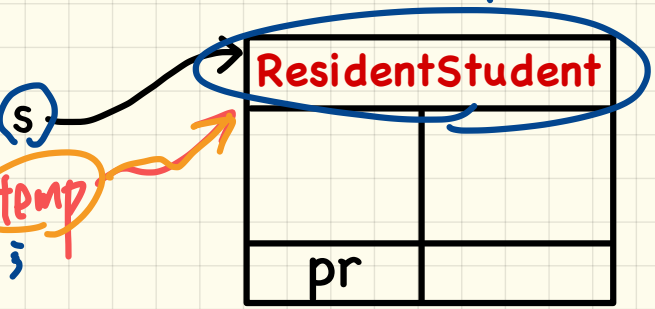
Lecture 19
Wednesday November 13

Anatomy of a Type Cast

```
Student s = new ResidentStudent("Jim");
```

ST — DT



cast version of 's'

temp

ST: Stu.

RS is ST

rs = s;

s: Stu.

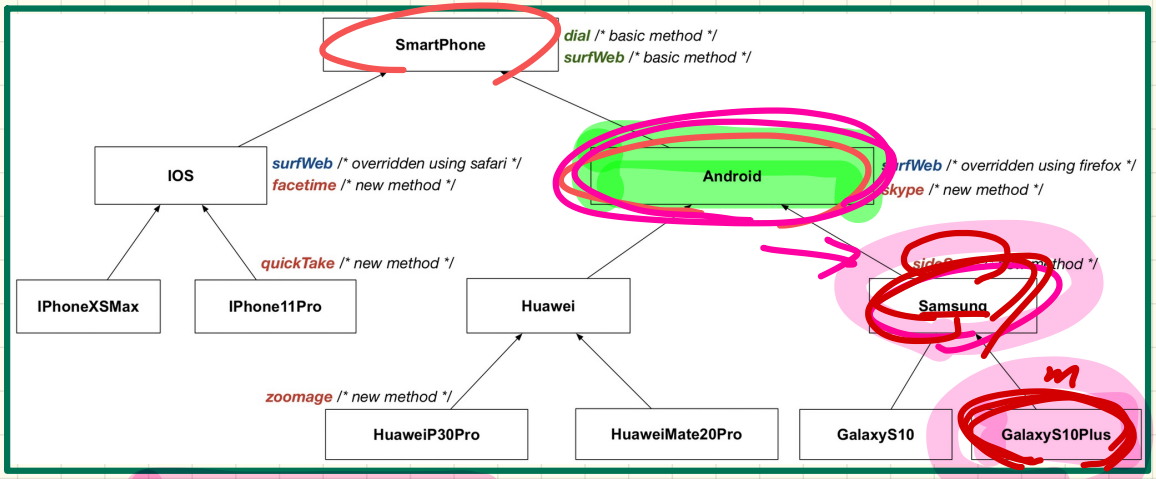
is this valid?

temp with ST RS

```
ResidentStudent rs = (ResidentStudent) s;
```

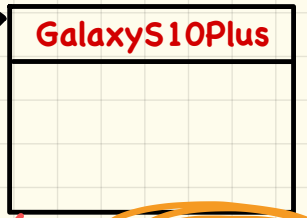
RS is a descendant of RS. ✓ ST RS

is a descendant of Stu. ; down casting.




```
Android s = new GalaxyS10Plus();
```

Android s



Substitutions of Type Cast

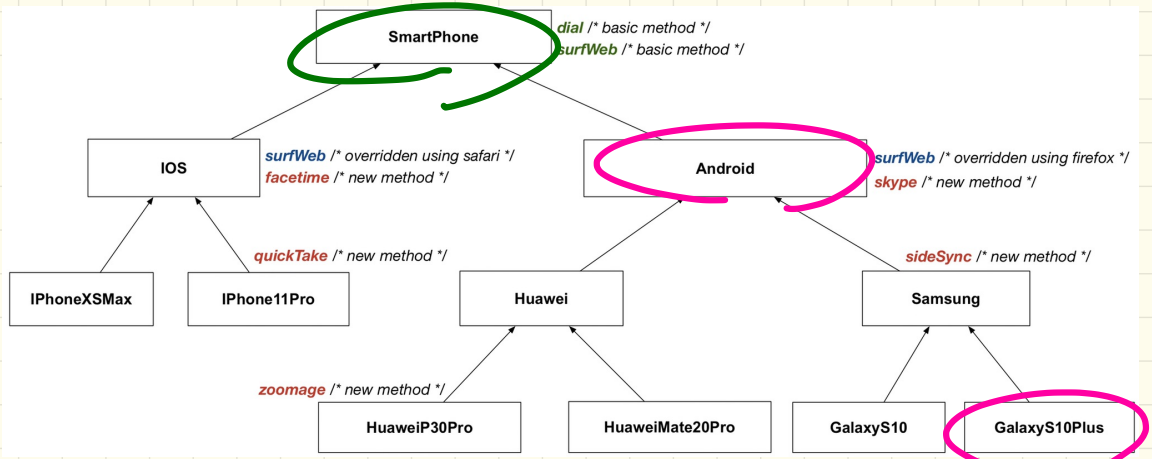
① Down cast

```
SmartPhone sp1 = (Samsung) s;
SmartPhone sp2 = (SmartPhone) s;
GalaxyS10Plus sp3 = (Samsung) s;
```

ST: Samsung - ✓ not a desc. of GS10P

ST: SmartPhone

ST: Samsung ✗



Android s = new Samsung(); ok: down

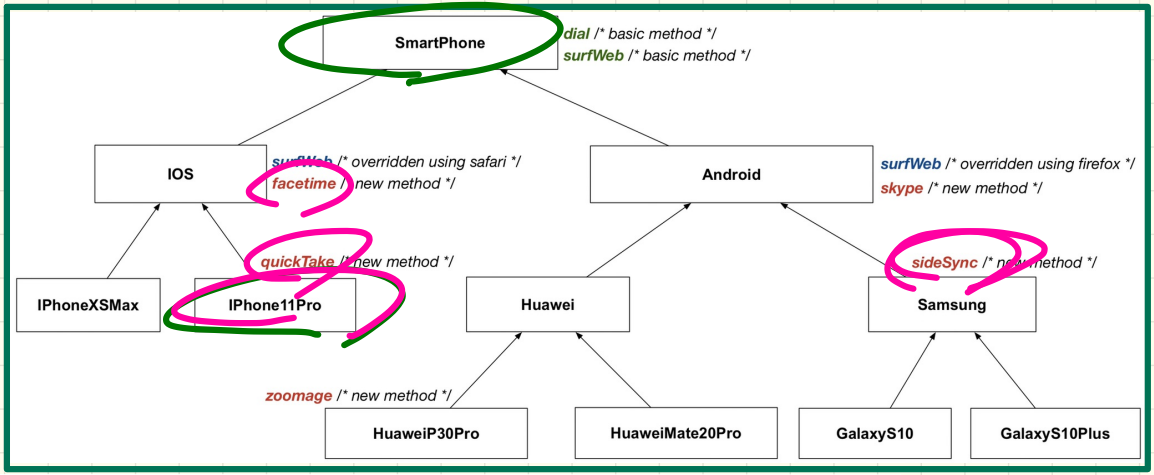
Android sp4 = (Samsung) s;

SmartPhone sp5 = (GalaxyS10Plus) s;

Samsung sp6 = sp5;

Type Cast

Named vs. Anonymous



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

```
SmartPhone aPhone = new IPhone11Pro();
IOS forHeeyeon = (IPhone11Pro) aPhone;
forHeeyeon.facetime();
```

down ST: IPhone11Pro.

Anonymous Cast: Use the cast result directly.

```
SmartPhone aPhone = new IPhone11Pro();
((IPhone11Pro) aPhone).facetime();
```

- sideSync X
- quickTake ✓

ST: IP11Pro

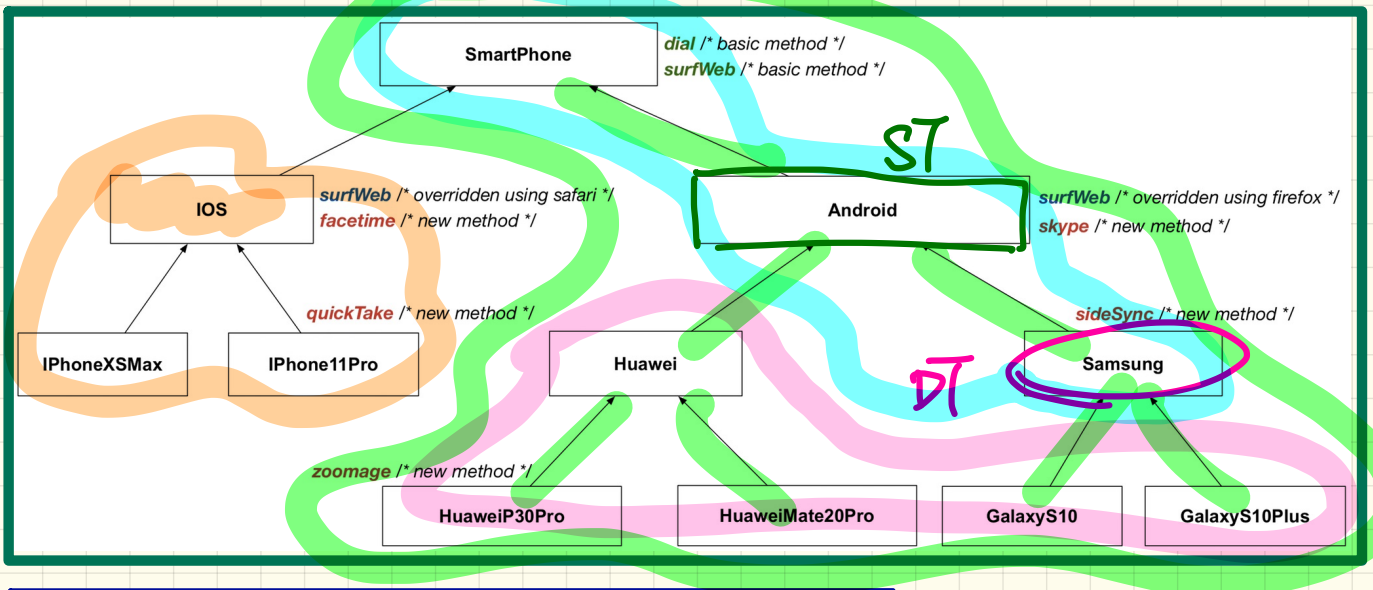
(IP11Pro) aPhone.facet

```
1 SmartPhone aPhone = new IPhone11Pro();
2 ((IPhone11Pro) aPhone).facetime();
```

✓ X

UP

Compilable Cast vs. Exception-Free Cast




```
Android myPhone = new Samsung();
```

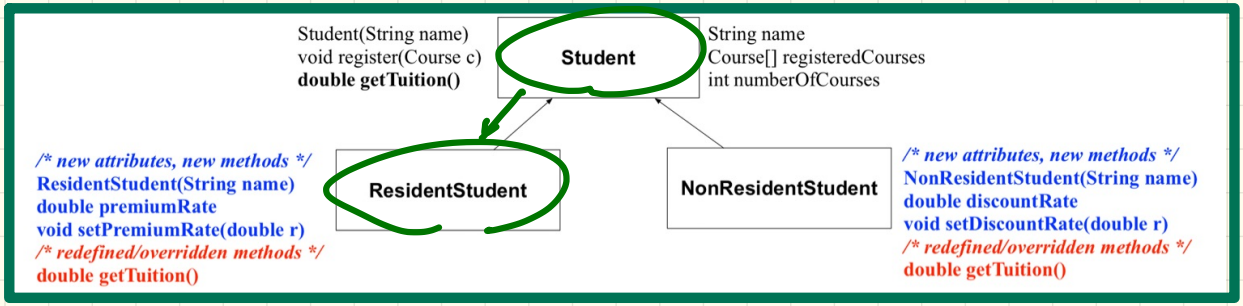
Compilable Casts

Exception-Free Casts

Non-Compilable Casts

ClassCastException

Compilable Type Cast May Fail at Runtime (1)




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

Compiles ok down

ST: RS

S jim

RS rs

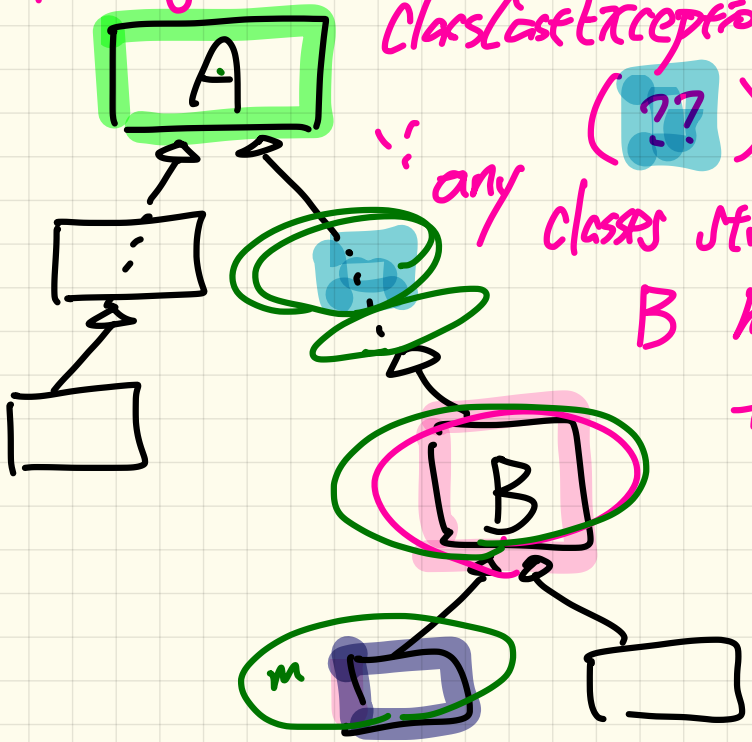
NRS

ClassCastExcep.

Executing this at runtime, it will be found that the DT of rs (NRS) cannot fulfill the expectations on ST of rs(RS)

- Down cast always compiles.
- Down cast beyon the DT of obj will cause ClassCastException

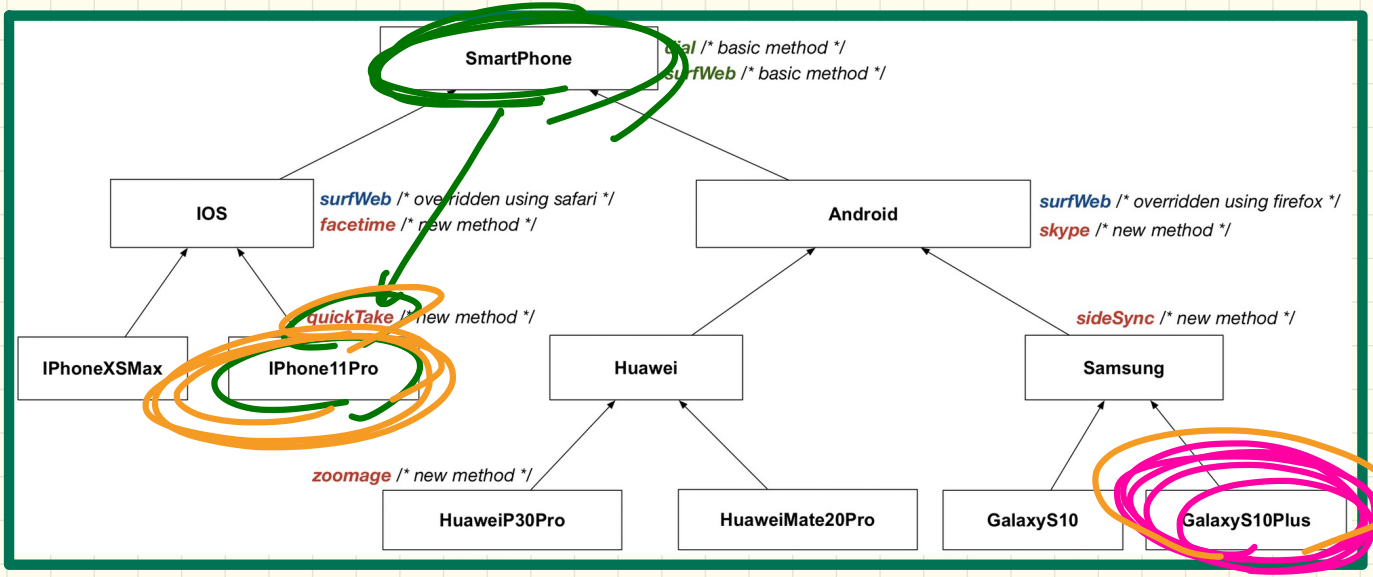
A obj = new B();



(??) obj

∴ any classes strictly lower than B have expectations that B can't support.

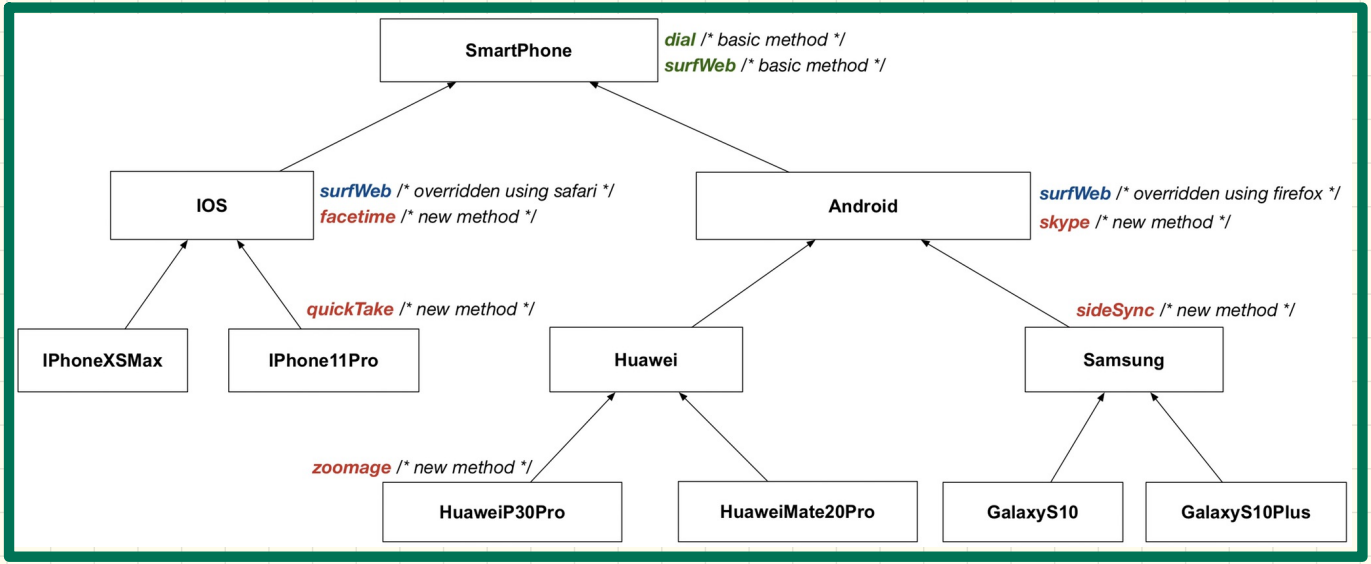
Compilable Type Cast May Fail at Runtime (2)




```
1 SmartPhone aPhone = new GalaxyS10Plus();
2 IPhone11Pro forHeeyeon = (IPhone11Pro) aPhone;
3 forHeeyeon.quickTake();
```

valid

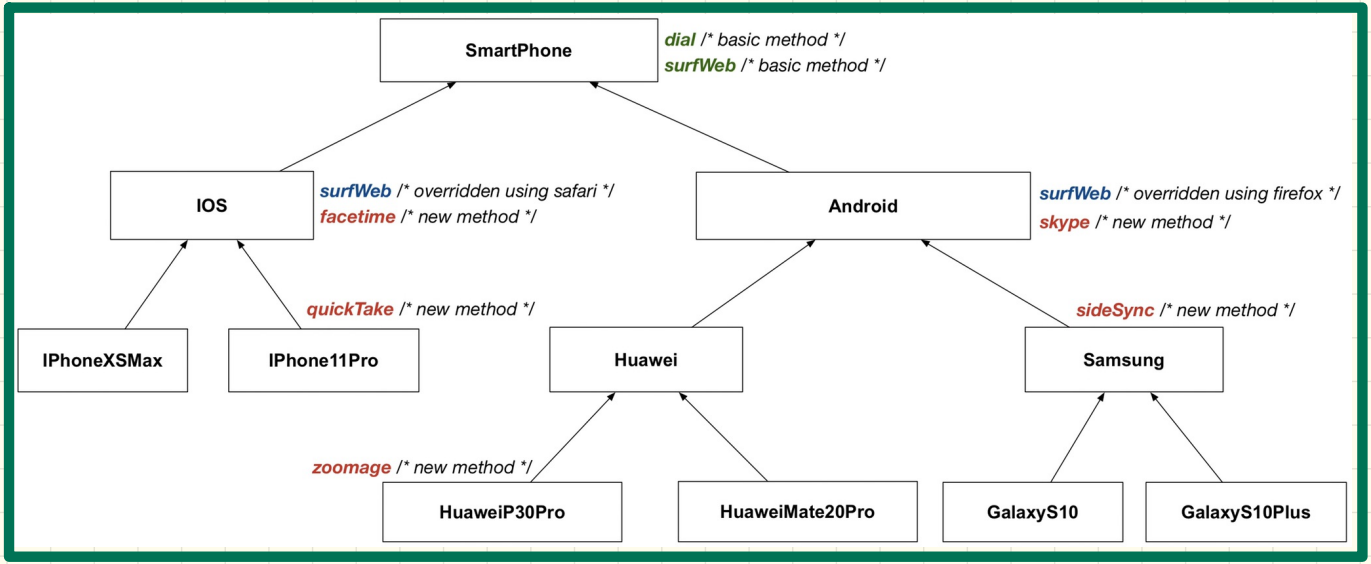
Exercise: Compilable Type Cast? Fail at Runtime? (1)




```
SmartPhone myPhone = new Samsung();
/* ST of myPhone is SmartPhone; DT of myPhone is Samsung */
GalaxyS10Plus ga = (GalaxyS10Plus) myPhone;
```

Compilable? ClassCastException at runtime?

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



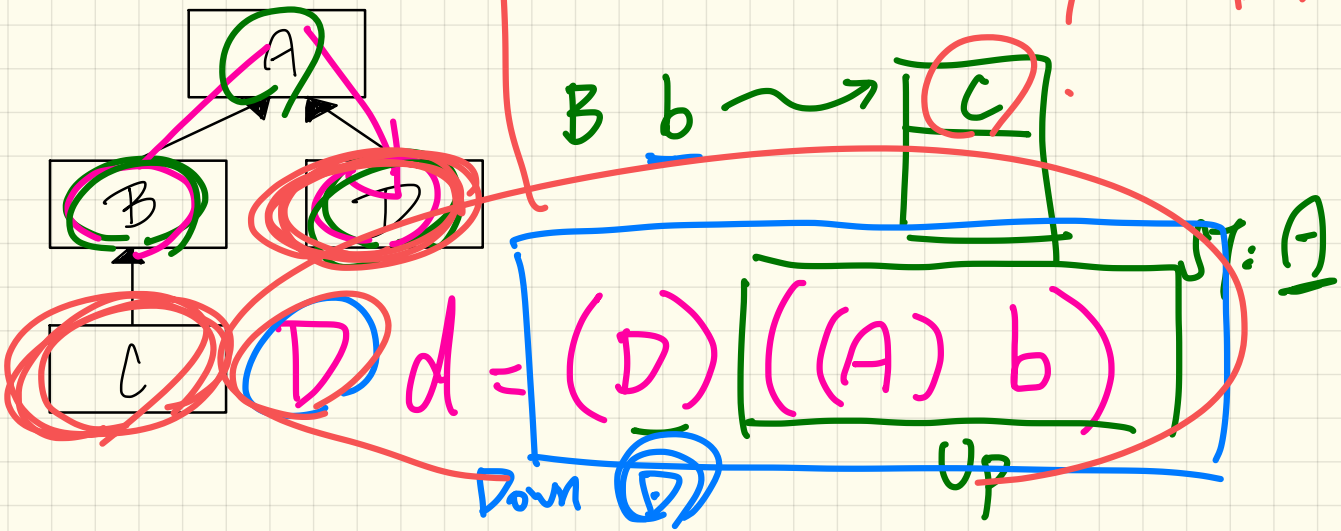

```
SmartPhone myPhone = new Samsung();
/* ST of myPhone is SmartPhone; DT of myPhone is Samsung */
IPhone11Pro ip = (IPhone11Pro) myPhone;
```

Compilable? ClassCastException at runtime?

Compilable Cast vs. Exception-Free Cast: Exercise

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

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

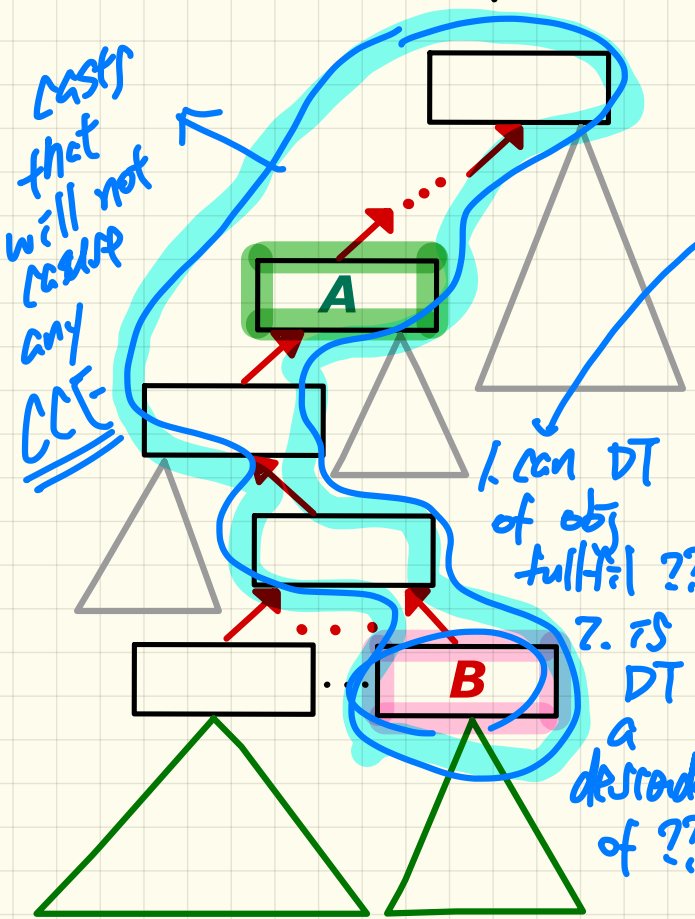


```
1  SmartPhone aPhone = new IPhone11Pro();
2  (IPhone11Pro) aPhone.[redacted]();
```

dial. X

void dial .

The instanceof Operator



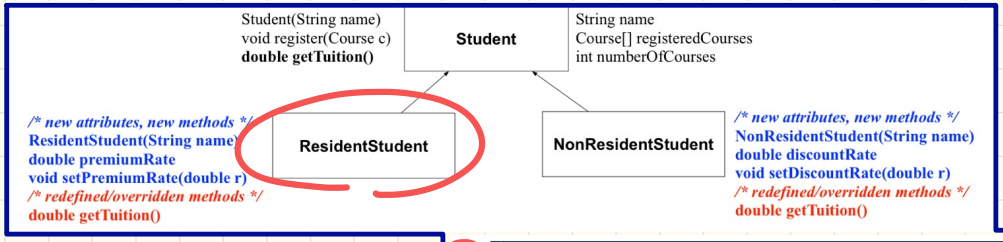

```
1  A obj = new B();
2  if (obj instanceof ??) {
3      ?? obj2 = (??) obj;
   }
```

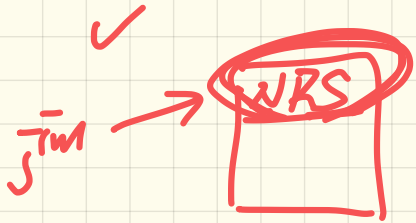
- L1 compiles if B can fulfill expectations of A.

- 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 ??.

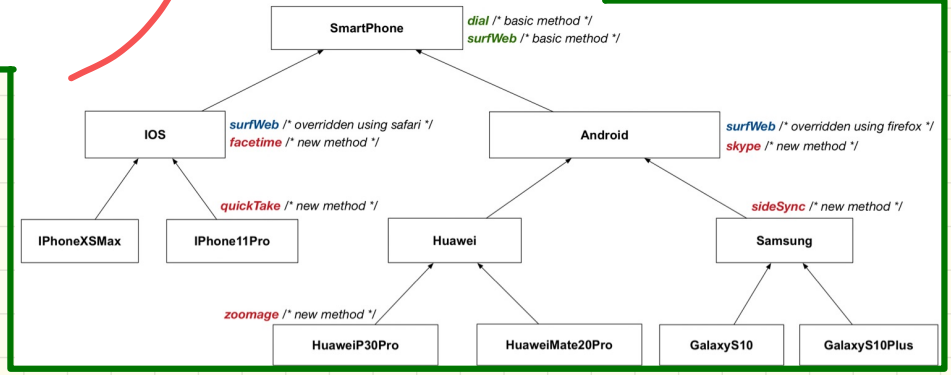
Checking Dynamic Types at Runtime



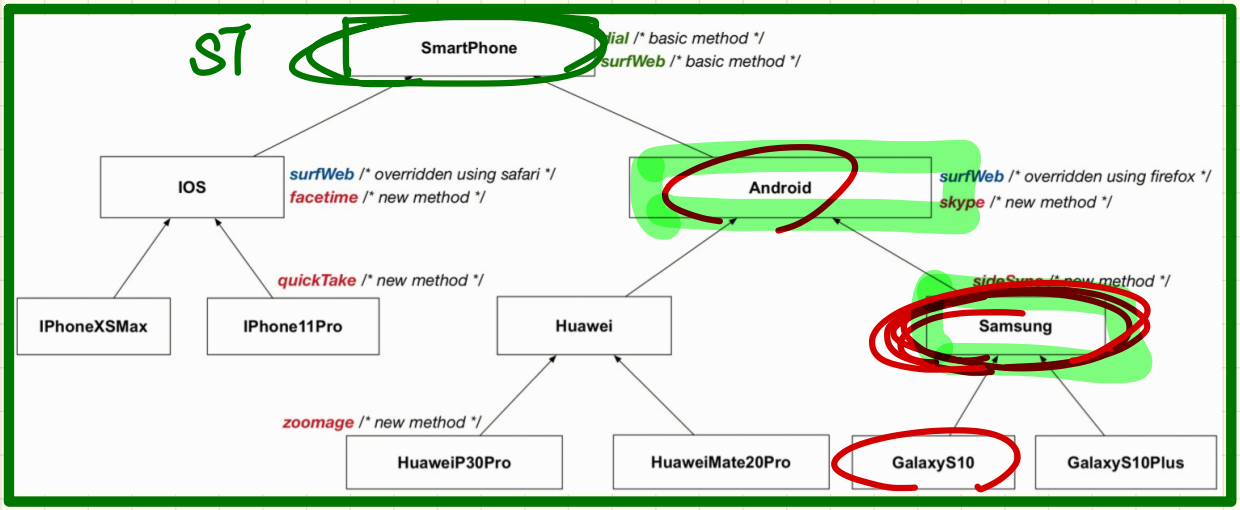



```
1  Student jim = new NonResidentStudent("J. Davis");
2  if (jim instanceof ResidentStudent)
3    ResidentStudent rs = (ResidentStudent) jim;
4    rs.setPremiumRate(1.5);
5  }
```

```
1  SmartPhone aPhone = new GalaxyS10Plus();
2  if (aPhone instanceof IPhone11Pro) {
3    IOS forHeeyeon = (IPhone11Pro) aPhone;
4    forHeeyeon.facetime();
5  }
```



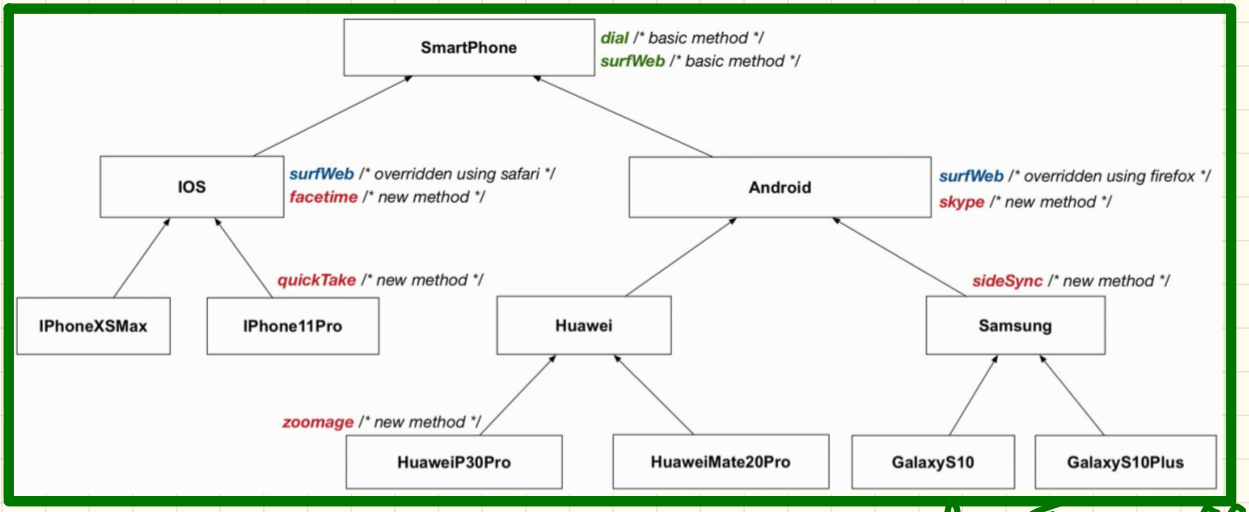
Use of the instanceof Operator




```
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 GalaxyS10);
/* false ∵ Samsung is not a descendant of GalaxyS10 */
println(myPhone instanceof IOS);
/* false ∵ Samsung is not a descendant of IOS */
println(myPhone instanceof IPhone11Pro);
/* [redacted] Samsung is not a descendant of IPhone11Pro */
```

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

Safe Cast via Use of the instanceof Operator



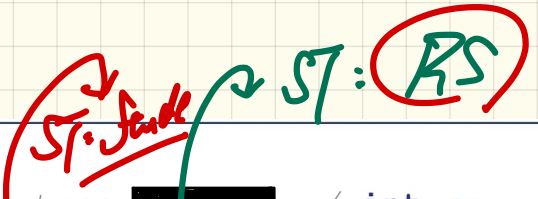

```
1  SmartPhone myPhone = new Samsung();
2  /* ST of myPhone is SmartPhone; DT of myPhone is Samsung
3  if(myPhone instanceof Samsung) {
4    Samsung samsung = (Samsung) myPhone;
5  }
6  if(myPhone instanceof GalaxyS10Plus) {
7    GalaxyS10Plus galaxy = (GalaxyS10Plus) myPhone;
8  }
9  if(myphone instanceof HTC) {
10   HTC htc = (HTC) myPhone;
11 }
```

before I can cast myPhone into Samsung, run a check to see if the DT of myP. can fullfil all expec. of Samsung

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

Polymorphic Arguments (1)

```
1  class StudentManagementSystem {
2    Student[] ss; /* ss[i] has static type [redacted] */ int c;
3    void addRS(ResidentStudent rs) { ss[c] = rs; c ++; }
4    void addNRS(NonResidentStudent nrs) { ss[c] = nrs; c++; }
5    void addStudent(Student s) { ss[c] = s; c++; } }
```



Q. Static type of ss[0], ss[1], …, ss[ss.length - 1]?

Q. In method addRS, does ss[c] = rs compile?