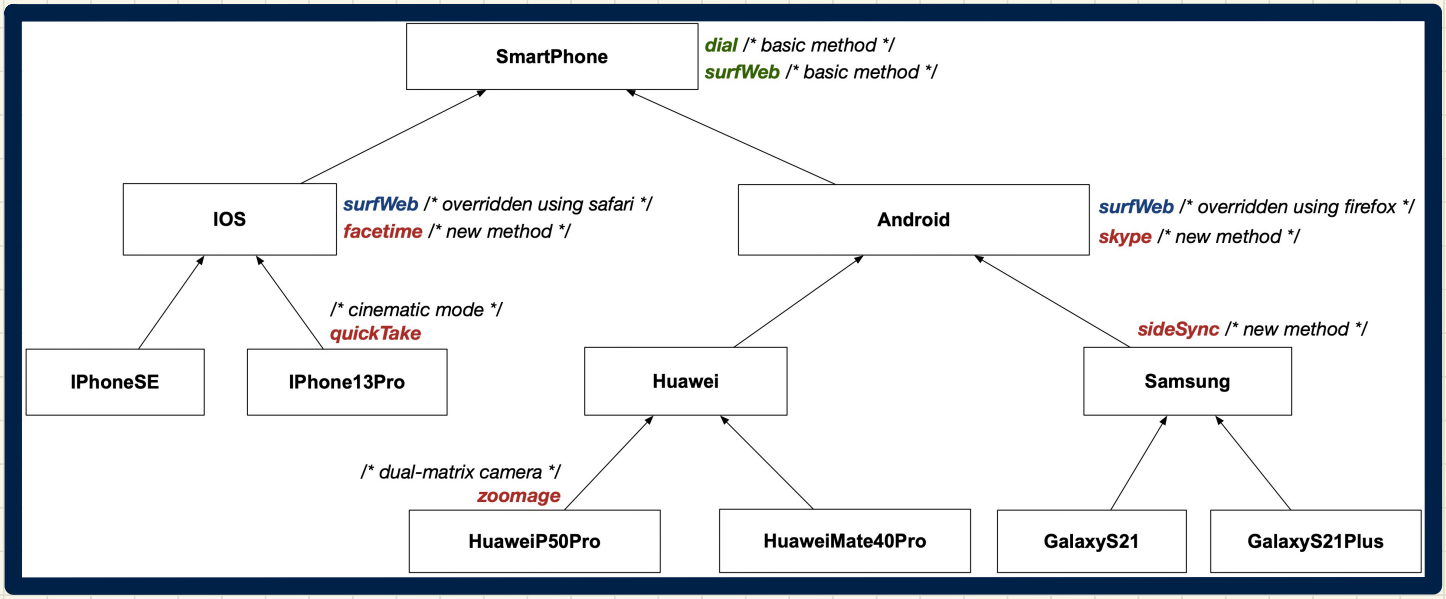


Example 2:

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

fail to compile
:(Student is not a descendant of RS
proposed DT
ST of jeremy.

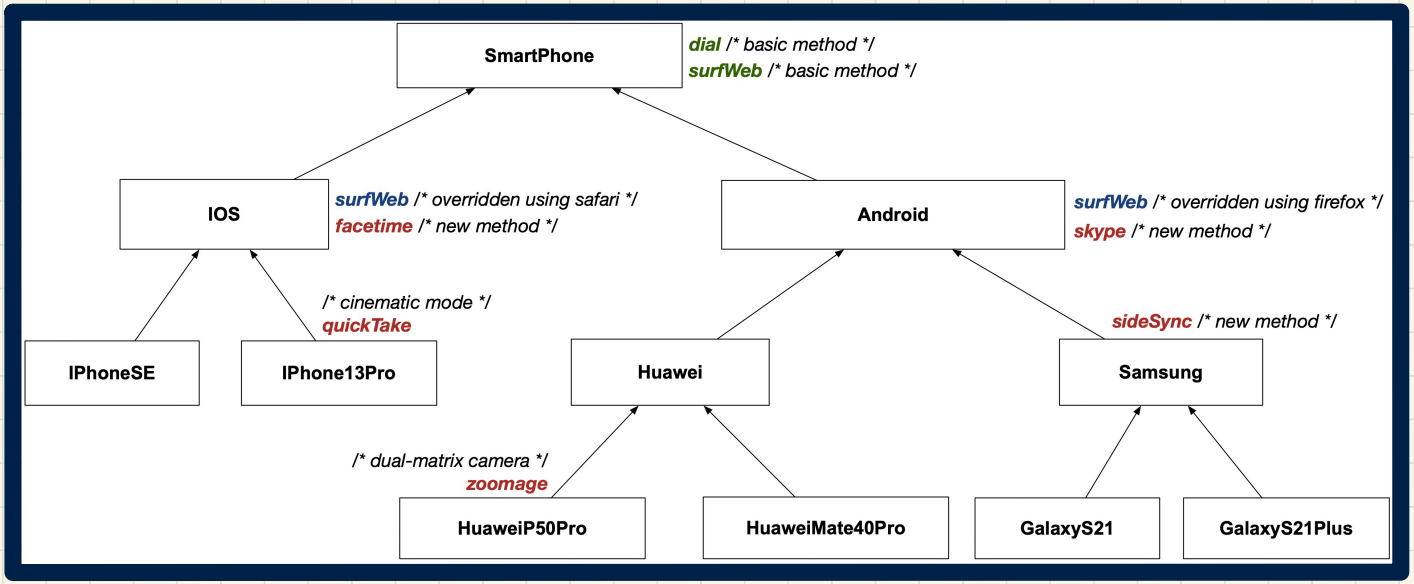
Change of Dynamic Type: Exercise (1)



Exercise 1:

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

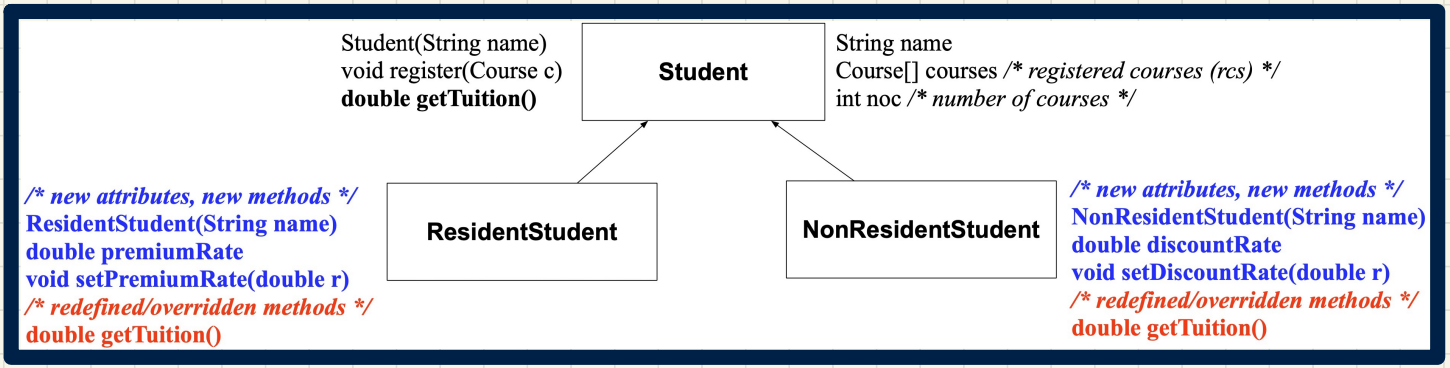
Change of Dynamic Type: Exercise (2)



Exercise 2:

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

Change of Dynamic Type (2.1)



Given:

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

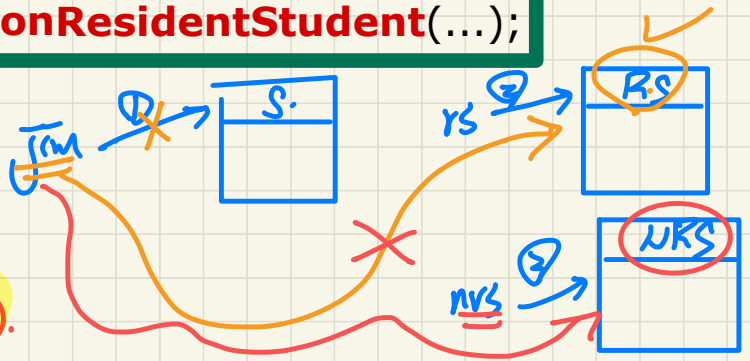
DT of jim: Student

Example 1:

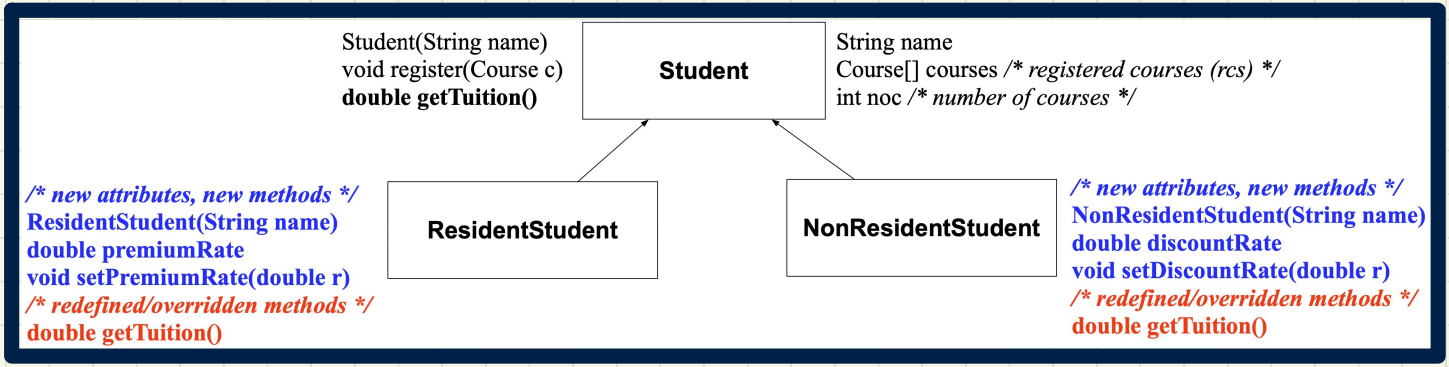
```
4  jim = rs;
   println(jim.getTuition());
5  jim = nrs;
   println(jim.getTuition());
```

DT of jim: changed to the DT of rs (RS)

DT of jim: changed to the DT of nrs (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());
```

not compile
∵ jim's ST (Student) not ≡ descendant of rs' ST (RS)
cannot fulfill exp of.

Polymorphism and Dynamic Binding

Polymorphism: shapes (DTs).

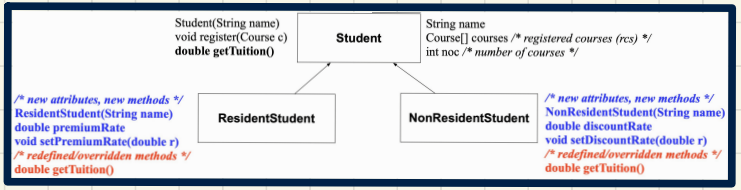
An object's **static type** may allow **multiple** possible **dynamic types**.

⇒ 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 sp1 = new IPhone13Pro(...);
SmartPhone sp2 = new GalaxyS21(...);
sp1.surfWeb();
sp1 = sp2;
sp1.surfWeb();
```