# Lecture 19 - Nov 13

## Inheritance

***Type Casts: Motivation, Syntax, Visual***
***Compilable Casts: Upward vs. Downward***
***Casts at Runtime: ClassCastException***

## Announcements/Reminders

- Today's class: notes template posted
- Lab4 released
- WrittenTest2 guide and example questions released
- Notes on Type Casting

many shapes

# Polymorphism and Dynamic Binding

T1 / ST — v = new T2();  DT.

T2 must fulfill the exp(T1)

**Polymorphism**:

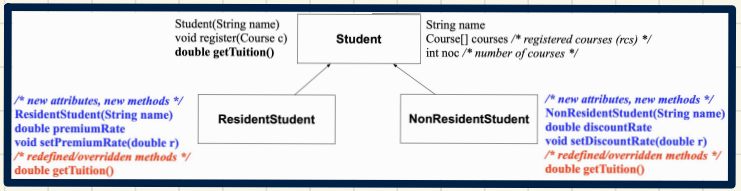
An object's **static type** may allow **multiple** possible **dynamic types**.

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

**Dynamic Binding**:

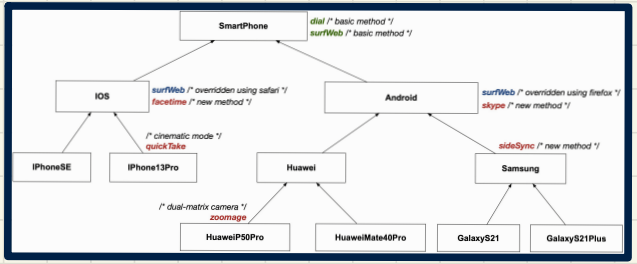
An object's **dynamic type** determines the **version** of method being invoked.

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

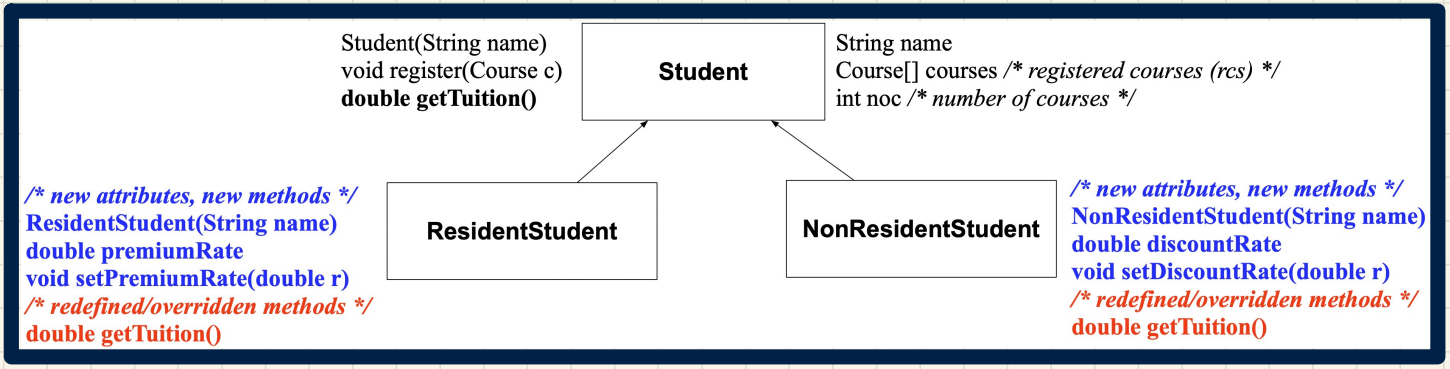



```
SmartPhone sp1 = new IPhone13Pro(...);
SmartPhone sp2 = new GalaxyS21(...);
sp1.surfWeb();
sp1 = sp2;
sp1.surfWeb();
```

after this substitution, DT of sp1 changes to DT of sp2.

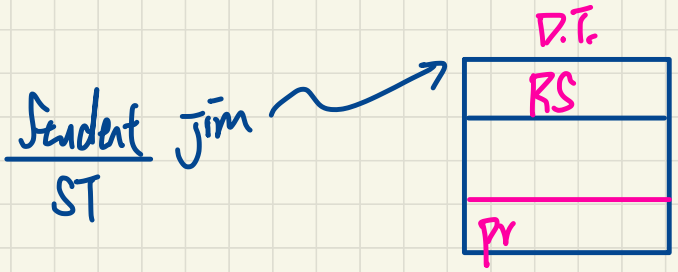


# Type Cast: Motivation

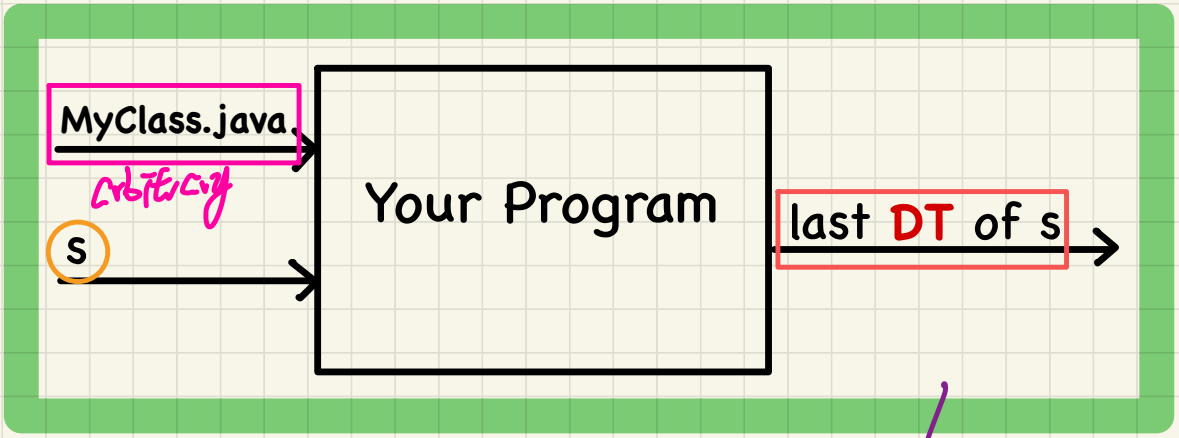



```
1  Student jim = new ResidentStudent("J. Davis");
2  ResidentStudent rs = jim;
3  rs.setPremiumRate(1.5);
```

ST: Student → not compile ∵ Student (jim's ST) is not descendant of RS.



# An A+ Challenge: Inferring the DT of a Variable




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

last DT of s

undecidable problem
(writing a program to infer the dynamic behavior of another arbitrary program)

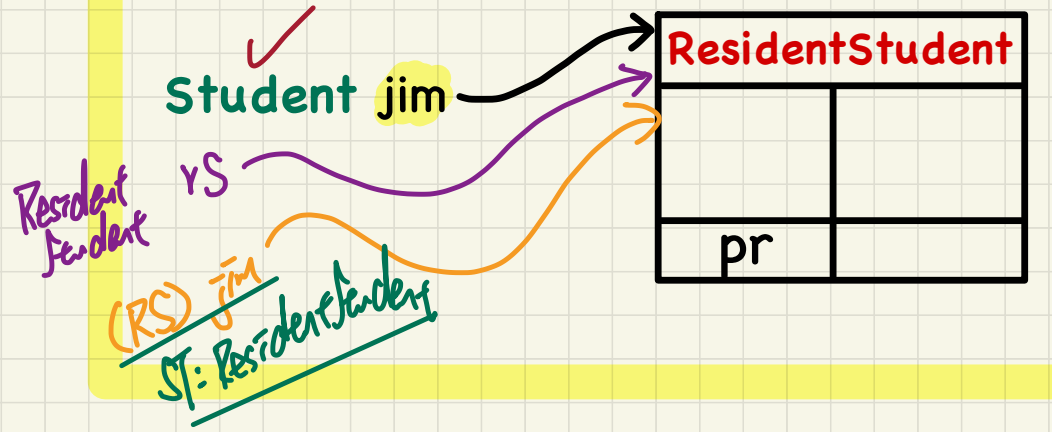
# Anatomy of a Type Cast

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

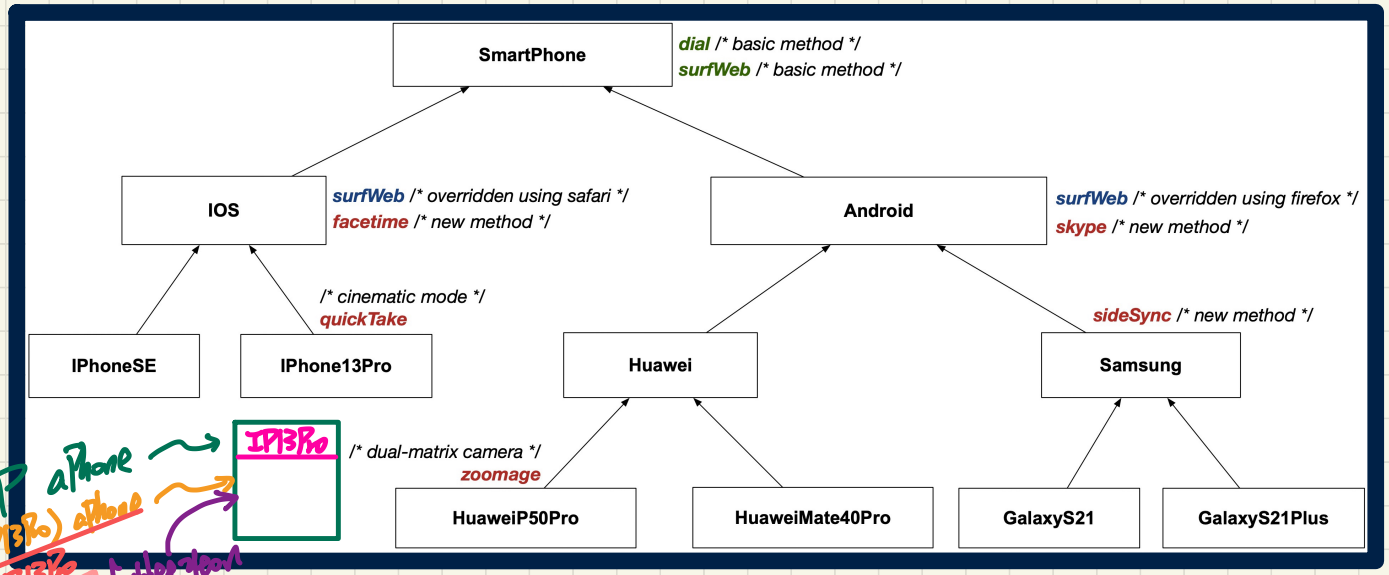

ST: ResidentStudent

jim.setPr X

rs.setPr. ✓



# Type Cast: Named vs. Anonymous




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

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

**Anonymous Cast**: Use the cast result directly.

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

## Exercise

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

# Compilable Casts: Upwards vs. Downwards

static type consideration only

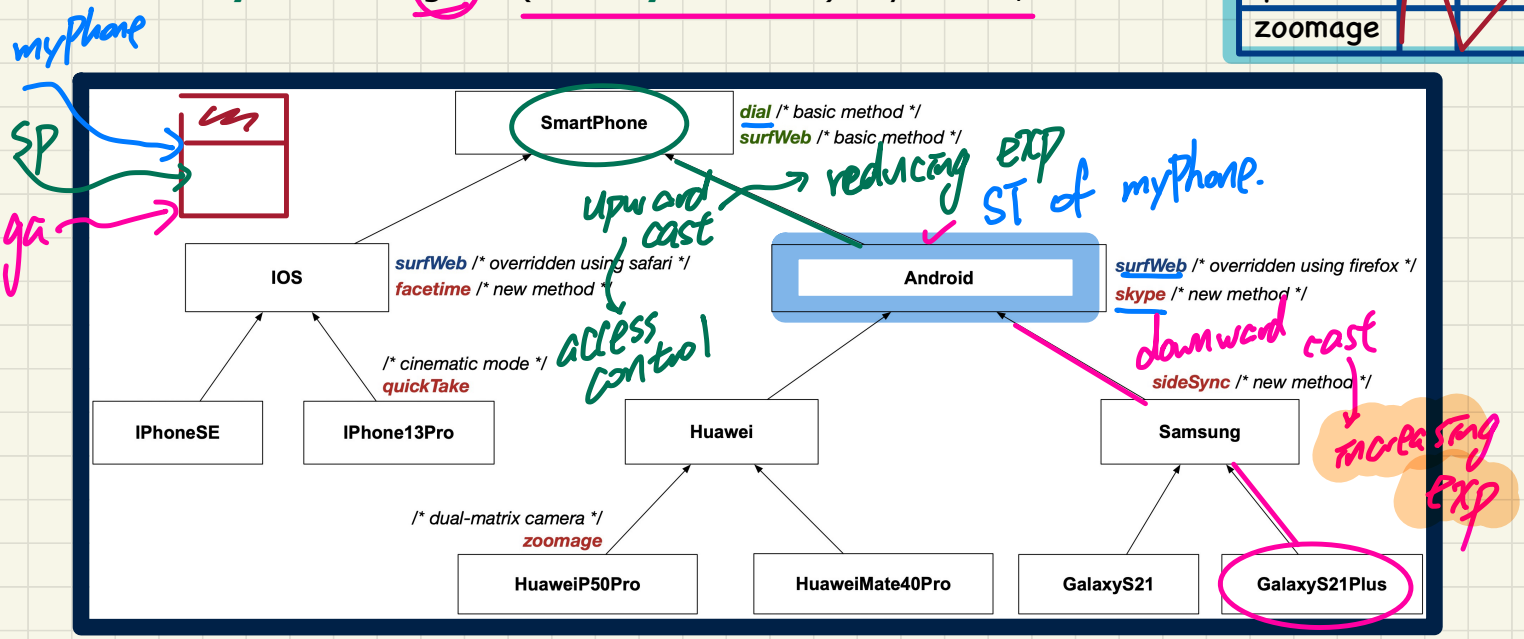
```
Android myPhone = new GalaxyS21Plus();

SmartPhone sp = (SmartPhone) myPhone;

GalaxyS21Plus ga = (GalaxyS21Plus) myPhone;
```

disregarding DT.

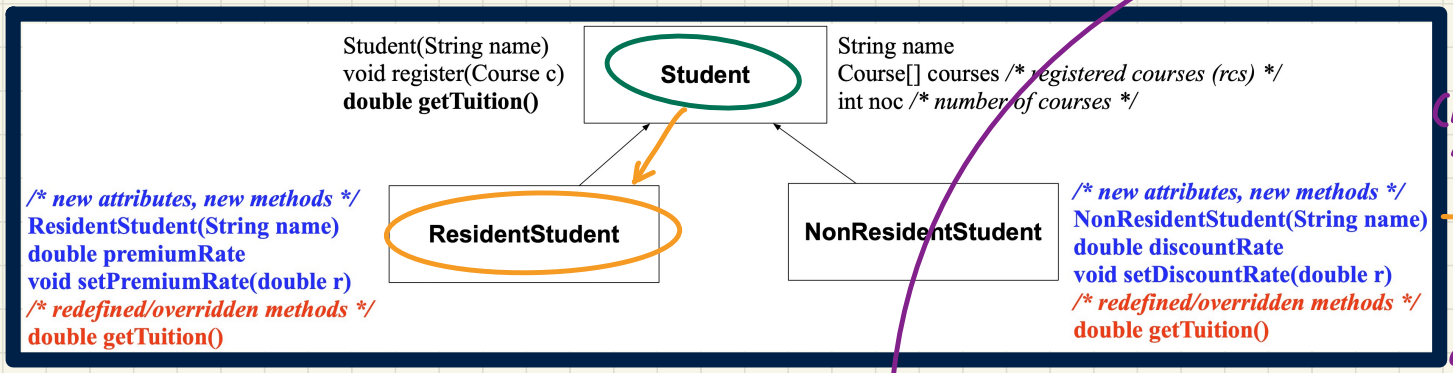
upward cast

## Expectations

| | sp | myPhone | ga |
|---|---|---|---|
| dial | | | |
| surfWeb | | | |
| skype | | | |
| sideSync | | | |
| facetime | | | |
| quickTake | | | |
| zoomage | | | |



# Compilable Type Cast May Fail at Runtime (1)




trigger: ClassCastException (CCE)

(1) when the e.g. NRS DT cannot fulfil the exp. of cast type. e.g. RS

(2) when the DT is not a descendant of cast type.

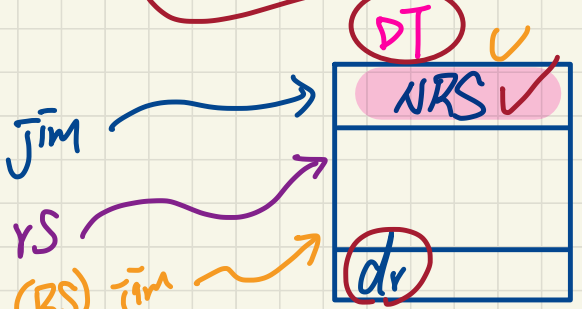
```
1  Student jim = new NonResidentStudent("J. Davis");
2  ResidentStudent rs = (ResidentStudent) jim;
3  rs.setPremiumRate(1.5);
```

downward cast

exp (RS) → pr.

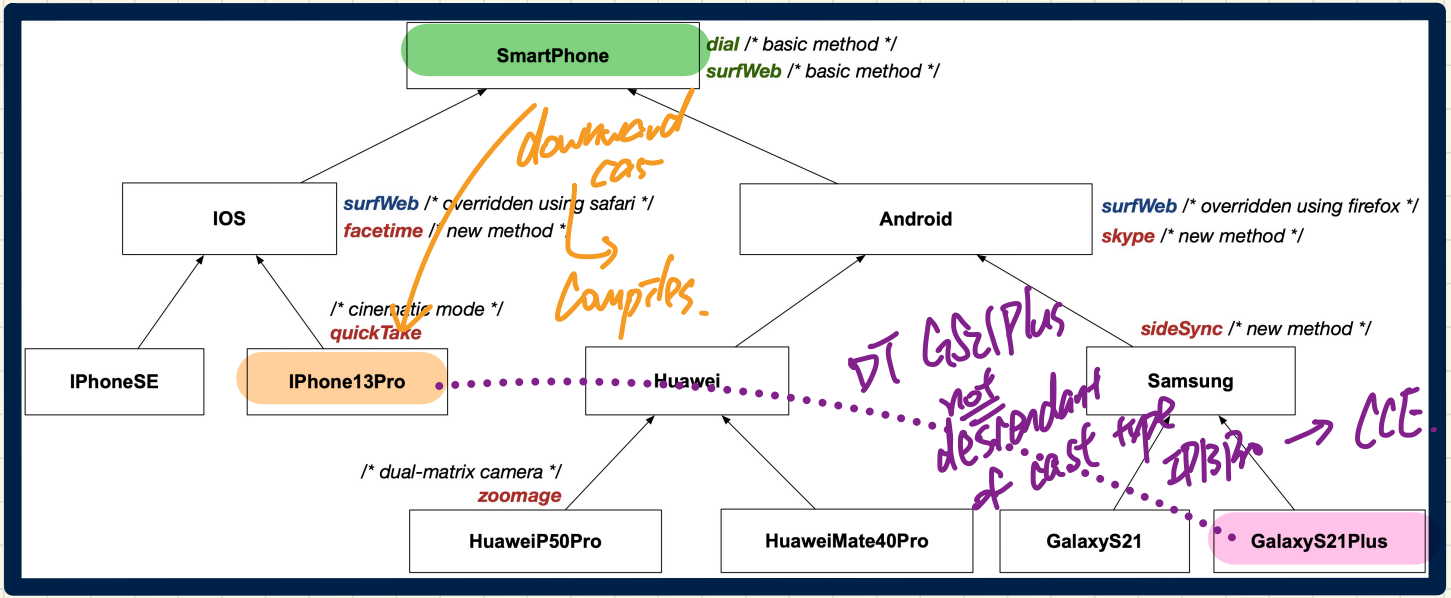
ST: Student

RS



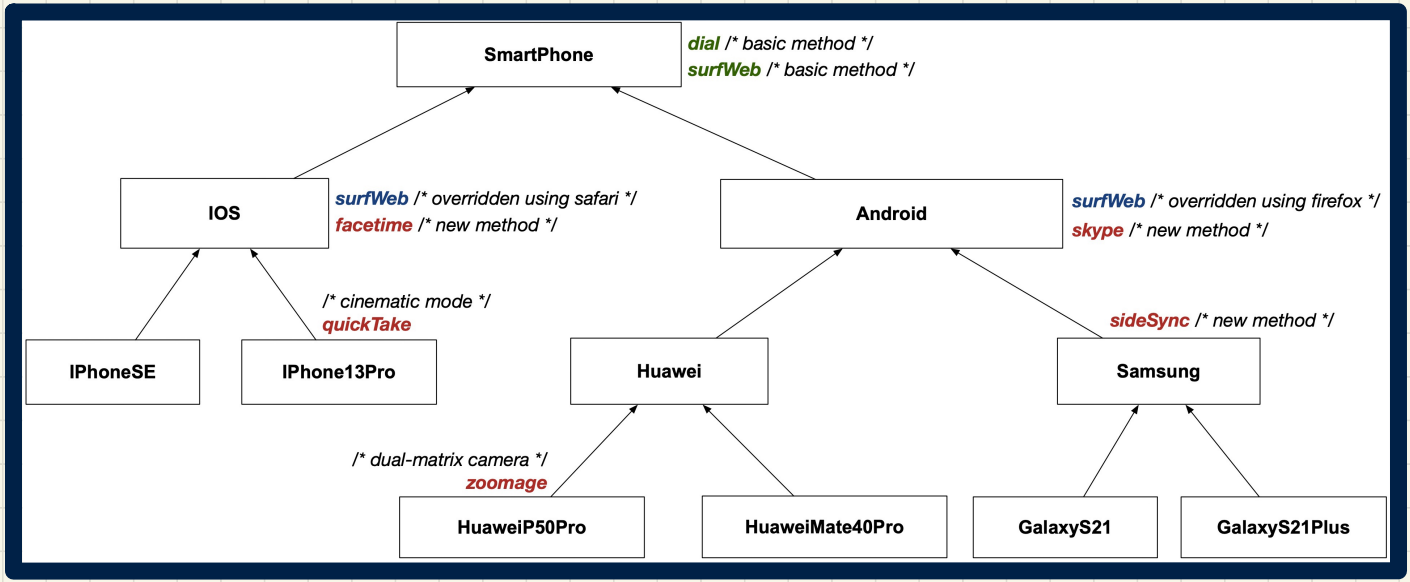

issue: DT NRS cannot fulfil the exp (RS)

# Compilable Type Cast May Fail at Runtime (2)




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

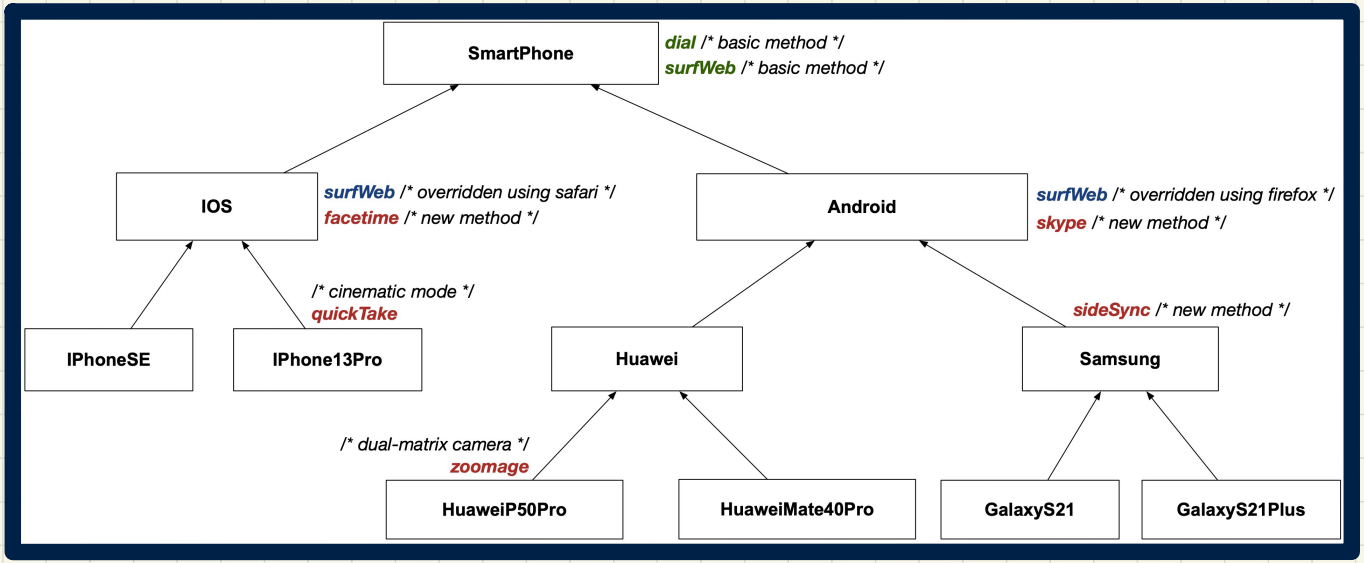
# Exercise: Compilable Type Cast? Fail at Runtime? (1)




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

Compilable? ClassCastException at runtime?

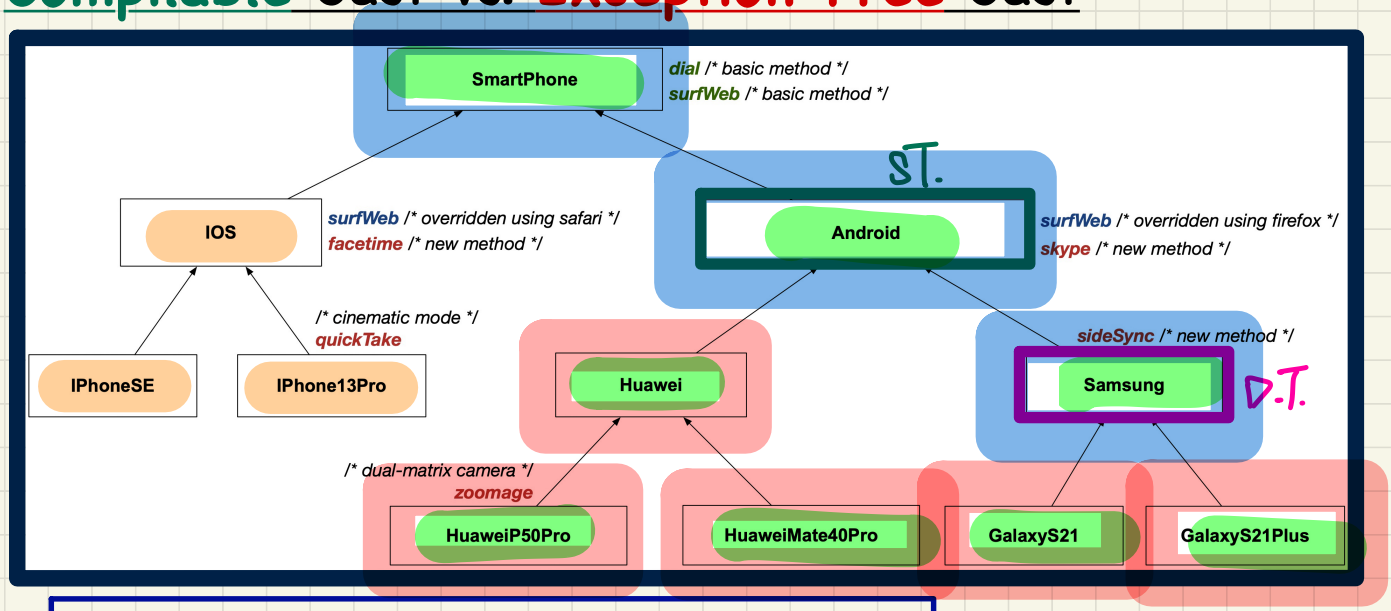
# Exercise: Compilable Type Cast? Fail at Runtime? (2)




```
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

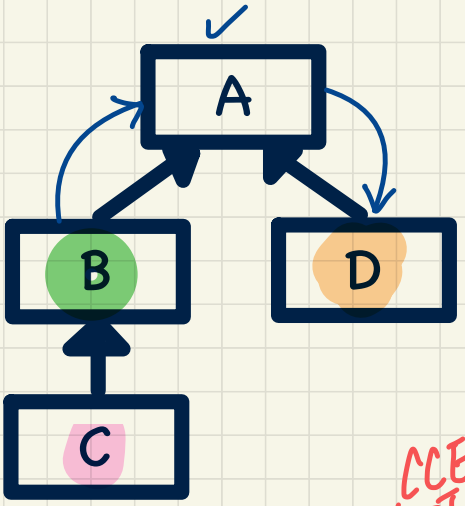
ClassCastException

# Exercise: Compilable Cast vs. Exception-Free Cast

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

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

→ ✗ compile ∵ cast type D is neither ancestor nor descendant of b's ST (B)




D d = (D) ((A) b)

downward cast

no CCE ∵ DT C is descendant of cast type A.

B b

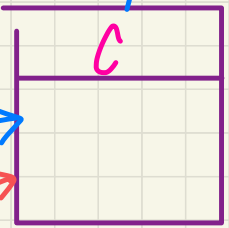
upward cast

ST: A.

CCE ∵ DT C cannot fulfill exp(D) cast type.

(A) b

(D) ((A) b)



Vaccine v = ... ;

Channel c = (Channel)((Object) v)