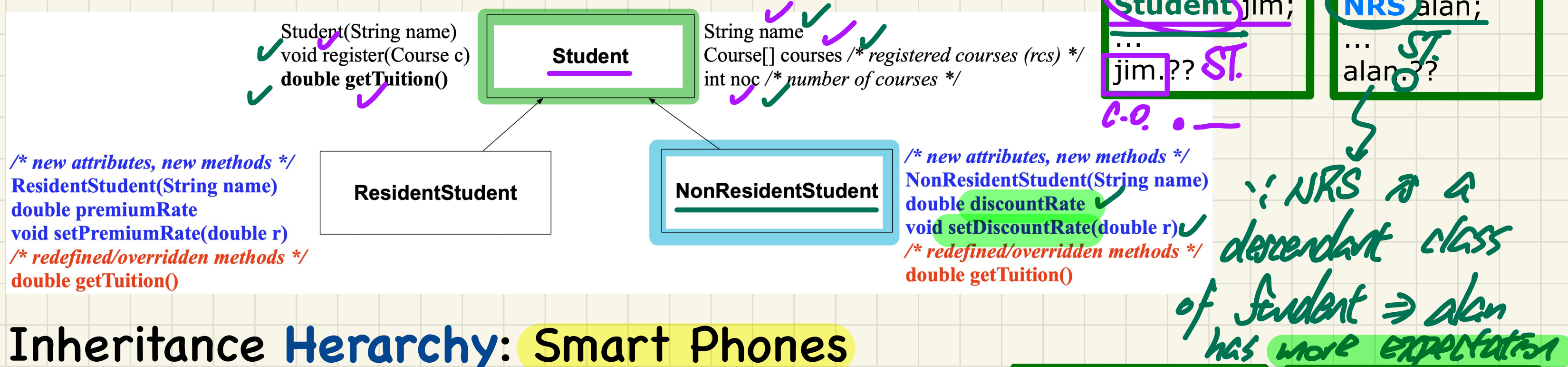
Lecture 5

Part H

Inheritance -
Static Types and Rules of Substitutions

Static Types determine Expectations

Inheritance Herarchy: Students



Inheritance Herarchy: Smart Phones

SmartPhone

dial /* basic method */

surfWeb /* basic method */

IOS

surfWeb /* overridden using safari */

facetime /* new method */

Android

surfWeb /* overridden using firefox */

skype /* new method */

IPhoneSE

/* cinematic mode */

quickTake

IPhone13Pro

Huawei

Samsung

sideSync /* new method */

/* dual-matrix camera */

zoomage

HuaweiP50Pro

HuaweiMate40Pro

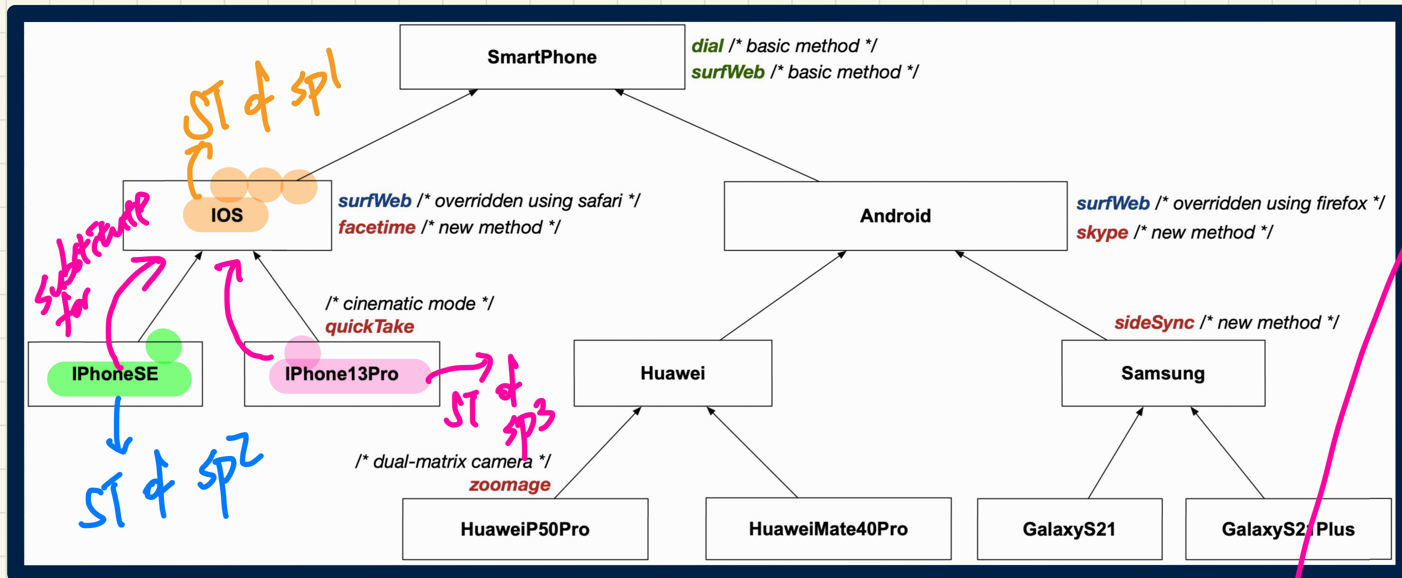
GalaxyS21

GalaxyS21Plus

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

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

Rules of Substitutions (1)



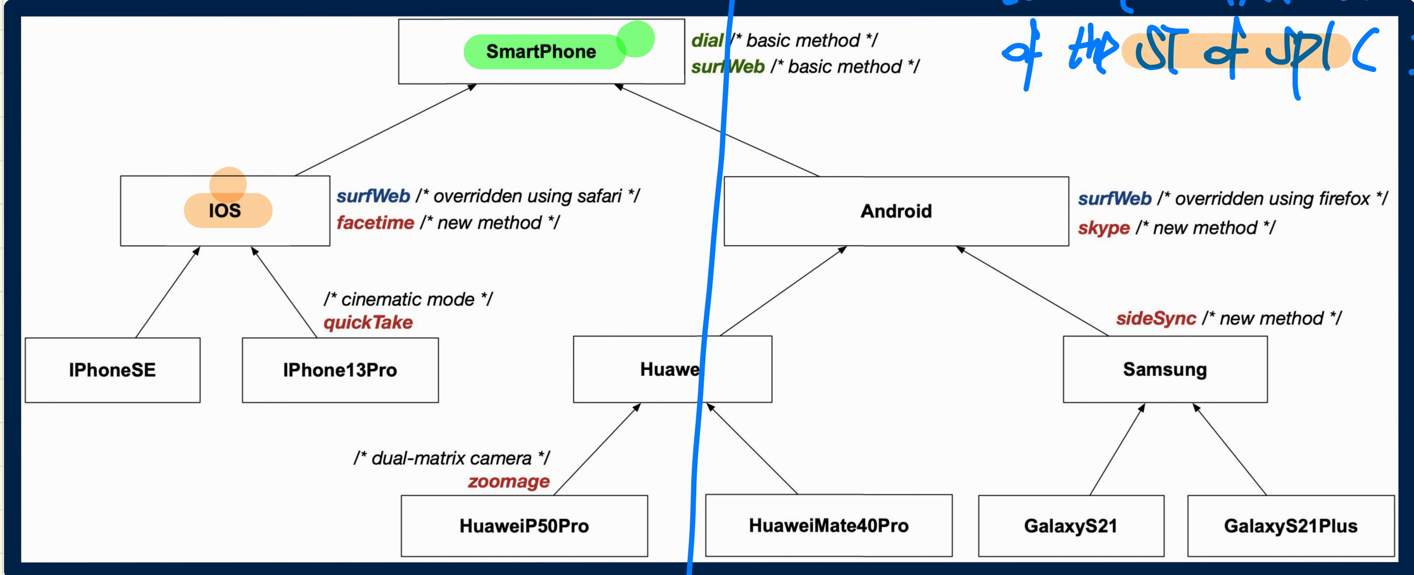
Declarations:

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

Substitutions:

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

Rules of Substitutions (2)



No. ∵ ST of sp2 (SmartPhone) cannot fulfill the exp. of the ST of sp1 (IOS).

Declarations:

IOS sp1;

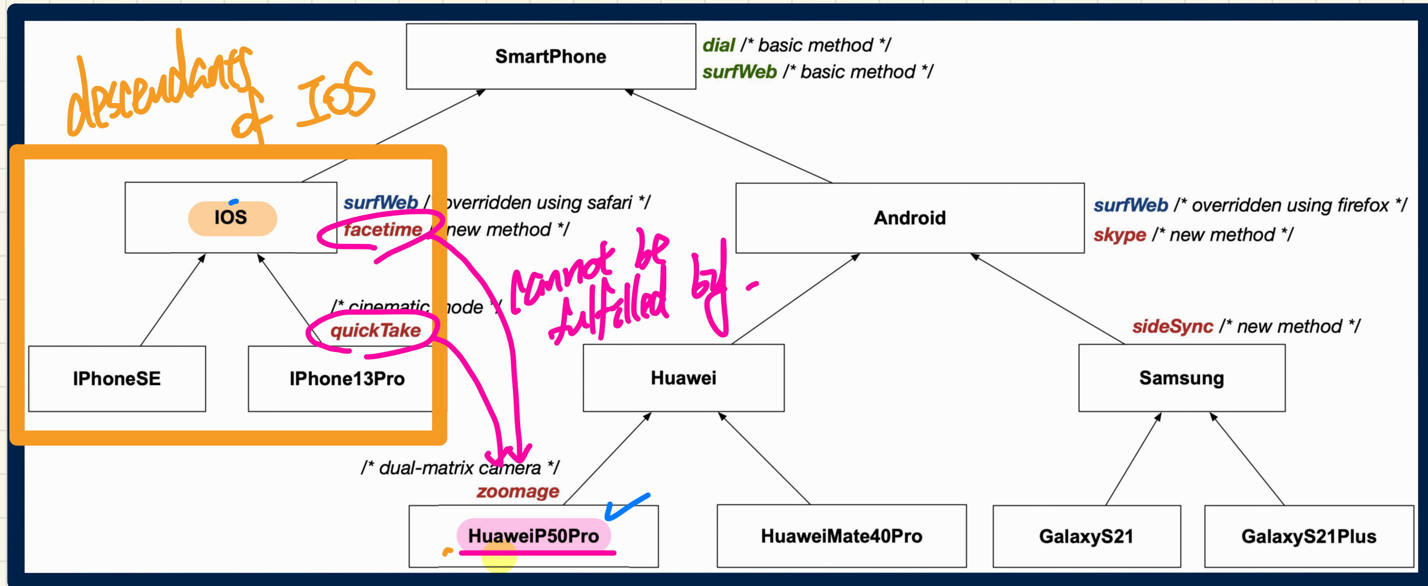
SmartPhone sp2;

Substitutions:

sp1 = sp2;

Can sp2 substitute for sp1

Rules of Substitutions (3)



Declarations:

IOS sp1;
HuaweiP50Pro sp2;

Substitutions:

sp1 = sp2;

Can HuaweiP50Pro fulfill exp. of IOS?
No. ∵ HP50Pro is not a descendant of IOS.

Substitutions

safe vs. **unsafe**

compiles — fails to compile

$$v1 = v2$$

Can v2 substitute v1?

Can the ST of v2 fulfill exp. of v1's ST?

Yes if v2's ST is a descendant of v1's ST.

Lecture 5

Part I

Inheritance -
Dynamic Types,
Polymorphism, Dynamic Binding

Visualization: Static Type vs. Dynamic Type

Declaration:

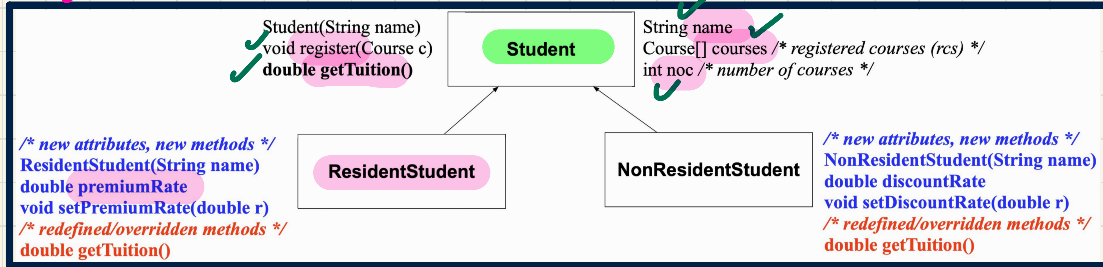
Student s;

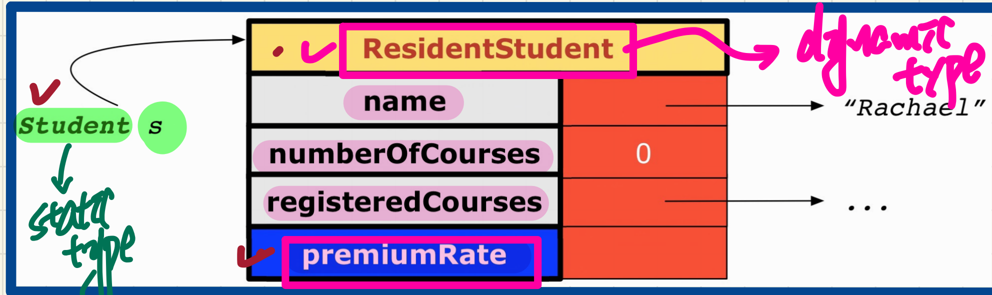
Substitution:

s = new ResidentStudent("Rachael");

Static Type: Expectation

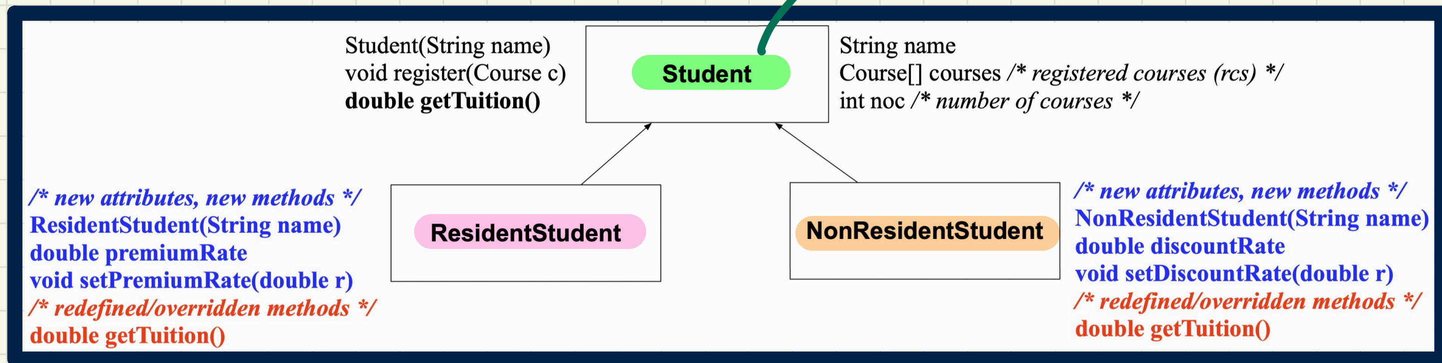
Dynamic Type: Accumulation of Code





Change of Dynamic Type (1.1)

static type of jim



Example 1:

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

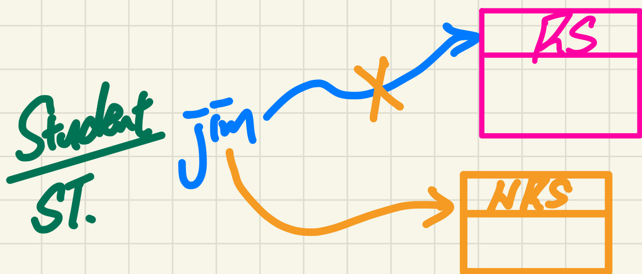
DT.

DT of S: RS

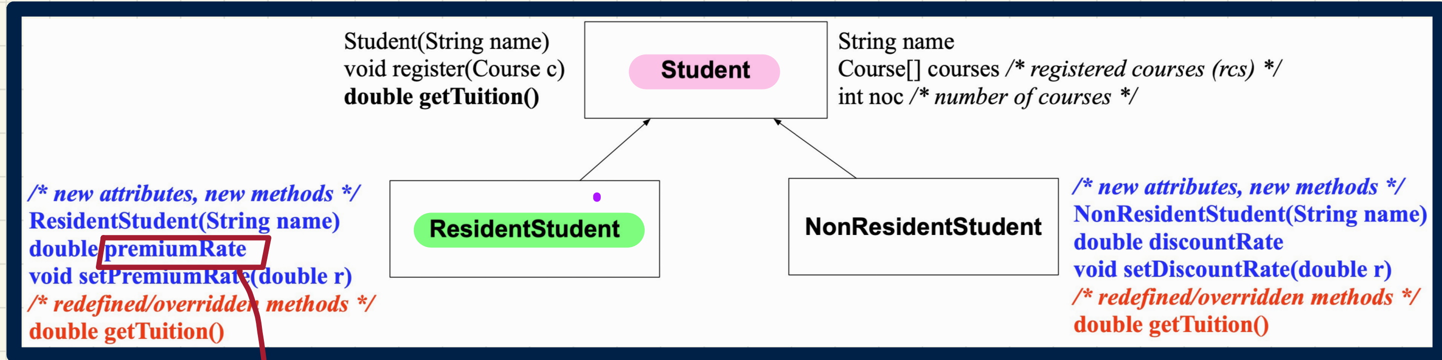
can fulfill exp. of Student ∵ RS is a descendant

DT of S: NRS

can fulfill exp of Student ∵ NRS is a descendant.



Change of Dynamic Type (1.2)



Example 2:

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

ST

proposed dynamic type

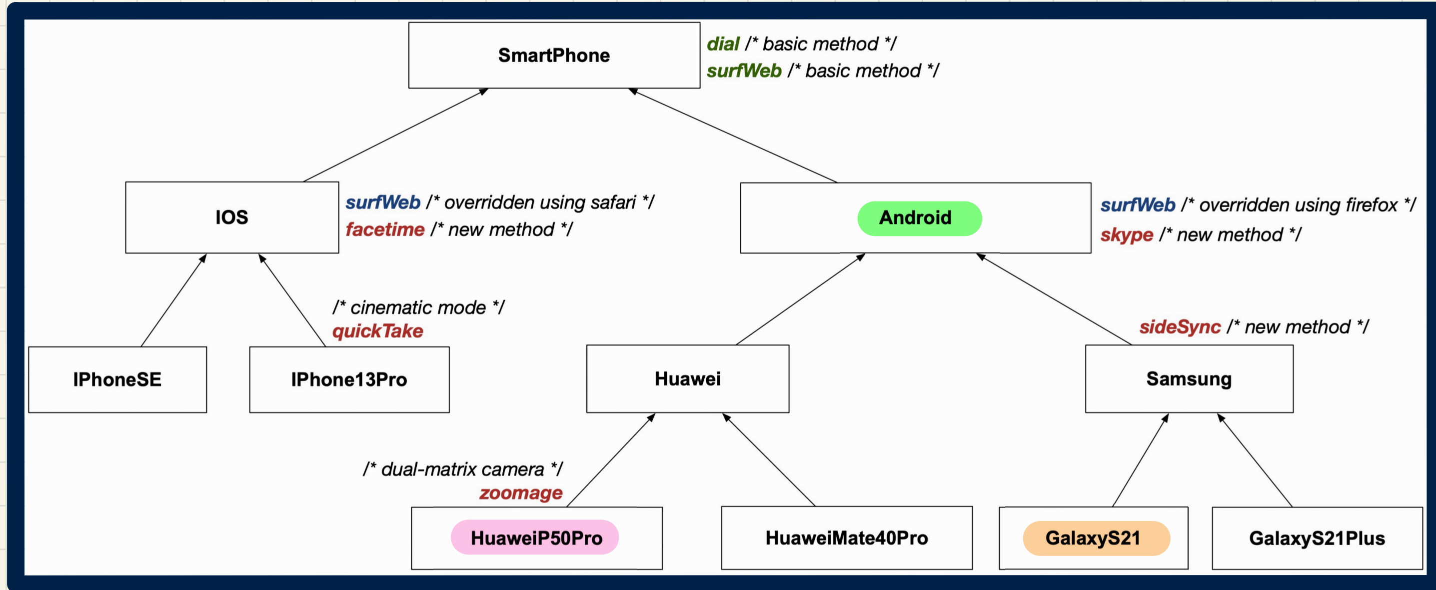
Can Student fulfill exp. of jeremy's ST (RS)?

No! ∵ Student is not a descendant of RS.

Proof by Contra.

If valid, the Student obj. does not support pr expected on Student → invalid

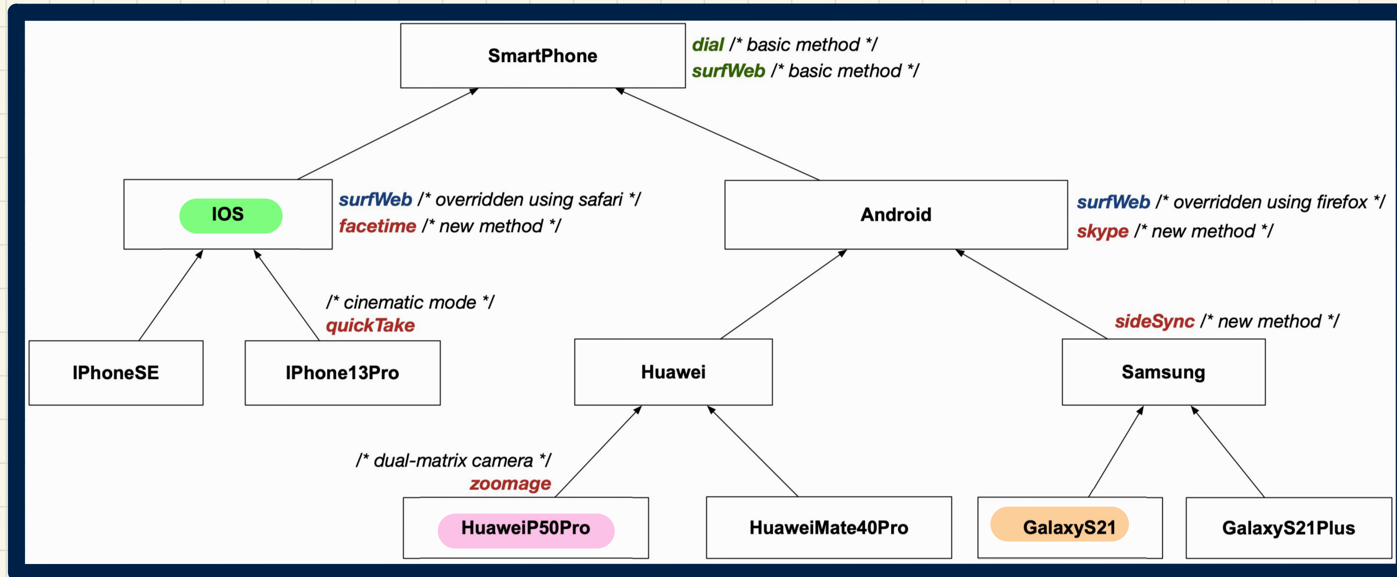
Change of Dynamic Type: Exercise (1)



Exercise 1:

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

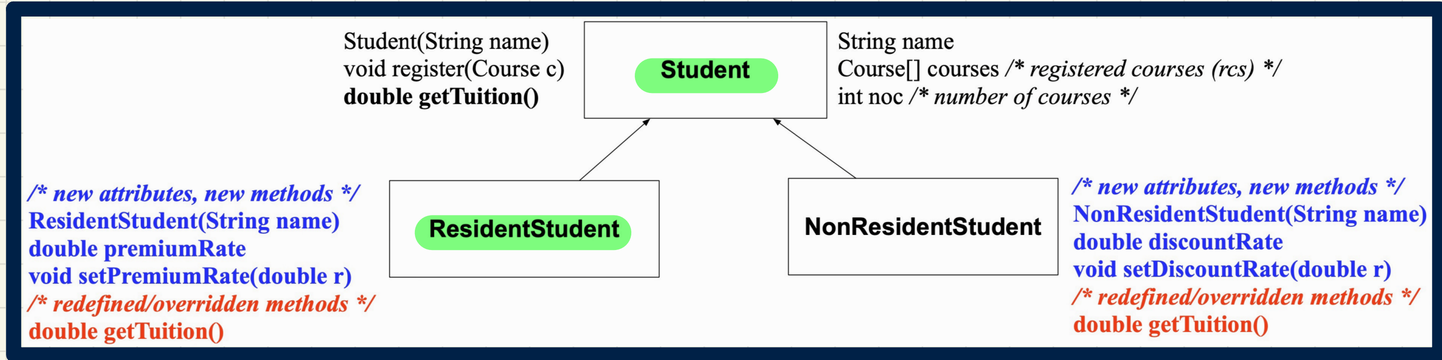
Change of Dynamic Type: Exercise (2)



Exercise 2:

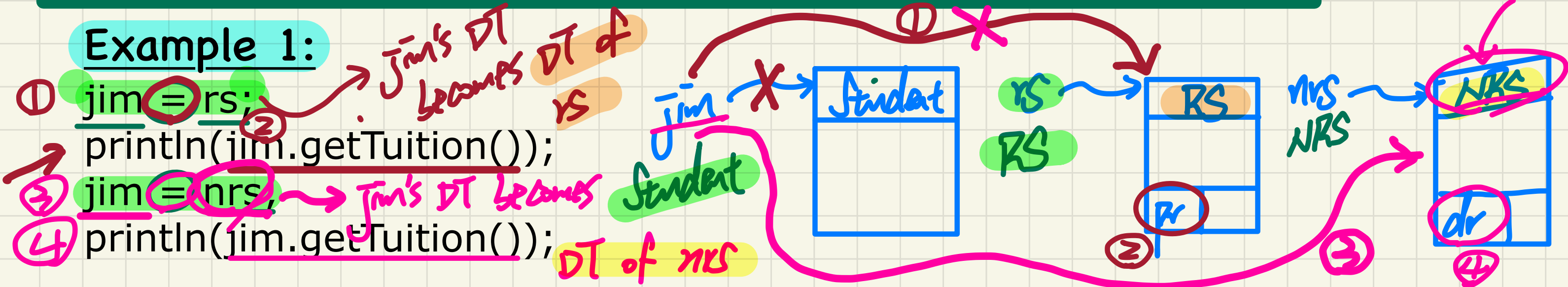
```
IOS myPhone = new HuaweiP50Pro(...);
myPhone = new GalaxyS21(...);
```

Change of Dynamic Type (2.1)

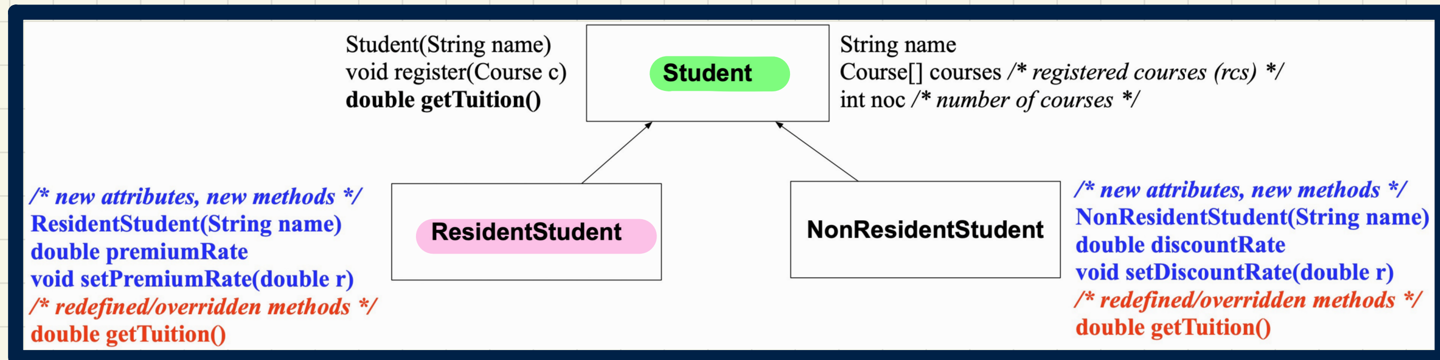


Given:

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



Change of Dynamic Type (2.2)



Given:

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

Example 2:

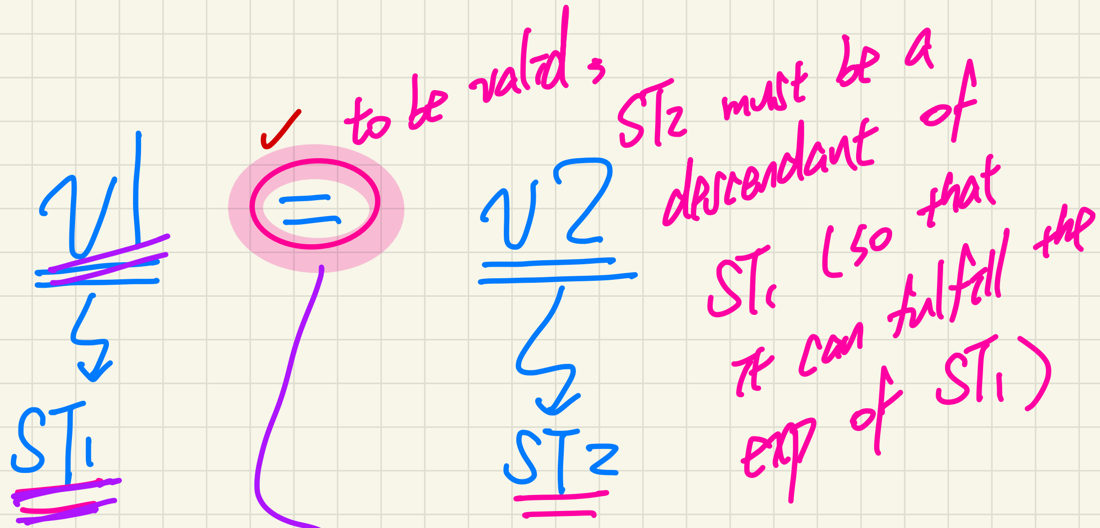
```
rs = jim;
println(rs.getTuition());
nrs = jim;
println(nrs.getTuition());
```

invalid ⇒ fail to compile!

ST: Student

ST: RS

Can jim's ST fulfill exp. of the ST of rs?



Polymorphism: allowable dynamic types (descendants)

dynamic binding

v1. m(...)
↳ each descendant class of ST1 may have its own version

which version to invoke depends on DT of v1.

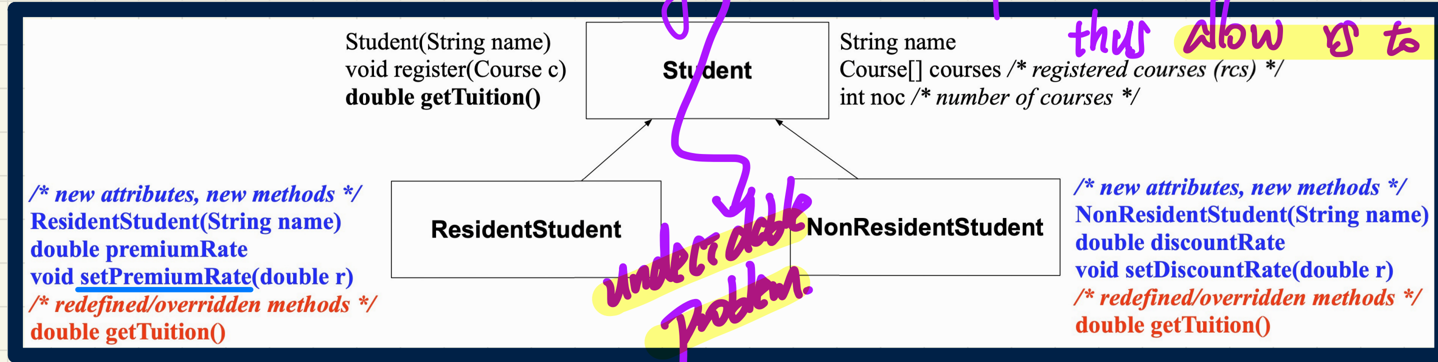
Lecture 5

Part J

Inheritance -
Type Casting: Motivation, Syntax, Rules

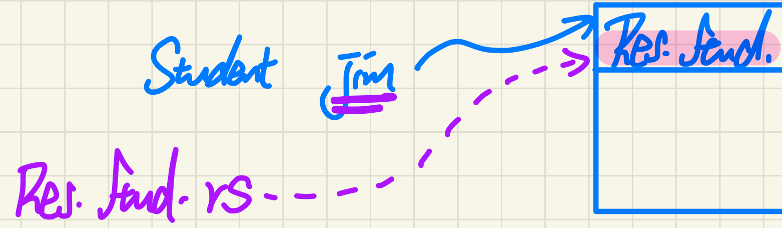
Type Cast: Motivation

But, given that jim's DT is Res. Stud., why can't the compiler know about this and thus allow rs to point to that object?

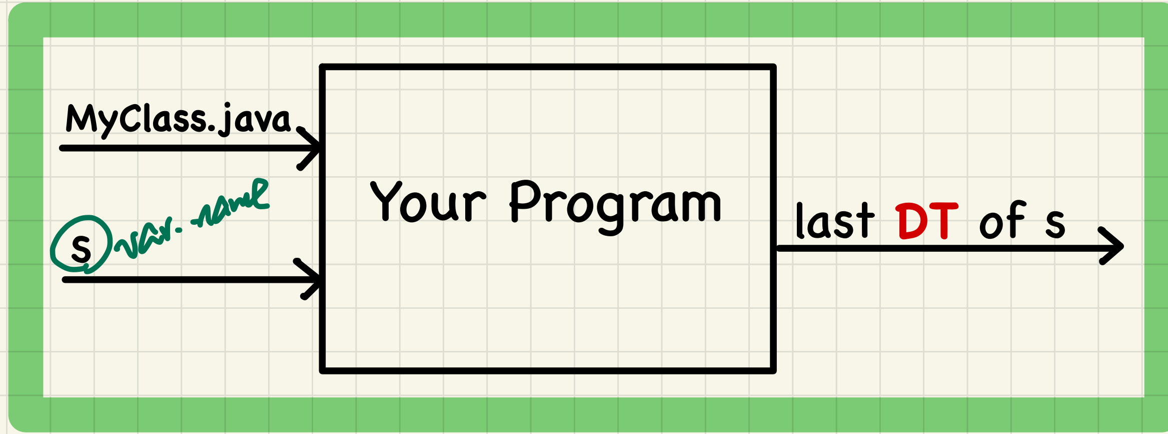



```
1  Student jim = new ResidentStudent("J. Davis");
2  ResidentStudent rs = jim;
3  rs.setPremiumRate(1.5);
```

∵ ST of jim (Student) cannot fulfill exp. of ST of rs (Res. Stud.)



An A+ Challenge: Inferring the DT of a Variable



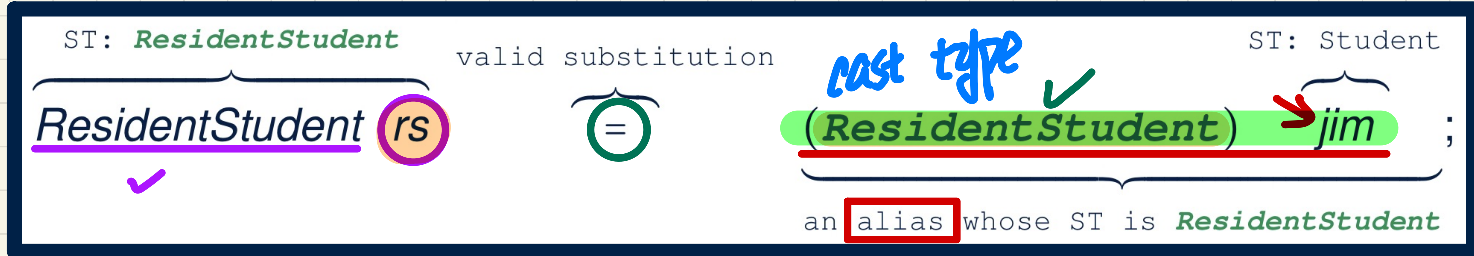

```
class MyClass {
  main (...)
    Student s = ...;
    ...
    s = new ResidentStudent(...);
  }
}
```

while(??) {
}

output: Res. Stud.

Anatomy of a Type Cast

Student jim = **new** **ResidentStudent**("Jim");



Res. Stud. rS

Student jim

ResidentStudent

pr

Res. Stu. (RS) jim

Expectation

jim.   ((Res. Stud) jim).   rS.

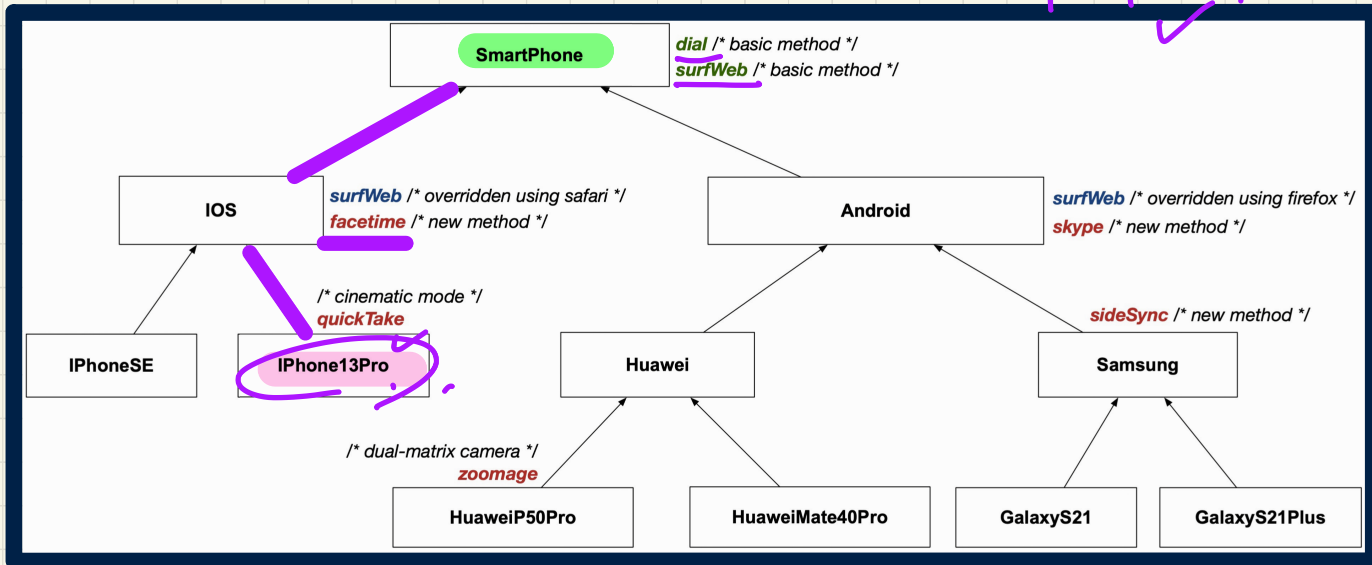
name
courses
noc
register( )
getTuition( )

pr
getPr

Type Cast: Named vs. Anonymous

((IPhone13Pro) aPhone).facetime ✓

↳ ST: IPhone13Pro ✓



Named Cast: Use intermediate variable to store the cast result.

```
SmartPhone aPhone = new IPhone13Pro();
IOS forHeeyeon = (IPhone13Pro) aPhone;
forHeeyeon.facetime();
```

anonymous cast

Anonymous Cast: Use the cast result directly.

```
SmartPhone aPhone = new IPhone13Pro();
((IPhone13Pro) aPhone).facetime();
```

↳ anonymous cast.

Exercise

```
SmartPhone aPhone = new IPhone13Pro();
(IPhone13Pro) aPhone.facetime();
```

✗

expectation of aPhone's ST?

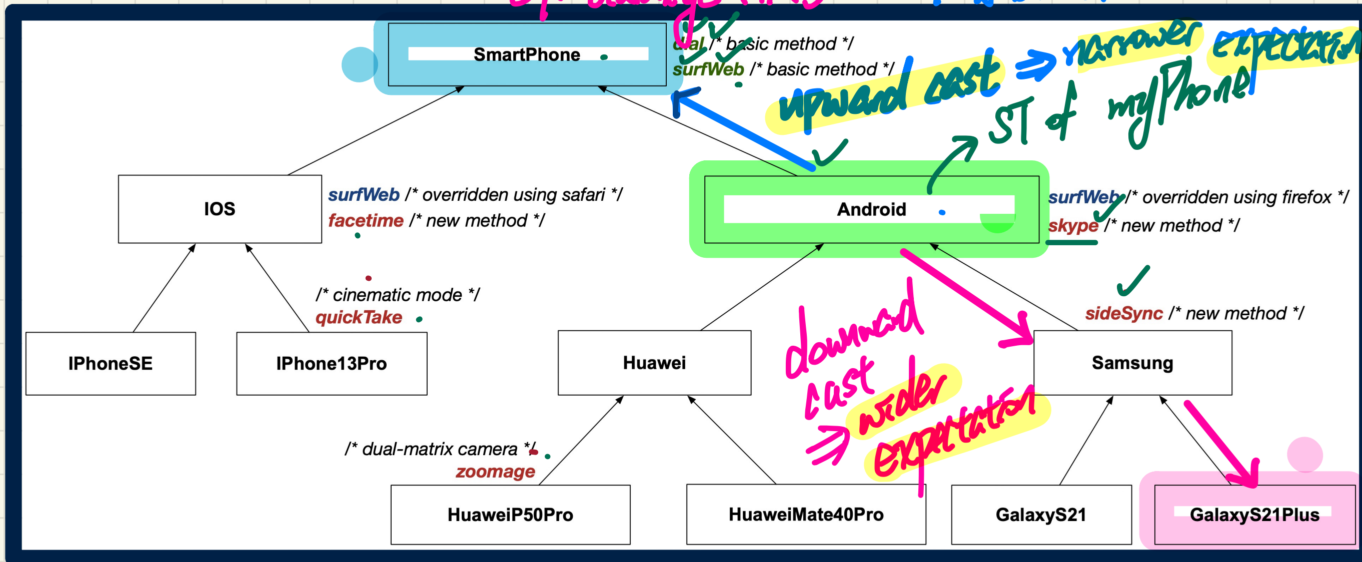
↓ this call first, then cast later

Compilable Casts: Upwards vs. Downwards

```
Android myPhone = new GalaxyS21Plus();
SmartPhone sp = (SmartPhone) myPhone;
GalaxyS21Plus ga = (GalaxyS21Plus) myPhone;
```

Expectations

	sp	myPhone	ga
dial	✓	✓	✓
surfWeb	✓	✓	✓
skype	✗	✓	✓
sideSync	✗	✗	✓
facetime	✗	✗	✗
quickTake	✗	✗	✗
zoomage	✗	✗	✗



Compilable Type Cast May Fail at Runtime (1)

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

Student

String name
Course[] courses /* *registered courses (rcs)* */
int noc /* *number of courses* */

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

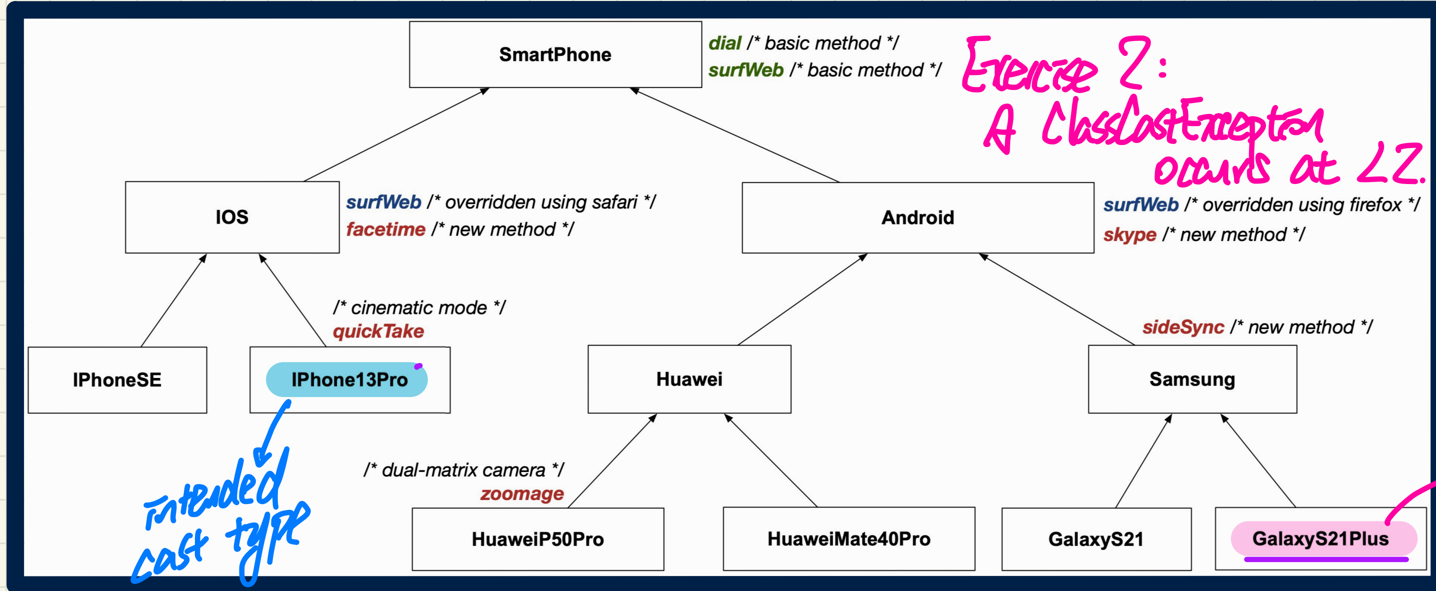
ResidentStudent

NonResidentStudent

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

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

Compilable Type Cast May Fail at Runtime (2)

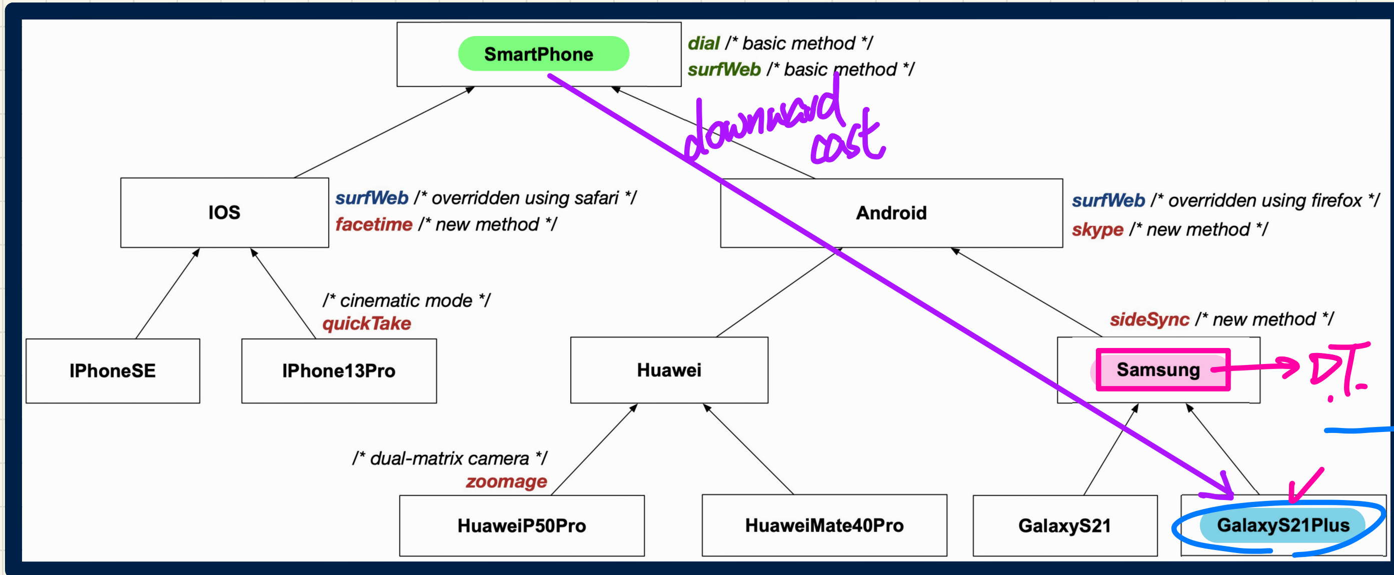



```
1  SmartPhone aPhone = new GalaxyS21Plus();
2  IPhone13Pro forHeeyeon = (IPhone13Pro) aPhone;
3  forHeeyeon.quickTake();
```

→ valid only if the variable's DT can fulfil the exp. of the cast type.

Exercise 1: Explain why L1, L2, L3 compile.

Exercise: Compilable Type Cast? Fail at Runtime? (1)



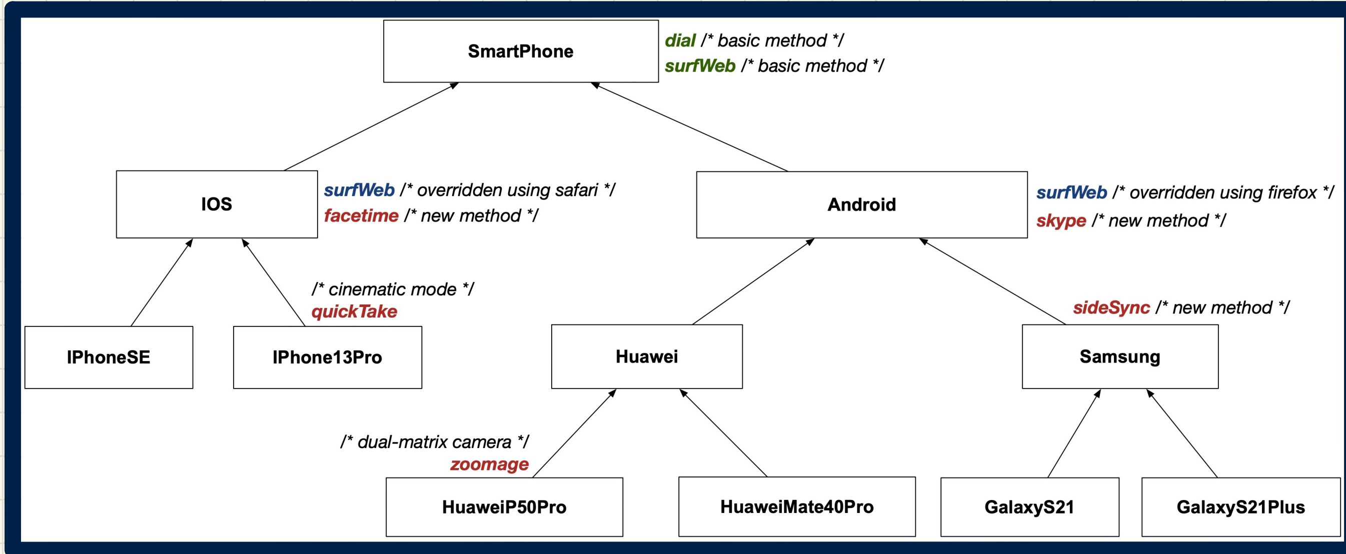

```
SmartPhone myPhone = new Samsung();
/* ST of myPhone is SmartPhone; DT of myPhone is Samsung */
GalaxyS21Plus ga = (GalaxyS21Plus) myPhone;
```

downward cast

Compilable? **ClassCastException at runtime?**

Runtime: Can the DT of myPhone (Samsung) fulfill exp. of cast type (GS21P)

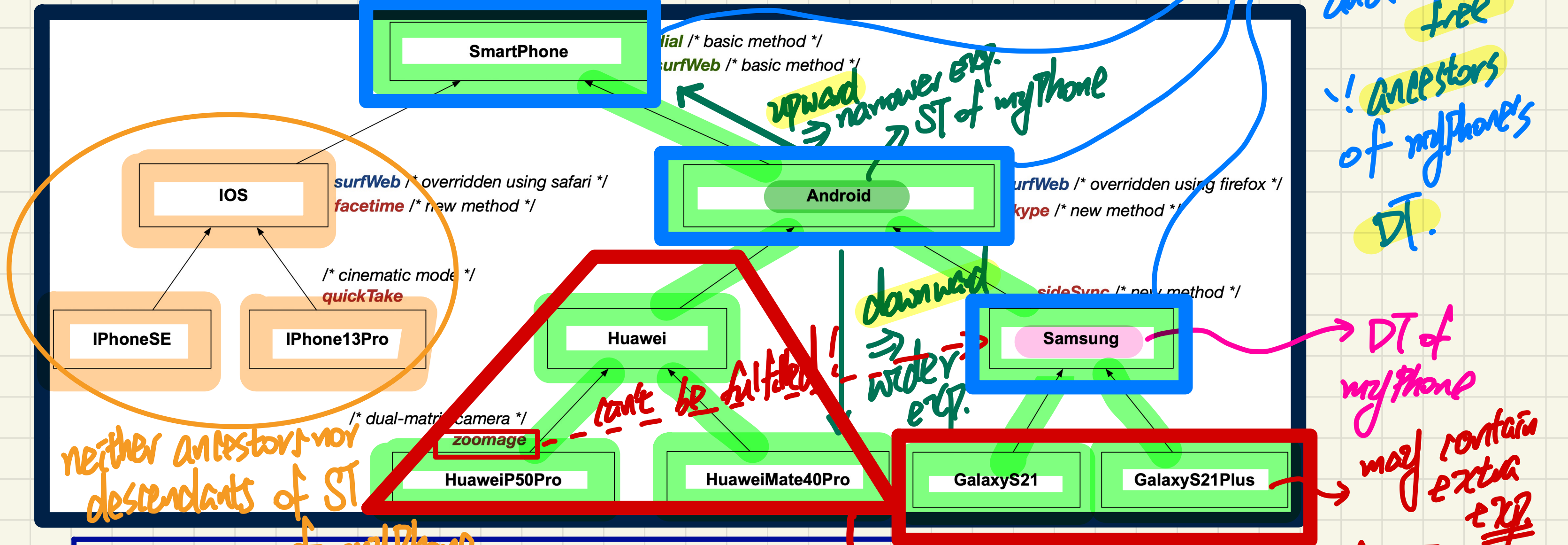
Exercise: Compilable Type Cast? Fail at Runtime? (2)




```
SmartPhone myPhone = new Samsung();
/* ST of myPhone is SmartPhone; DT of myPhone is Samsung */
IPhone13Pro ip = (IPhone13Pro) myPhone;
```

Compilable? ClassCastException at runtime?

Compilable Cast vs. Exception-Free Cast



Android myPhone = new Samsung();

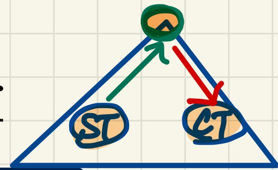
Compilable Casts

Exception-Free Casts

Non-Compilable Casts

ClassCastException

Exercise: Compilable Cast vs. Exception-Free Cast




```
class A { }
class B extends A { }
class C extends B { }
class D extends A { }
```

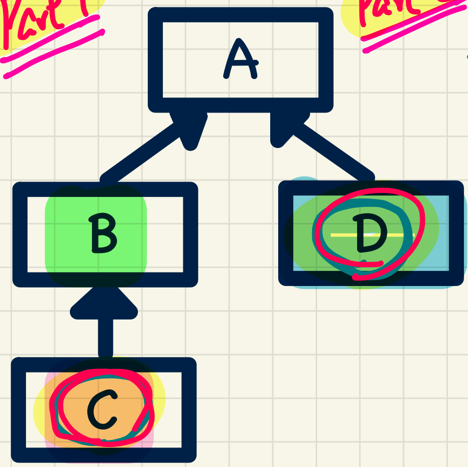
Part 3 Alternative to ⓩ2
D d = (D)((A) b) ✓ valid (compilable?)

e.g.
Floor f = (Floor)((Object) v)

```
1 B b = new C();
2 D d = (D) b;
```

→ valid if it's either upward or downward cast w.r.t. ST B

Part 1



Part 2 Compilation (error at ⓩ2)

ⓩ1 ✓ DT C can fulfill the exp. of ST B (∵ C is a descendant of B)

ⓩ2 ✗ ∵ cast type D is neither an ancestor nor a descendant of b's ST (B)

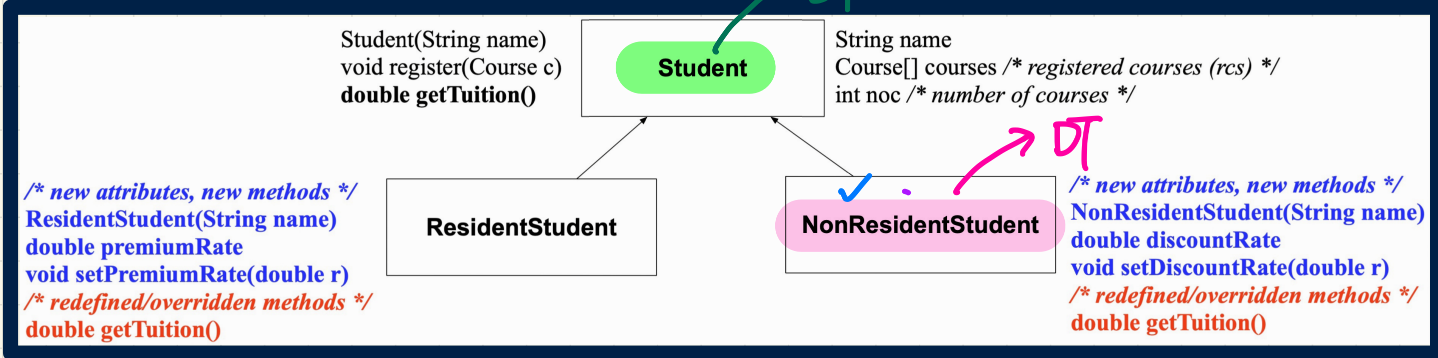
Part 4 ClassCastException
∵ b's DT (C) cannot fulfill exp. of cast type (D)
cast type is not an ancestor of b's DT.

Lecture 5

Part K

Inheritance -
Checking Dynamic Type via instanceof

Checking Dynamic Types at Runtime (1)




```
1  Student jim = new NonResidentStudent("J. Davis");
2  if (jim instanceof ResidentStudent) {
3    ResidentStudent rs = (ResidentStudent) jim;
4    rs.setPremiumRate(1.5);
5  }
```

√3. Is ResidentStudent* an ancestor of DT of jim?

False ∵ RS is not an ancestor of jim's DT

1. Can DT of jim fulfill the expectation of ResidentStudent*?

2. Is DT of jim a descendant of ResidentStudent*?

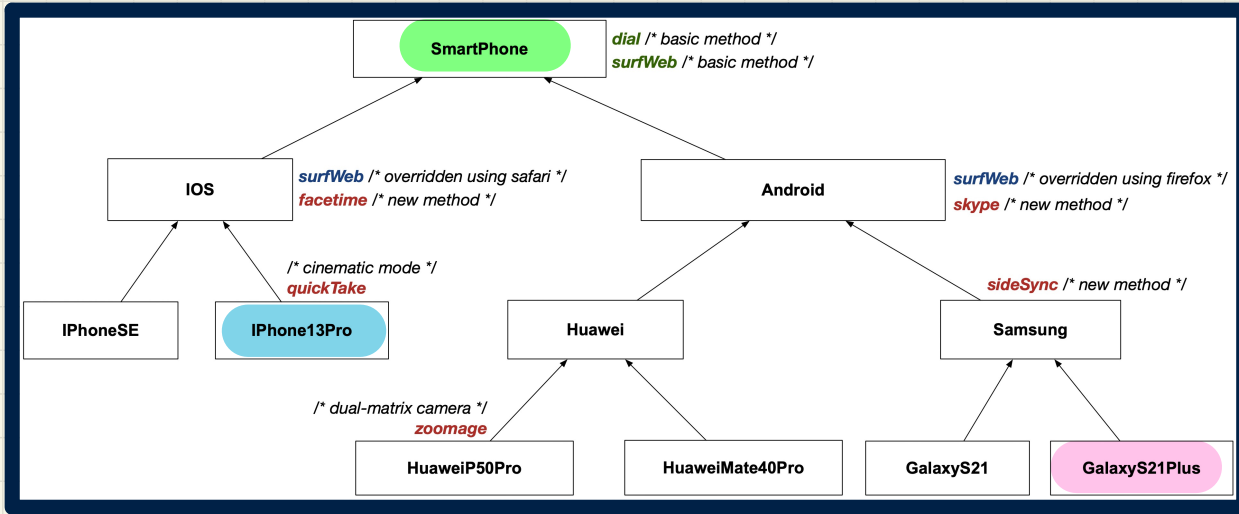
expression (dot notation) denoting an object

checked in the if branch

ClassCastException if executed

a class name

Checking Dynamic Types at Runtime (2)

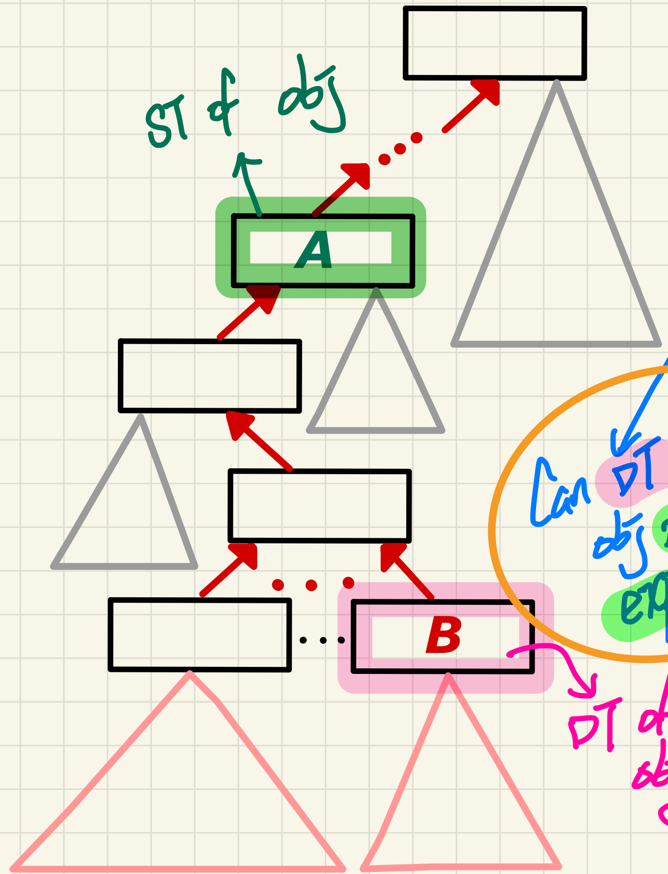



```
1  SmartPhone aPhone = new GalaxyS21Plus();
2  if (aPhone instanceof IPhone13Pro) {
3    IOS forHeeyeon = (IPhone13Pro) aPhone;
4    forHeeyeon.facetime();
5  }
```

Can DT of aPhone fulfill expectations of IPhone13Pro*?

executing that line will cause a ClassCastException.

The instanceof Operator




```
1  A obj = new B();
2  if (obj instanceof ??) {
3    ?? obj2 = (??) obj;
   }
```

meant as a guard constraint to prevent a ClassCastExcep.

might run into CCE

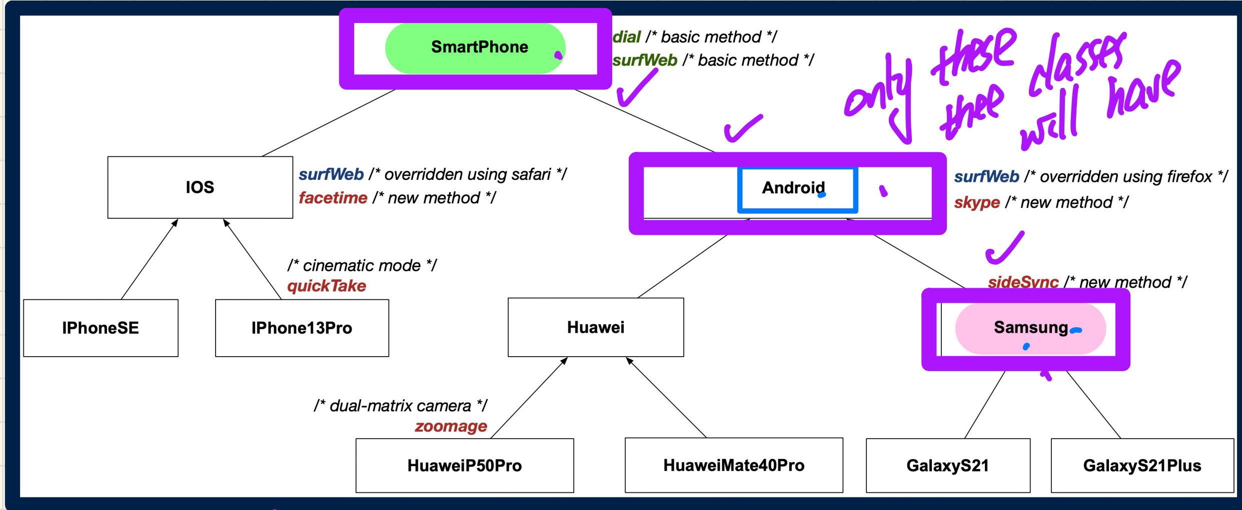
Can DT of obj fulfil exp. of ?? ?

- L1 compiles if B can fulfill expectations of A.

  if DT of obj cannot fulfill exp. of ??

- L3:
  - Compiles if Up or Down cast w.r.t. A.
  - ClassCastException if B cannot fulfill expectations on ??.
- L2:
  - Evaluates to true if B can fulfill expectations on ??.

Use of the instanceof Operator

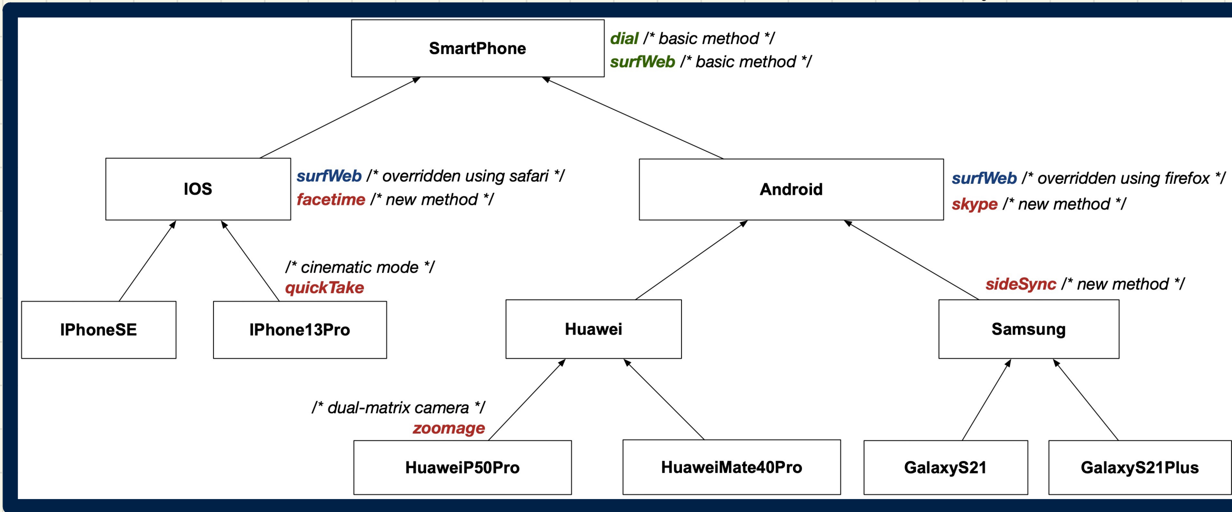



```
SmartPhone myPhone = new Samsung();
println(myPhone instanceof Android);
/* true ∵ Samsung is a descendant of Android */
println(myPhone instanceof Samsung);
/* true ∵ Samsung is a descendant of Samsung */
println(myPhone instanceof GalaxyS21);
/* false ∵ Samsung is not a descendant of GalaxyS21 */
println(myPhone instanceof IOS);
/* false ∵ Samsung is not a descendant of IOS */
println(myPhone instanceof IPhone13Pro);
/* false ∵ Samsung is not a descendant of IPhone13Pro */
```

Can DT of myphone fulfill Android?

myPhone **instanceof** ?? evaluates to **true** if **Samsung** can fulfill expectations on ??.

Safe Cast via Use of the instanceof Operator




```
1  SmartPhone myPhone = new Samsung();
2  /* ST of myPhone is SmartPhone; DT of myPhone is Samsung */
3  if(myPhone instanceof Samsung) {
4    Samsung samsung = (Samsung) myPhone;
5  }
6  if(myPhone instanceof GalaxyS21Plus) {
7    GalaxyS21Plus galaxy = (GalaxyS21Plus) myPhone;
8  }
9  if(myphone instanceof HuaweiMate40Pro) {
10   Huawei hw = (HuaweiMate40Pro) myPhone;
11 }
```

myPhone instanceof ??
evaluates to true if
Samsung can
fulfill expectations on ??.