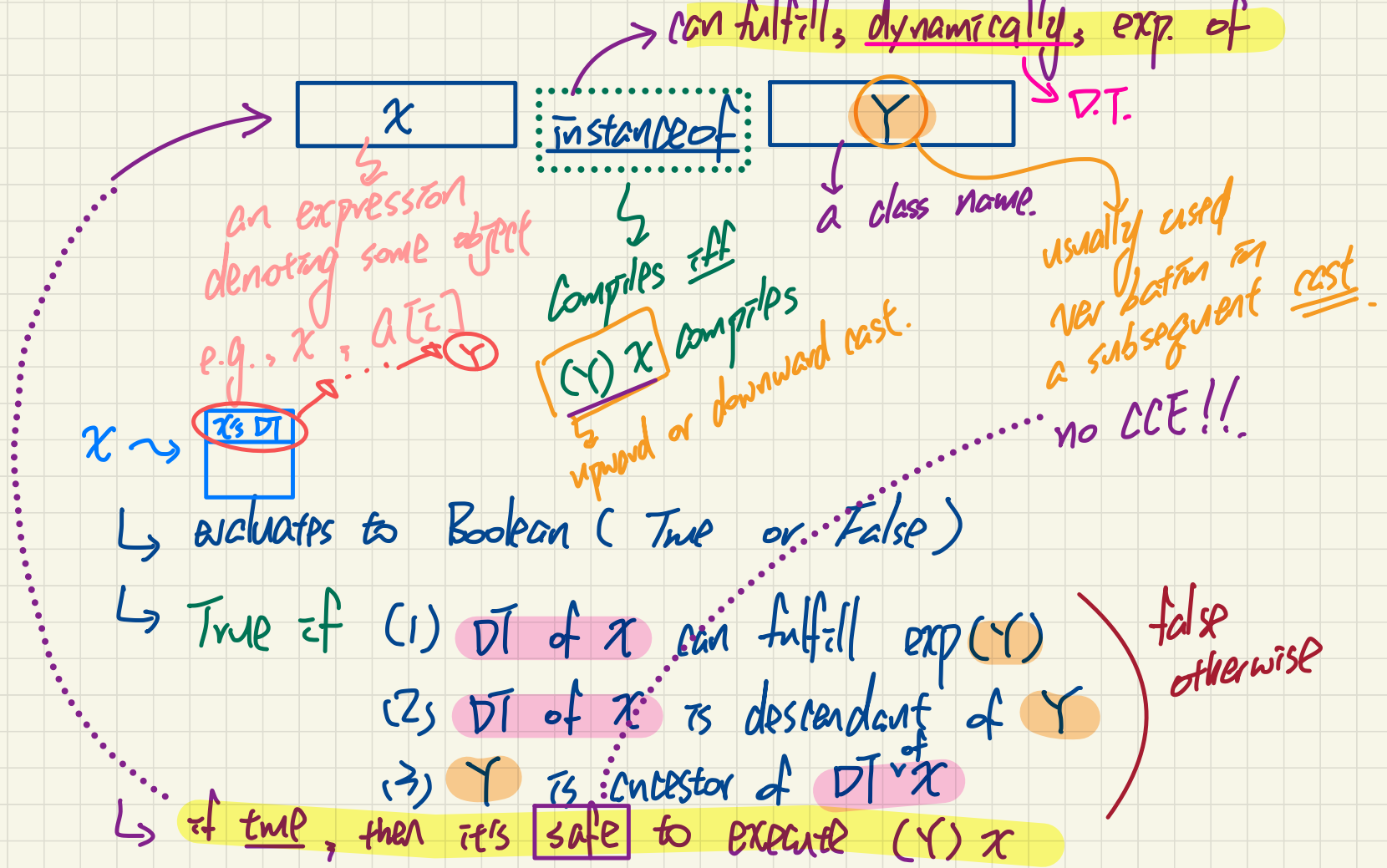
Lecture 20 - Nov 18

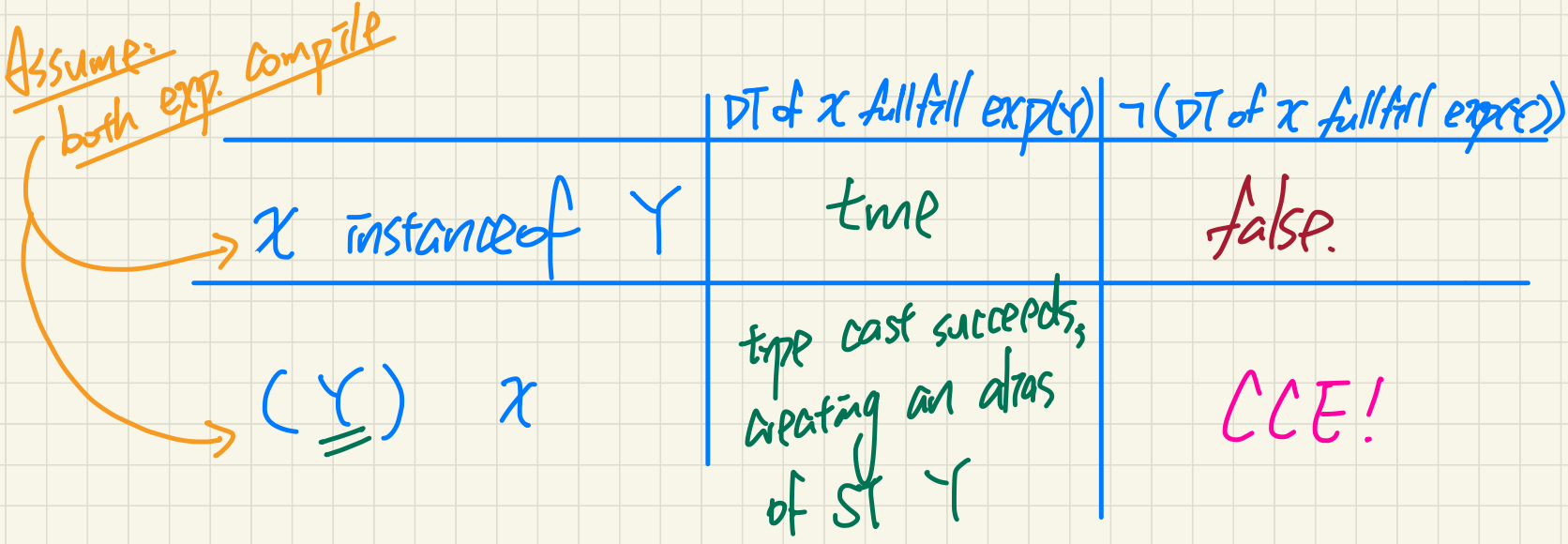
Inheritance

instanceof vs. Compilable Cast
instanceof vs. ClassCastException Freedom
Call by Value and Inheritance

Announcements/Reminders

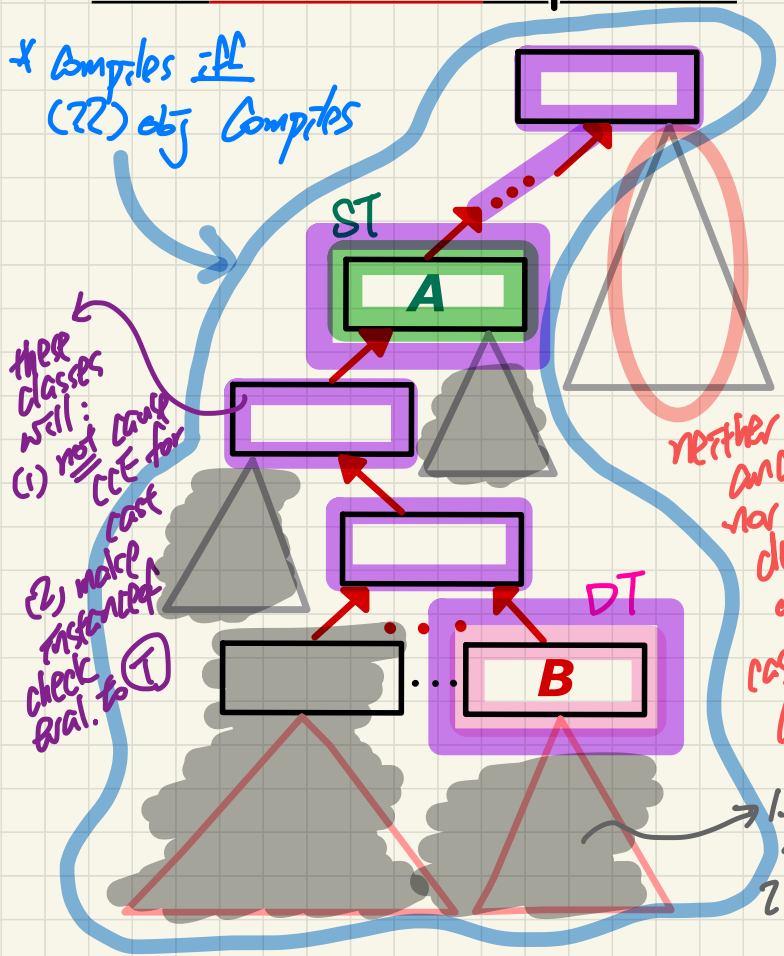
- Today's class: notes template posted
- Lab4 due soon
- WrittenTest2 guide and example questions
  + In-person review session: 3 PM on Tuesday @ R N203
  + Notes on Type Casting
- Online Course Evaluation


can fulfill, dynamically, exp. of
D.T.
x
instanceof
Y
an expression denoting some object
e.g., x : a[i]
a class name.
usually used w/o actual as a subsequent cast.
Compiles iff
(Y) x compiles
upward or downward cast.
no CCE!!
x ~
x's DT
↳ evaluates to Boolean (True or False)
↳ True if (1) DT of x can fulfill exp(Y)
(2) DT of x is descendant of Y
(3) Y is ancestor of DT of x
false otherwise
↳ if true, then it's safe to execute (Y) x


Assume: both exp. compile
DT of x fulfill exp(Y)
¬(DT of x fulfill exp(Y))
x instanceof Y
true
false.
(Y) x
type cast succeeds, creating an alias of Y
CCE!

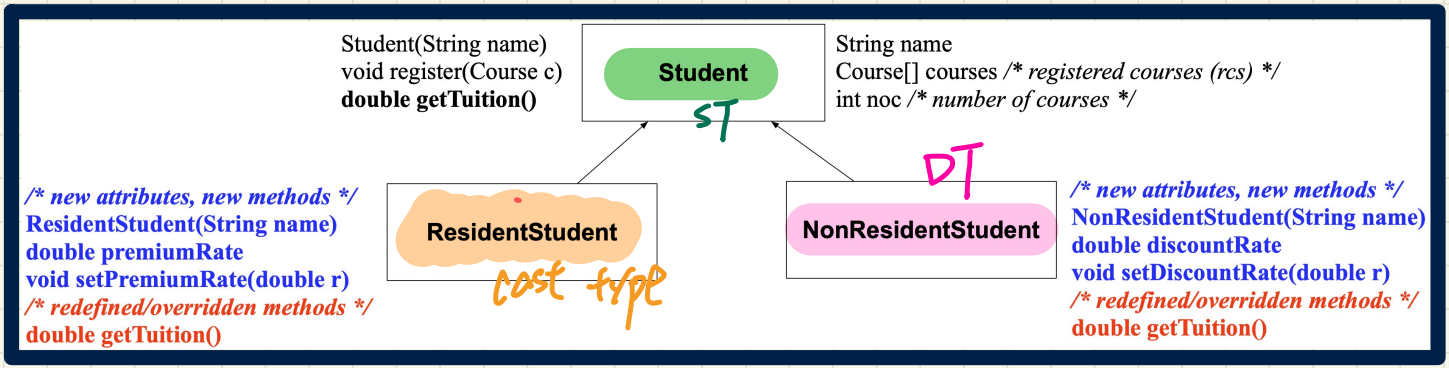
The instanceof Operator

```
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)




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

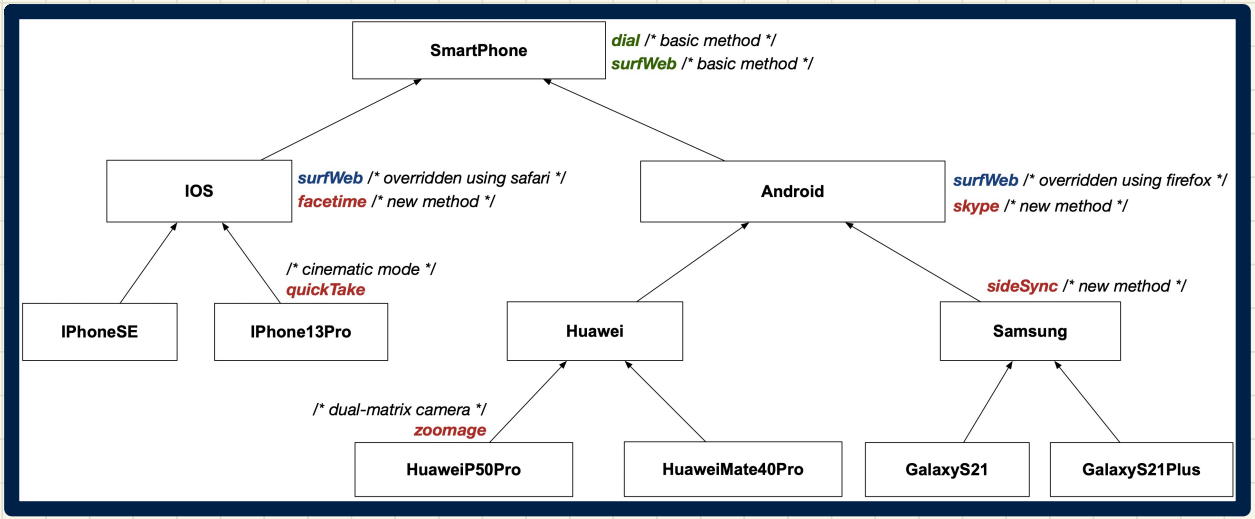
1. compile? downward cast
2. CCE! ∵ DT NRS not descendant of cast type RS.

....bypassed, preventing CCE from happening!!

Can jim's DT fulfill the ECD(RS)
↳ NO → false

Checking Dynamic Types at Runtime (2)

ExercISP




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

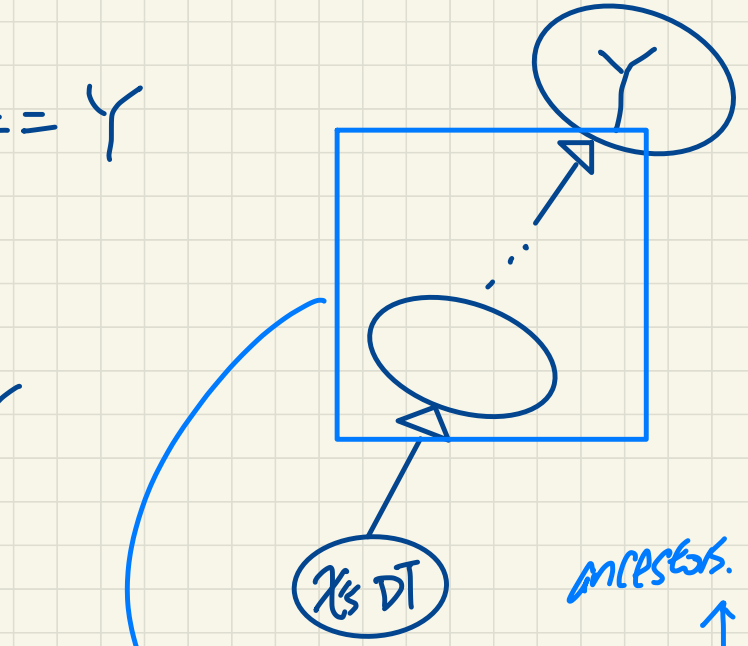
$x.getClass() == Y$

DT of x

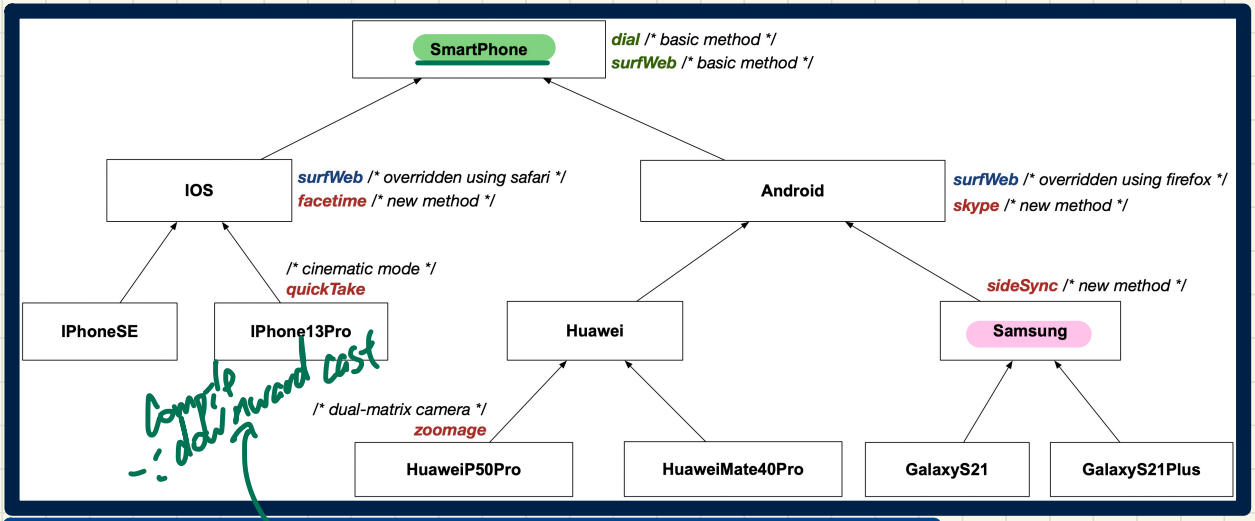
x instanceof Y

$\leq$

DT of x



Use of the instanceof Operator

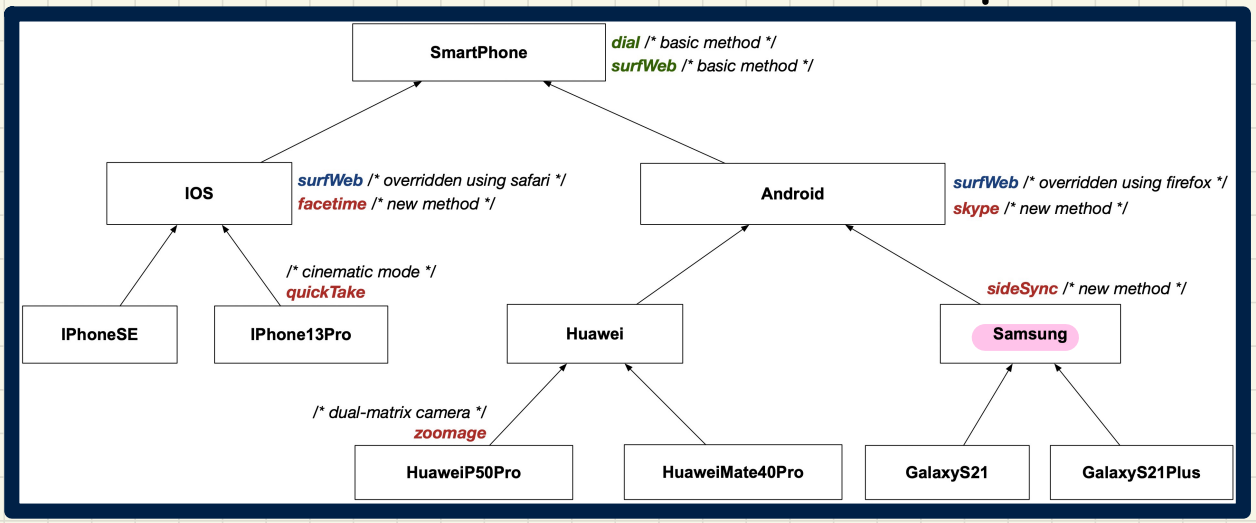



```
SmartPhone myPhone = new Samsung();
println(myPhone instanceof Android);
println(myPhone instanceof Samsung);
println(myPhone instanceof GalaxyS21);
println(myPhone instanceof IOS);
println(myPhone instanceof IPhone13Pro);
```

T T F F F

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 }
```

T

E no CCE

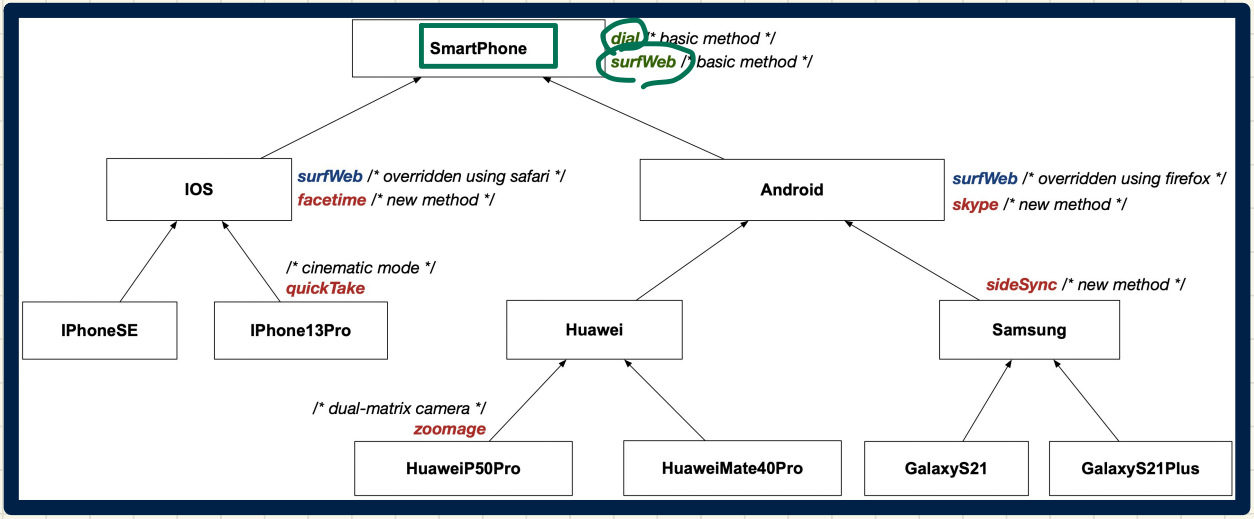
X bypassed

E

X CCE prevented!

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

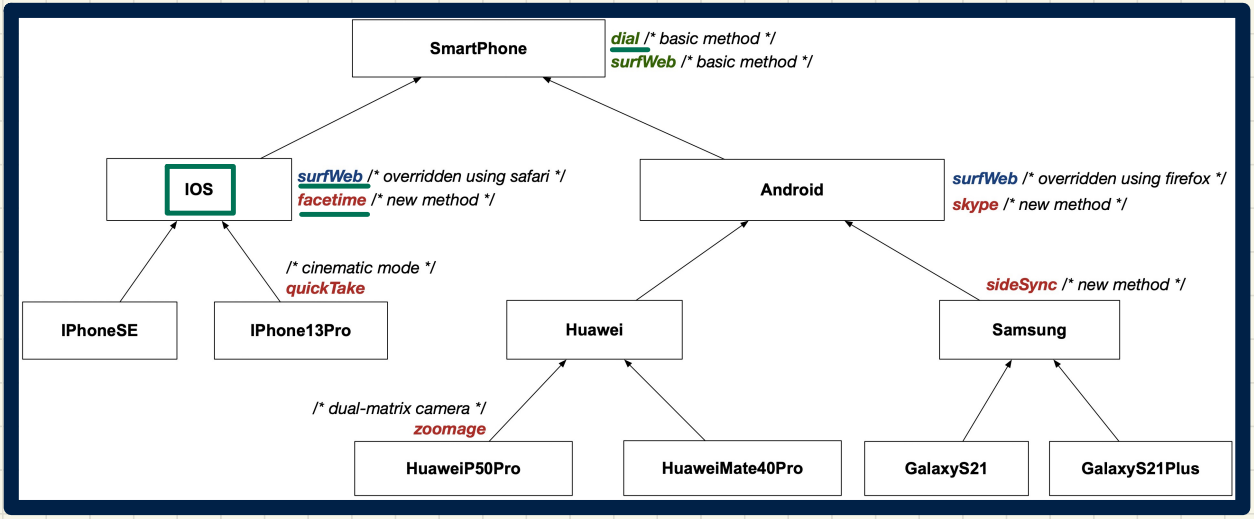
Static Types, Casts, Polymorphism (1)




```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

```
1  SmartPhone sp = new IPhone13Pro();
2  sp.dial();
3  sp.facetime();
4  sp.quickTake();
```

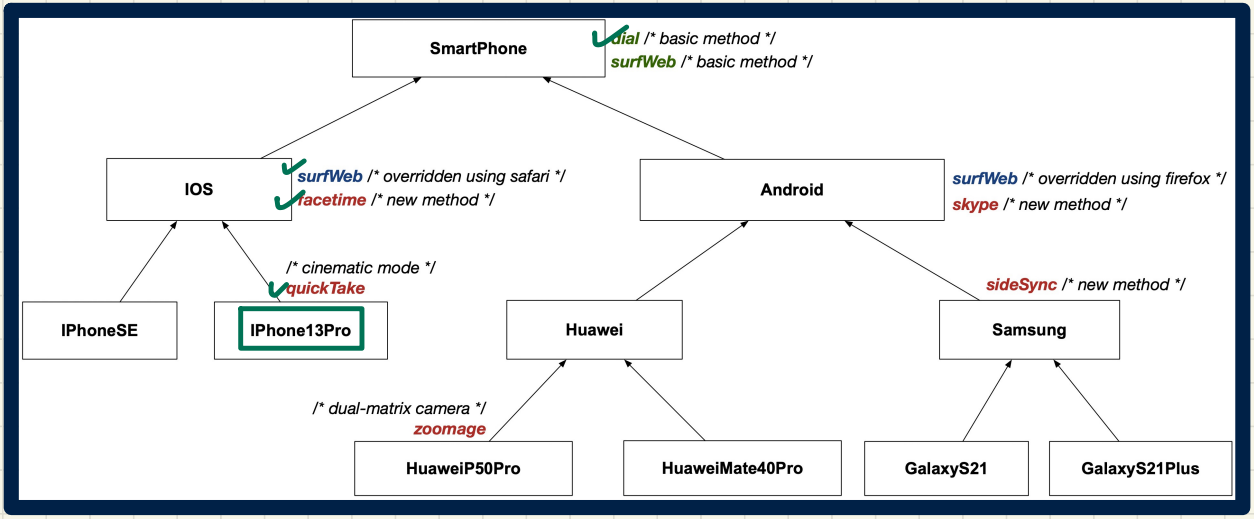
Static Types, Casts, Polymorphism (2)




```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

```
1  IOS ip = new IPhone13Pro();
2  ip.dial();✓
3  ip.facetime();✓
4  ip.quickTake();✗
```

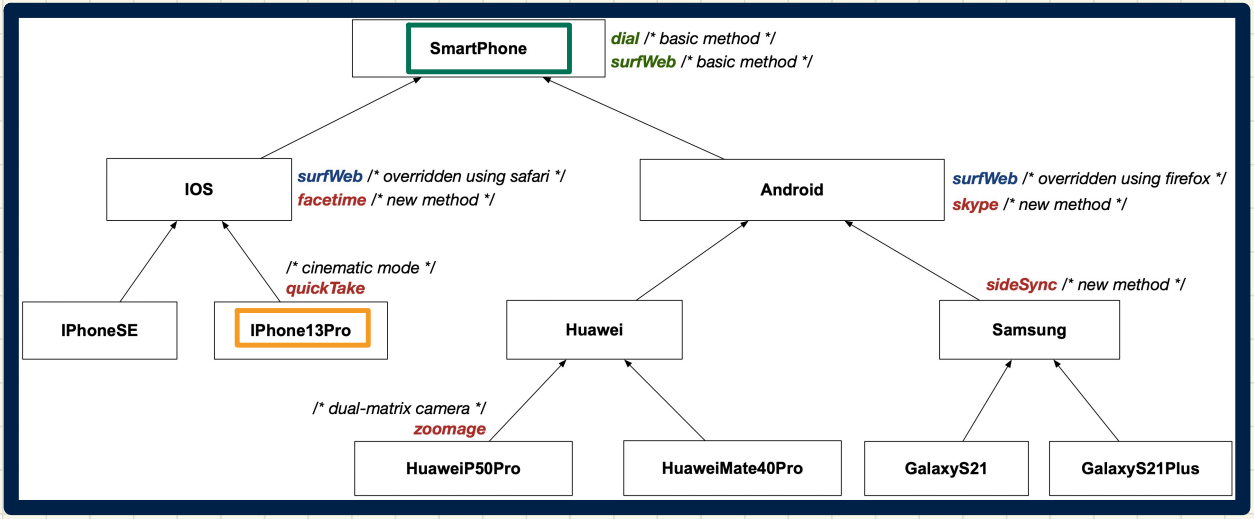
Static Types, Casts, Polymorphism (3)




```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

```
1  IPhone13Pro ip6sp = new IPhone13Pro();
2  ip6sp.dial();
3  ip6sp.facetime();
4  ip6sp.quickTake();
```

Static Types, Casts, Polymorphism (4)

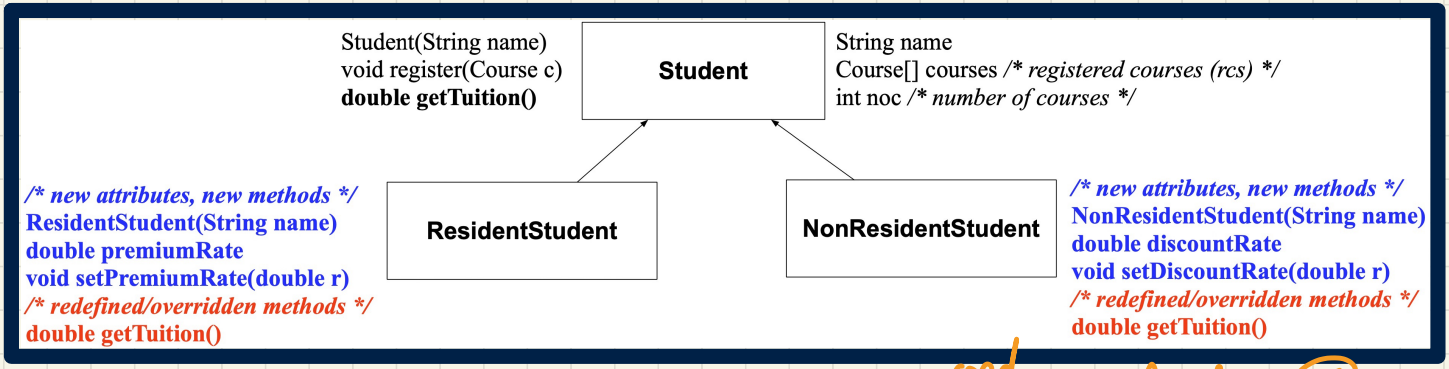



```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

```
1  SmartPhone sp = new IPhone13Pro();
2  ((IPhone13Pro) sp).dial();
3  ((IPhone13Pro) sp).facetime();
4  ((IPhone13Pro) sp).quickTake();
```

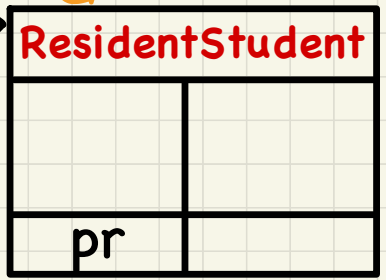
cast compiles (exp has ST: IP13Pro)

Static Types, Casts, Polymorphism (5)



safety guaranteed :: instanceof eval. to T. check

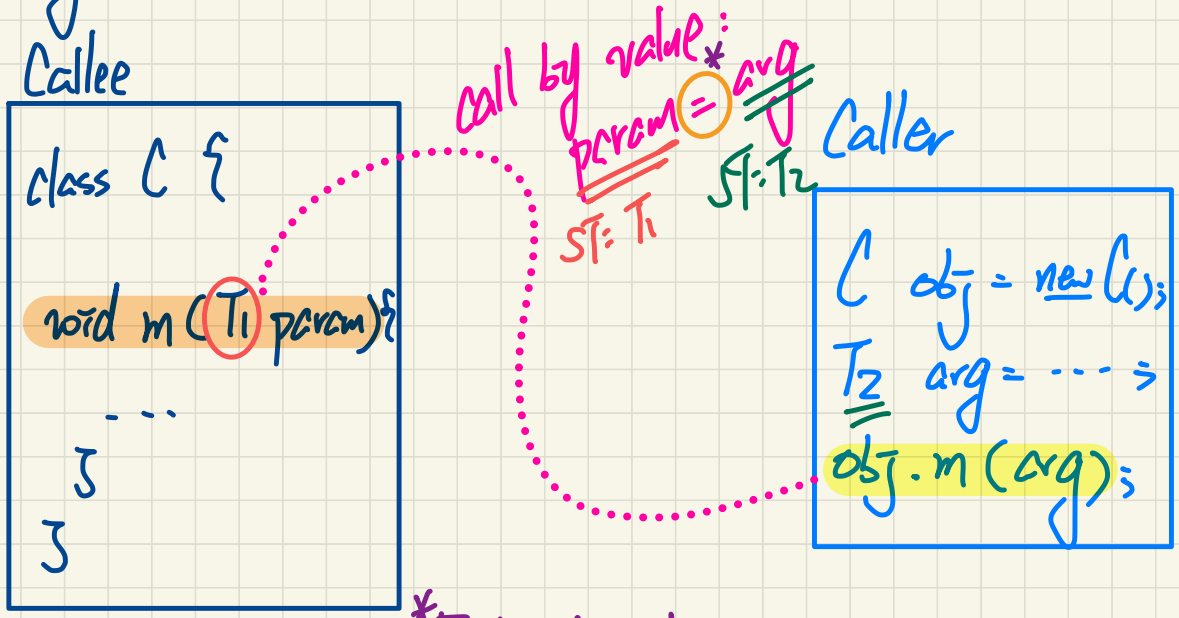
Student s




```
Course eecs2030 = new Course("EECS2030", 500.0);
Student s = new ResidentStudent("Jim");
s.register(eecs2030);
if(s instanceof ResidentStudent) {
  ((ResidentStudent) s).setPremiumRate(1.75);
  System.out.println(((ResidentStudent) s).getTuition());
}
```

true if it's safe to cast s to RS

Call by Value

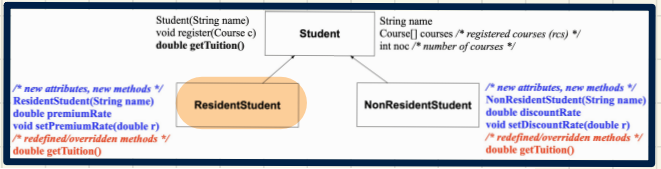


*Rule of substitutions

(1) T2 fulfills exp(T1)

(2) T2 descendant of T1

Polymorphic Parameters (1)



each element of ss may has ST Student.

```
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]?

ST: Student.

call by value: rs = o

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

ST: RS ST: ? compiles if ? is descendant of RS.

ST: Student ST: RS ✓

Q. Under what circumstances can the following method call be valid/compilable?

Student sms.addRS(o)

under what condition does this method call compile?

Polymorphic Parameters (2)

```
1  class StudentManagementSystem {
2    Student [] ss; /* ss[i] has static type Student */ 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++; } }
```

call to this method only compiles if the ST of arg. is descendant of ST of param (RS)

```
Student s1 = new Student();
Student s2 = new ResidentStudent();
Student s3 = new NonResidentStudent();
ResidentStudent rs = new ResidentStudent();
NonResidentStudent nrs = new NonResidentStudent();
StudentManagementSystem sms = new StudentManagementSystem();
sms.addRS(s1); ✗
sms.addRS(s2); ✗
sms.addRS(s3); ✗
sms.addRS(rs); ✓
sms.addRS(nrs); ✗
sms.addStudent(s1); ✓
sms.addStudent(s2); ✓
sms.addStudent(s3); ✓
sms.addStudent(rs); ✓
sms.addStudent(nrs); ✓
```

arg with ST a descendant of Student compiles

ST: RS → ST: S

call by value : rs = s1

ST: RS → ST: RS

rs = rs

C.b.v : rs = nrs ✗