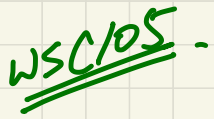
# Lecture 18 - Nov 11

## Inheritance

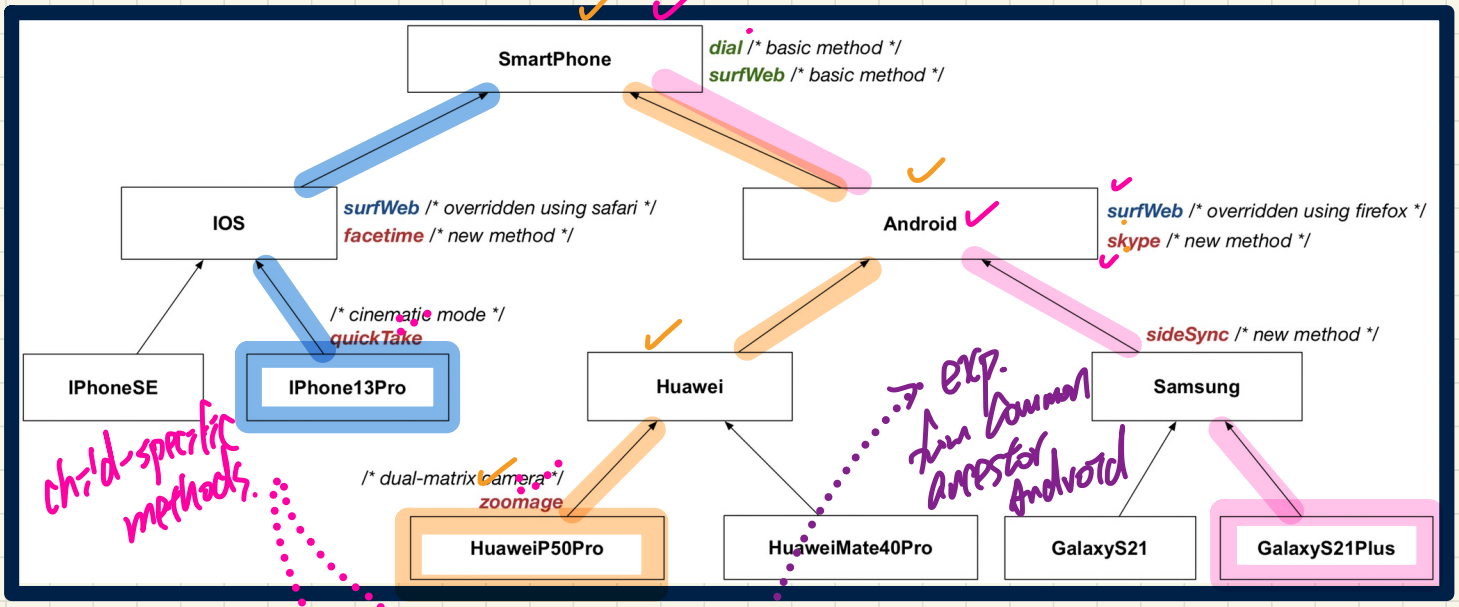
***Ancestors, Descendants, Expectations***
***Rules of Substitutions***




## Announcements/Reminders

- Today's class: notes template posted
- Lab4 released
- Tracing Exercises: assertEquals and Person, PersonCollector

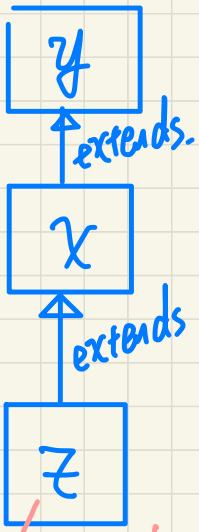
# Multi-Level Inheritance Hierarchy: Smartphones



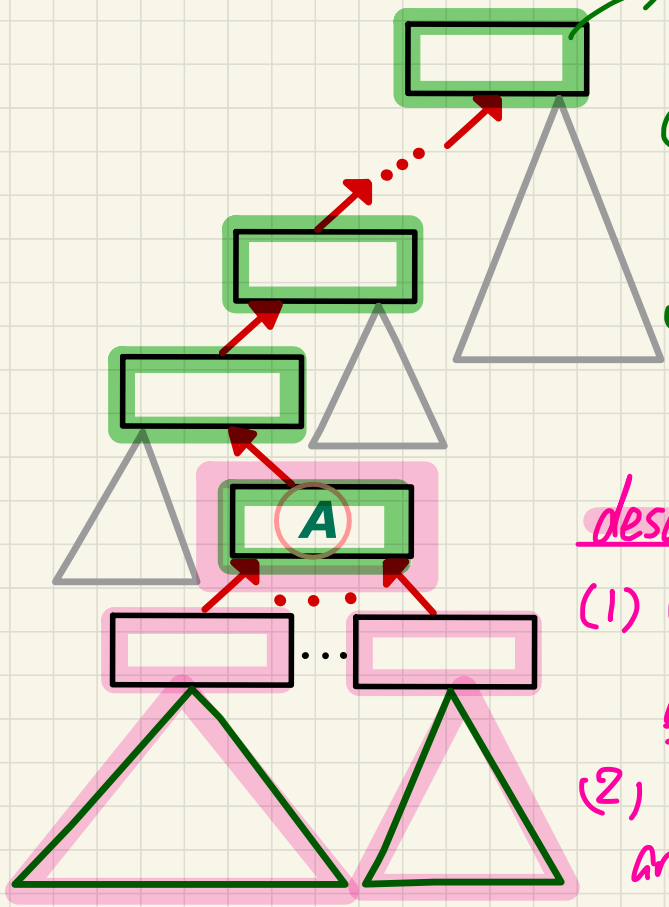

**Exercise** Compare the ranges of expectations of:

+ IPhone13Pro
+ HuaweiP50Pro
+ GalaxyS21Plus

# Inheritance Forms a Type Hierarchy




Z indirectly extends Y.




root.

**ancestors of A**

(1) A accumulates all atts/mths. from its ancestors
expectation

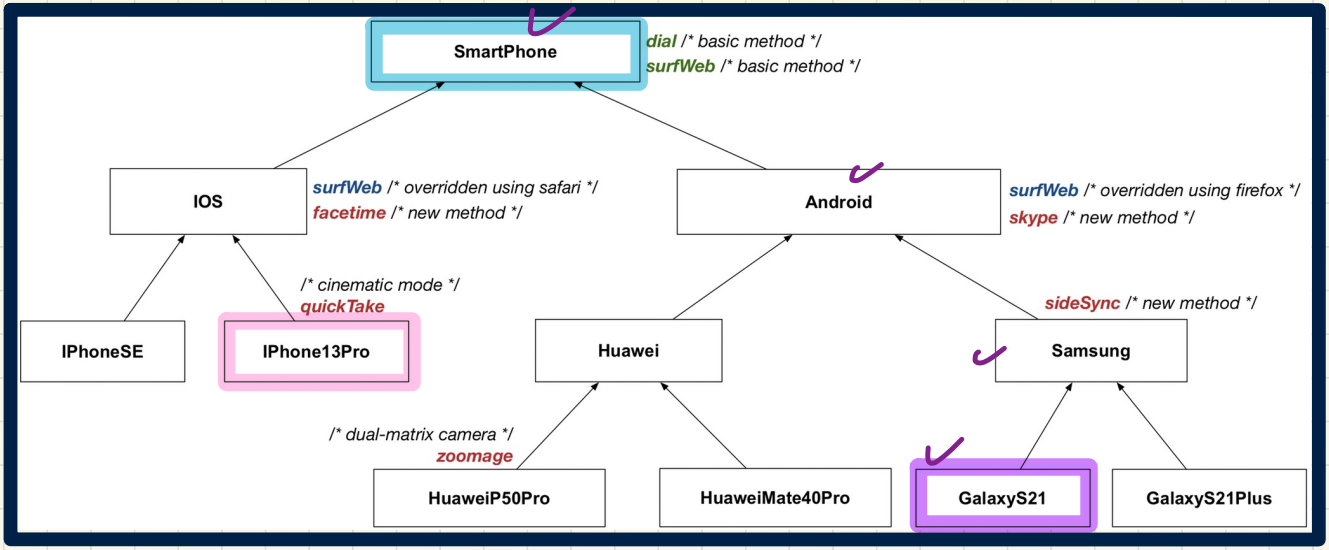
(2) exp(A) ≥ exp of any ancestor class

**descendants of A**

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

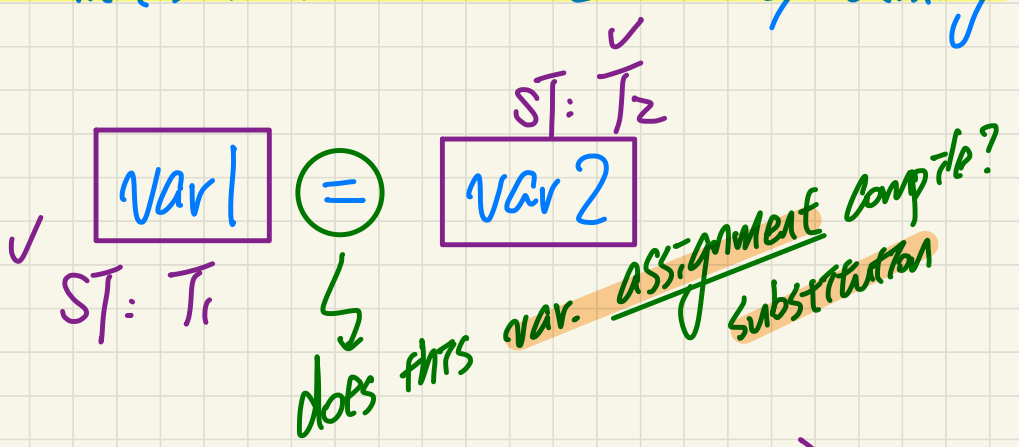
(2) exp(A) ≤ exp of any decendant class.

# Inheritance Accumulates Code for Reuse




| | ancestors | expectations | descendants |
|---|---|---|---|
| | GS21, S.., A.., SP | Exercise | GS21 |
| | | | |
| | SP | | all classes |

assumption: there's some inheritance hierarchy existing




## Rule of substitutions (inheritance).

T2 must be a descendant of T1

exp(T2) >= exp(T1)

# Rules of Substitutions

does a var. assignment compile? (static types; inheritance)

* oa's expectations must be fulfilled by the ST of ob.

exp(?) ≥ exp(A)

↳ ? must be a descendant of A.

ST of oa

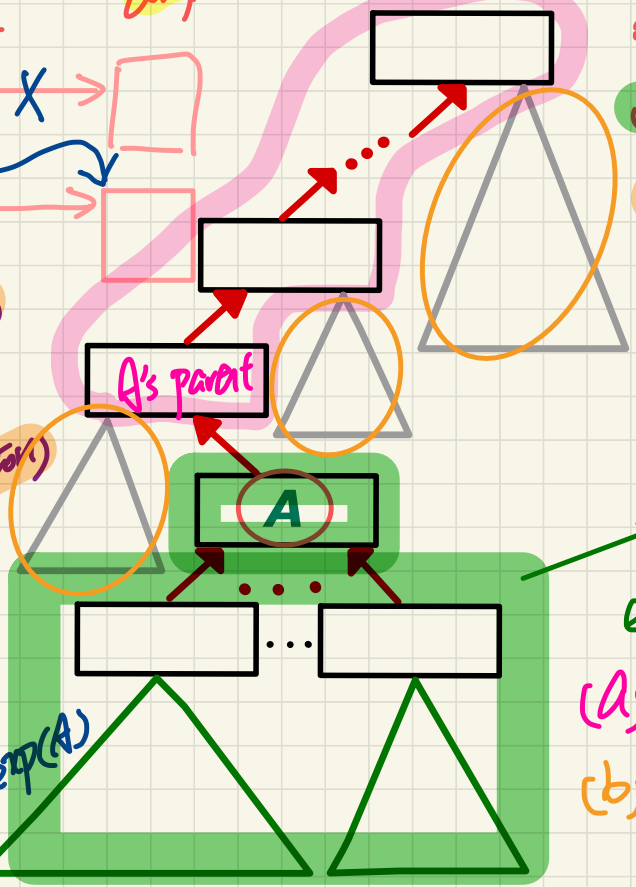
ST of ob

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

ST: A

oa ✗ → ob

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




descendants of A

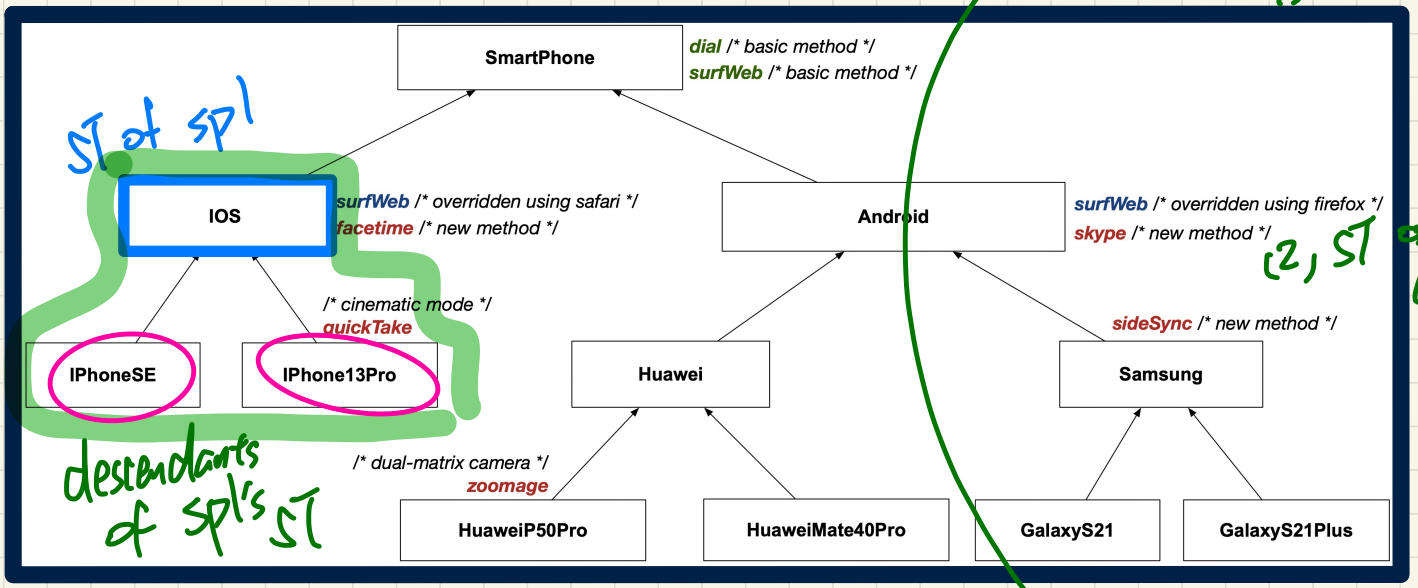
exp ≥ exp(A).

(a) ancestors of A's parent

(b) neither ancestor nor descendant

- substitute oa by ob
- after the substitution we expect: *oa.

# Rules of Substitutions (1)




Compiles if: (1) ST of sp2 (IPSE) is a descendant of the ST of sp1 (IOS).

(2) ST of sp2 can fulfill the exp of sp1's ST.

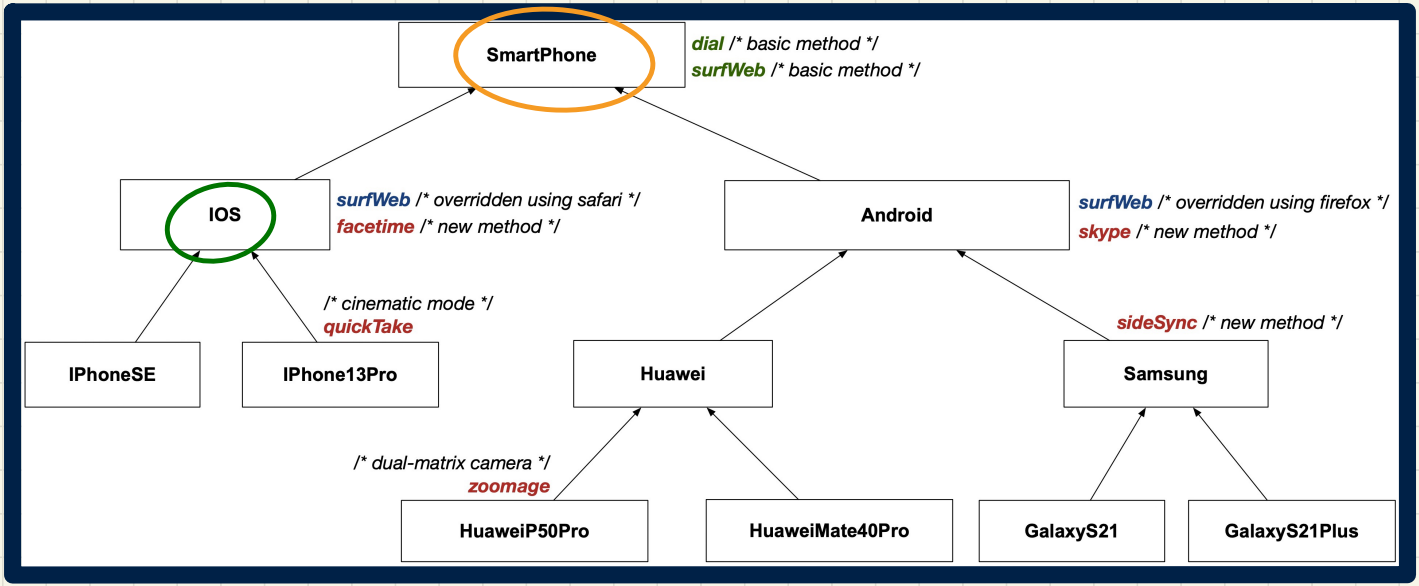
**Declarations:**

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

**Substitutions:**

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

# Rules of Substitutions (2)




**Declarations:**

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

**Substitutions:**

```
sp1 = sp2;
```

fail to compile - SP is not descendant of IOS

# Visualization: Static Type vs. Dynamic Type

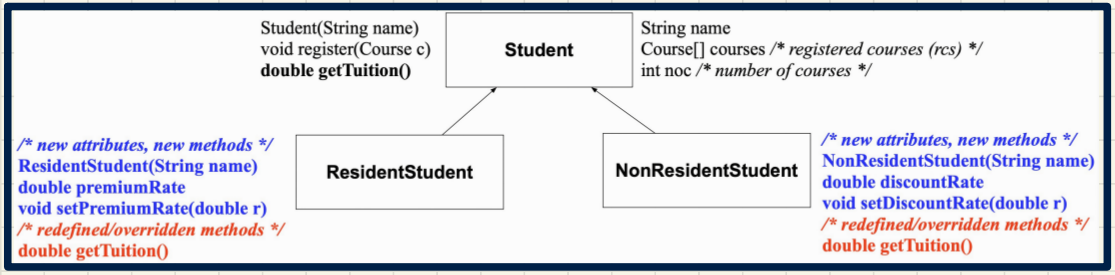
**Declaration:**
**Student** s;

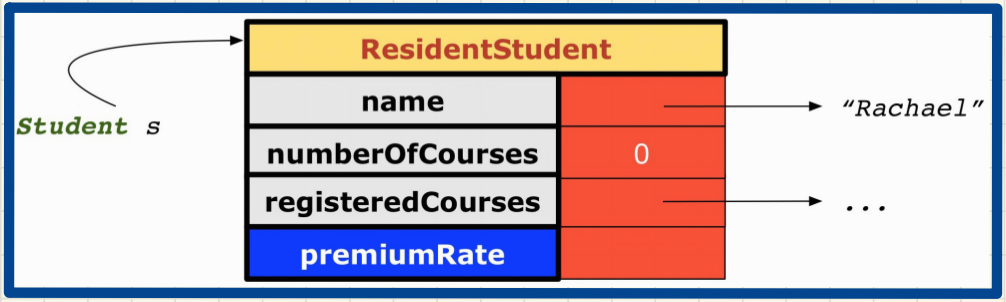
**Substitution:**
s = **new ResidentStudent**("Rachael");

Compiles :: (1) RS is a des. of the ST of s (Student)

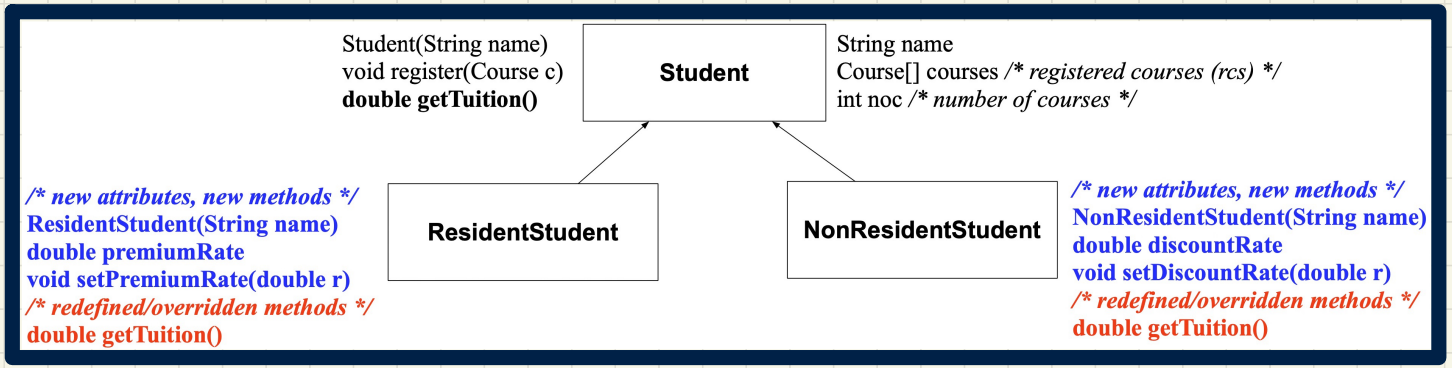
**Static Type**: Expectation
**Dynamic Type**: Accumulation of Code

(2) RS can fulfill the exp. of s' ST (Student)



# Change of Dynamic Type (1.1)

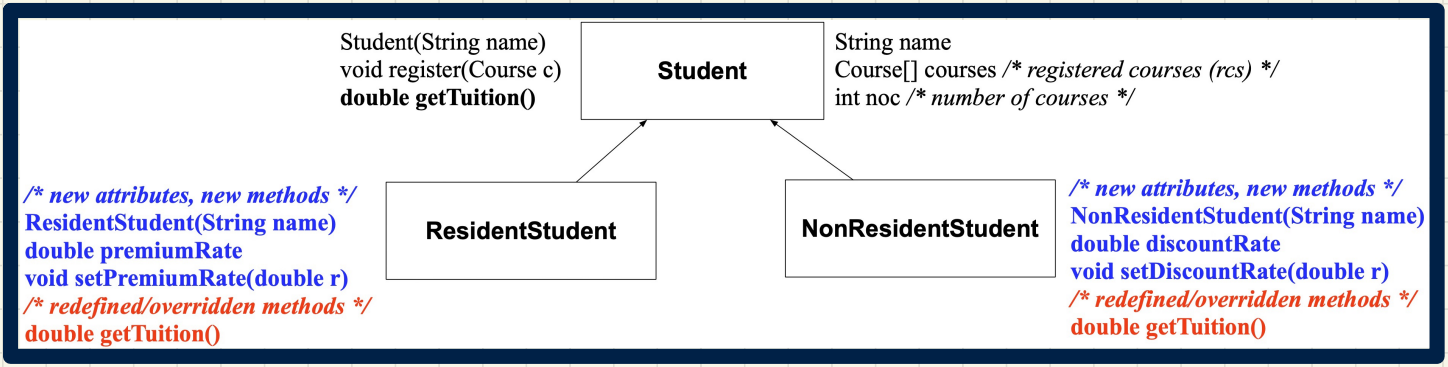



## Example 1:

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

Compile-time RS and NRS are descendants of jim's ST (Student).

# Change of Dynamic Type (1.2)

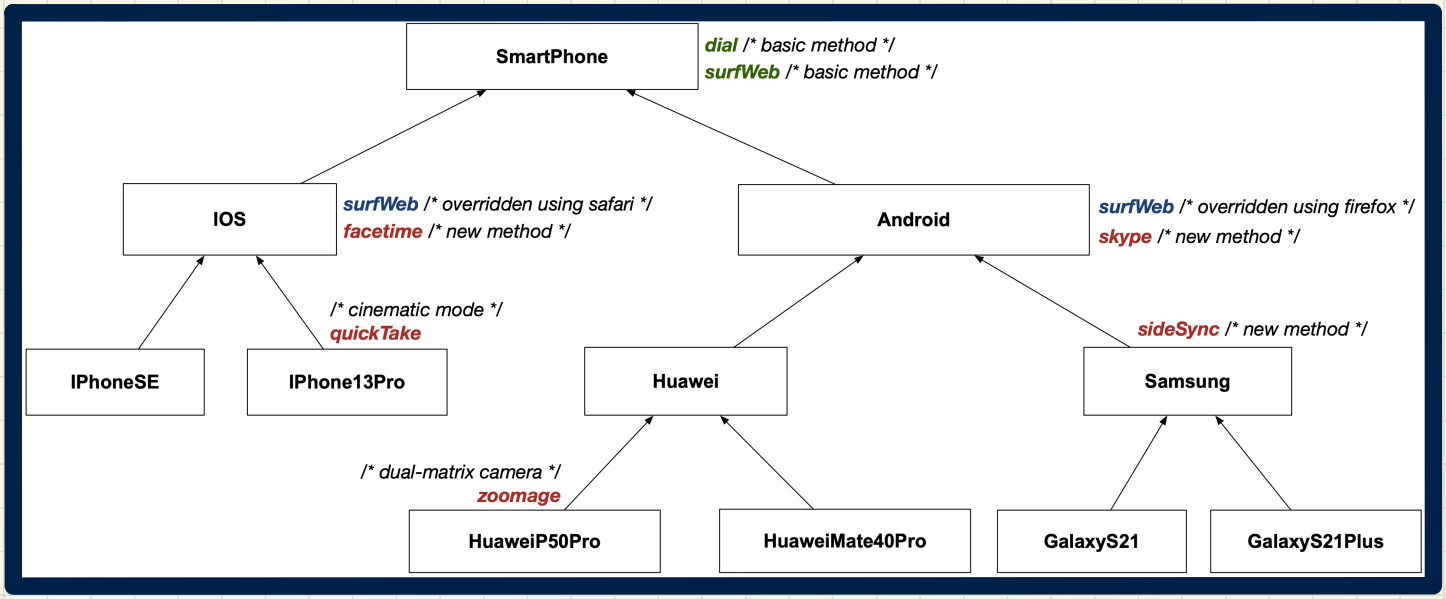



## Example 2:

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

fail to compile
∵ Student cannot fulfil exp of jeremy's ST (RS).

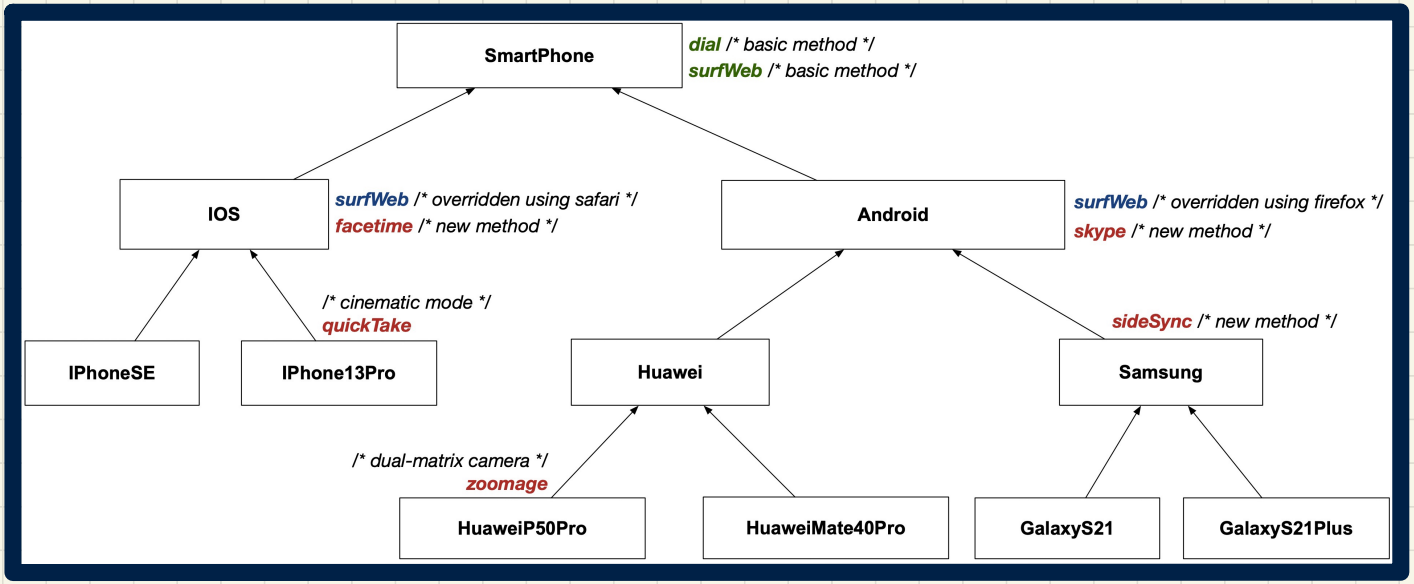
# Change of Dynamic Type: Exercise (1)




## Exercise 1:

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

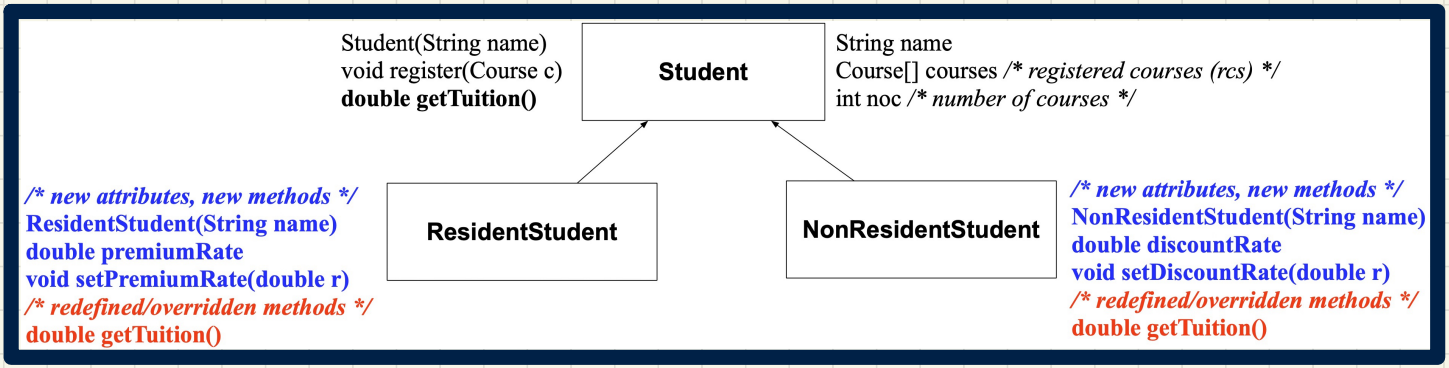
# Change of Dynamic Type: Exercise (2)




## Exercise 2:

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

# Change of Dynamic Type (2.1)




**Given:**

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

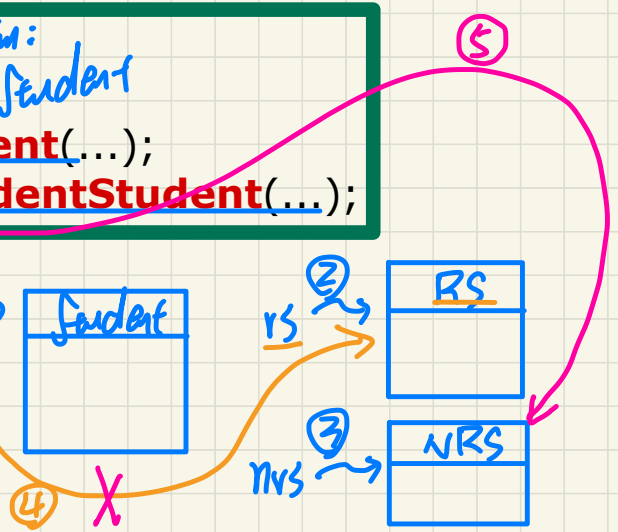
DT of jim: Student

**Example 1:**

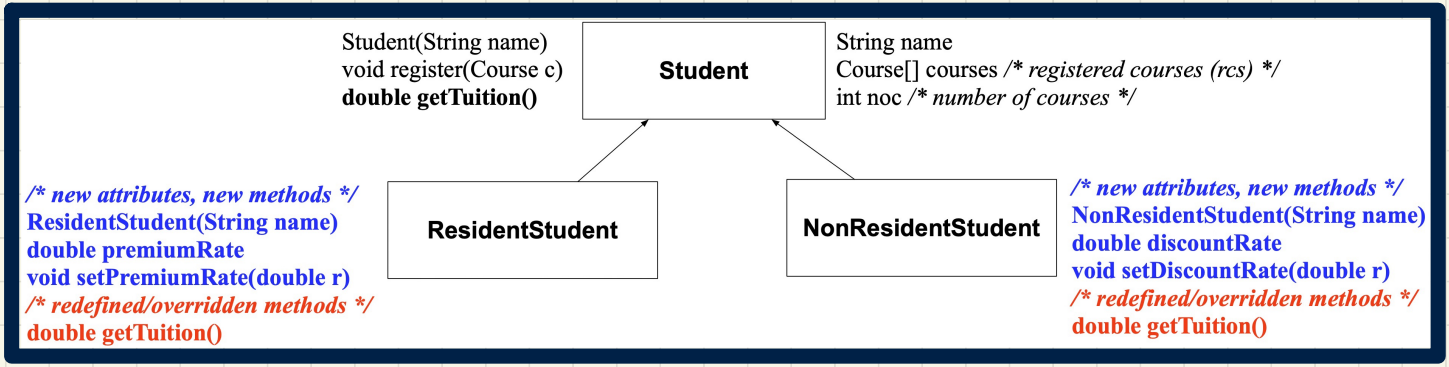
```
④ jim = rs;
println(jim.getTuition());
⑤ jim = nrs;
println(jim.getTuition());
```

DT of jim: changed to D.T. of rs (ResidentStudent)

DT of jim: changed to D.T. of nrs (NonR.S.)



# 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
∵ Student (jim's ST) not descendant of ST of LHS