Lecture 20 - Nov. 19

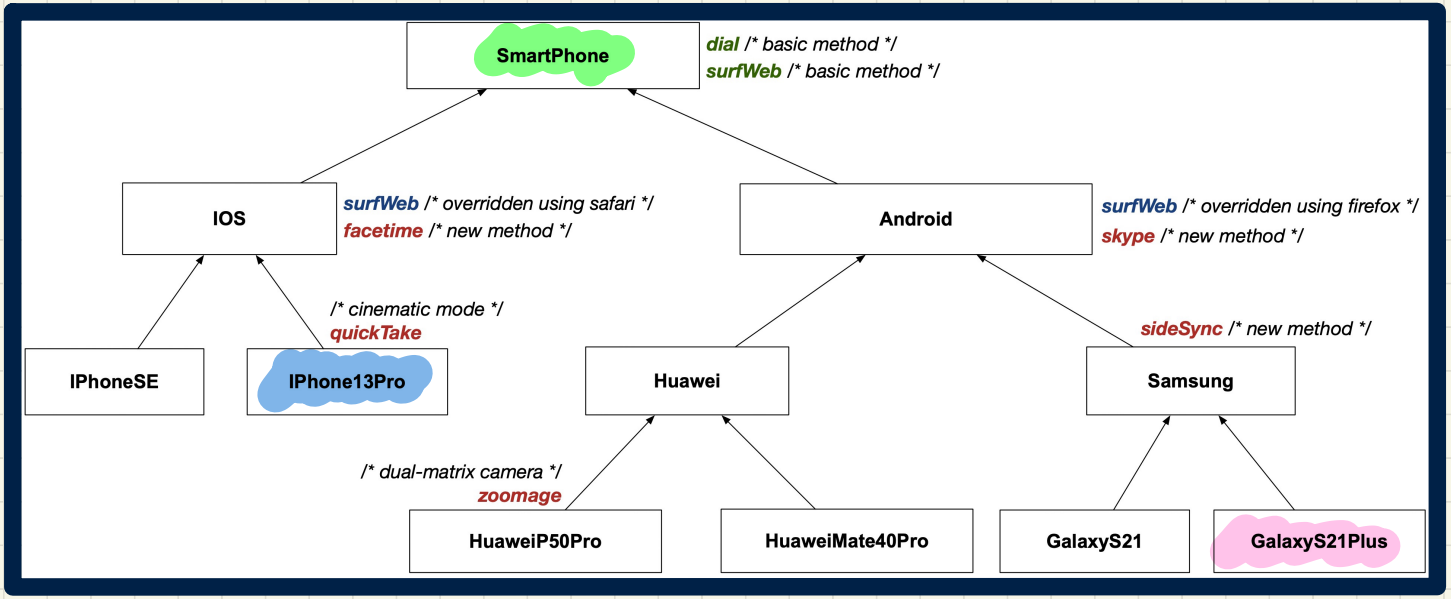
Inheritance

Type Casts: Exercise
Checking Dynamic Types: instanceof
Polymorphic Method Parameters

Announcements/Reminders

- **WrittenTest2** results released
- **Lab5** released
  + Required study: **Abstract Classes** & **Interfaces**
- **ProgTest3** tomorrow
  + **Not** covered: equals method, copy constructors
- **Bonus** Opportunity coming: Formal Course Evaluation

Compilable Type Cast May Fail at Runtime (2)



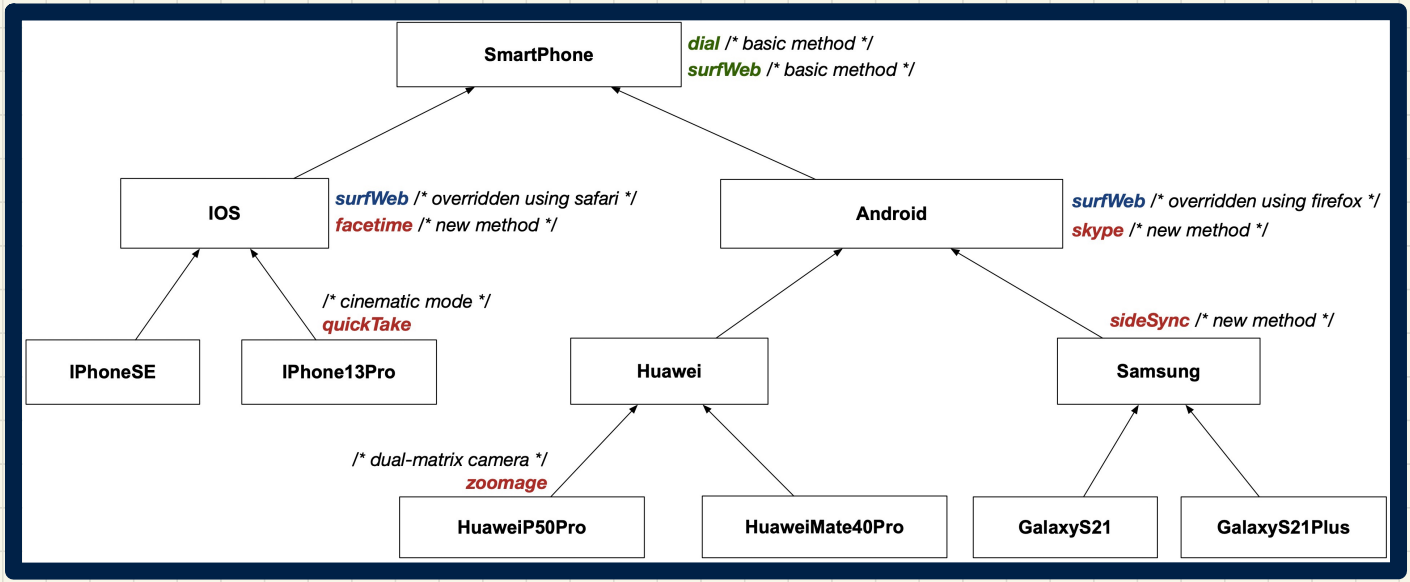

```
1  SmartPhone aPhone = new GalaxyS21Plus();
2  IPhone13Pro forHeeyeon = (IPhone13Pro) aPhone;
3  forHeeyeon.quickTake();
```

Compiles :: down ward cast

descendant of cast type IP13Pro

CCE :: aPhone's DT (GS21Plus) is not a

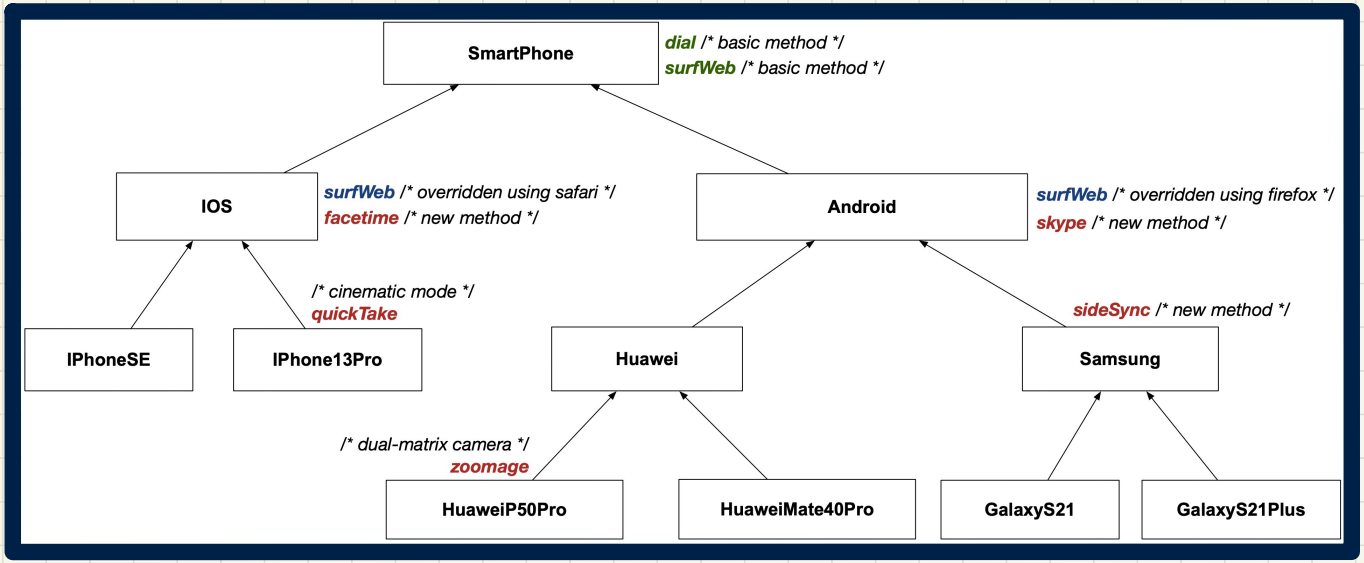
Exercise: Compilable Type Cast? Fail at Runtime? (1)




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

Compilable? ClassCastException at runtime?

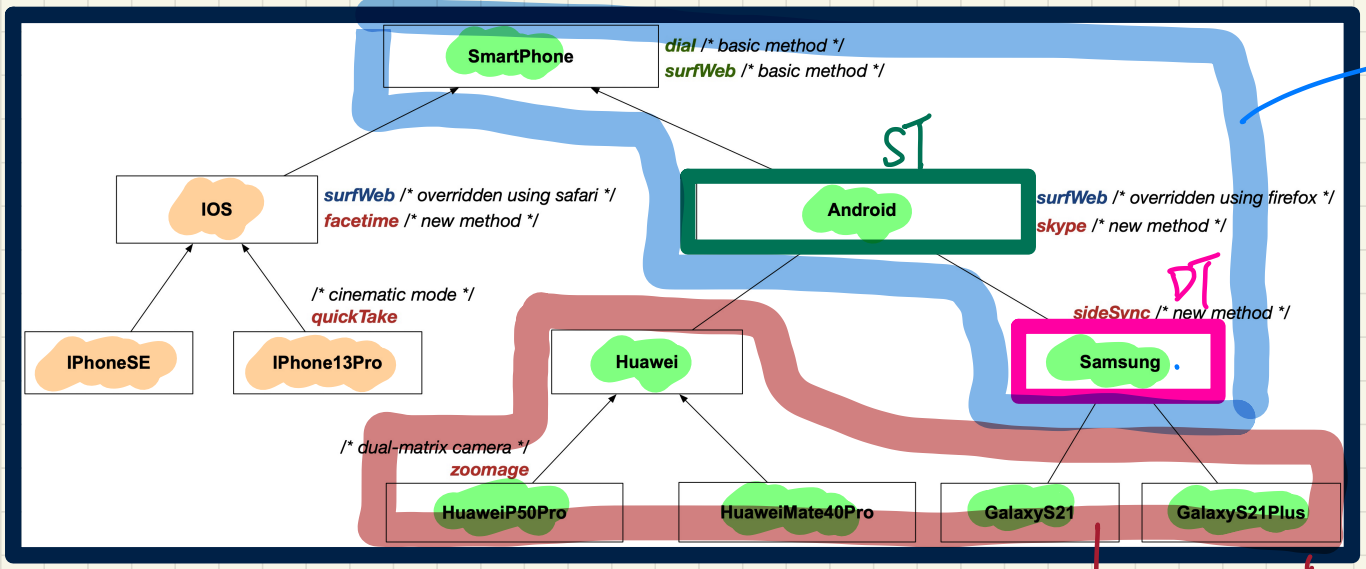
Exercise: Compilable Type Cast? Fail at Runtime? (2)




```
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



classes whose exp. can be fulfilled by DT.

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

classes that are not ancestors of DT (Samsung).

Compilable Casts

Exception-Free Casts

Non-Compilable Casts

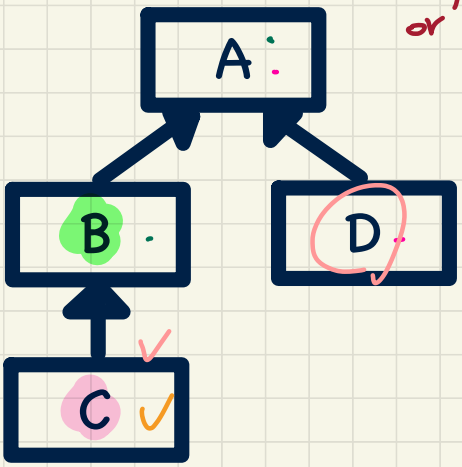
ClassCastException

Exercise: Compilable Cast vs. Exception-Free Cast

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

Compilable: ✓ (Blueprint)((Object) obj) ST: Vaccine

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



↳ X ∵ neither upward or downward cast.

CCE ∵ DT C is not a descendant of D

No CCE ∵ DT C is a descendant of A

Dd = (D)((A) b)

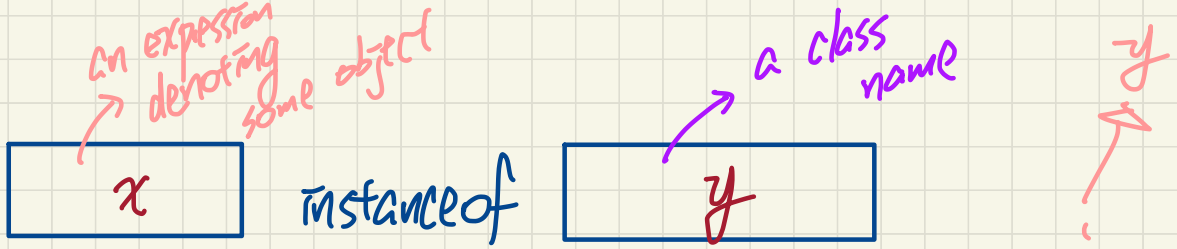
upward cast

downward cast

↳ Runtime:

?

(D) (A) b

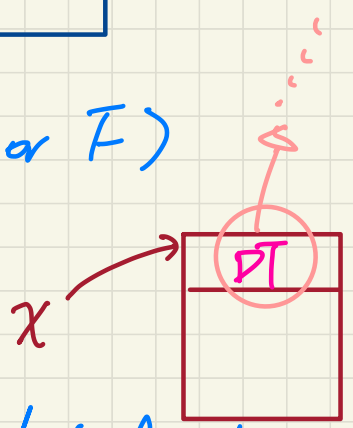


↳ evaluates to Boolean (T or F)

↳ True if (1) DT of x can fulfill the exp. of y

(2) DT of x is a descendant of y

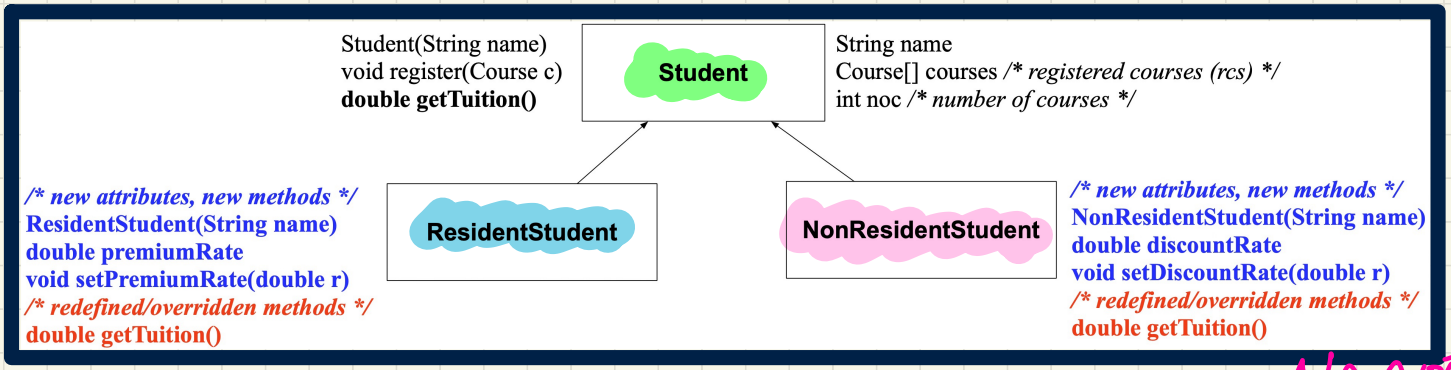
(3) y is an ancestor of the DT of x



↳ if true, then it's safe to write (y) x

No CCE

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  }
```

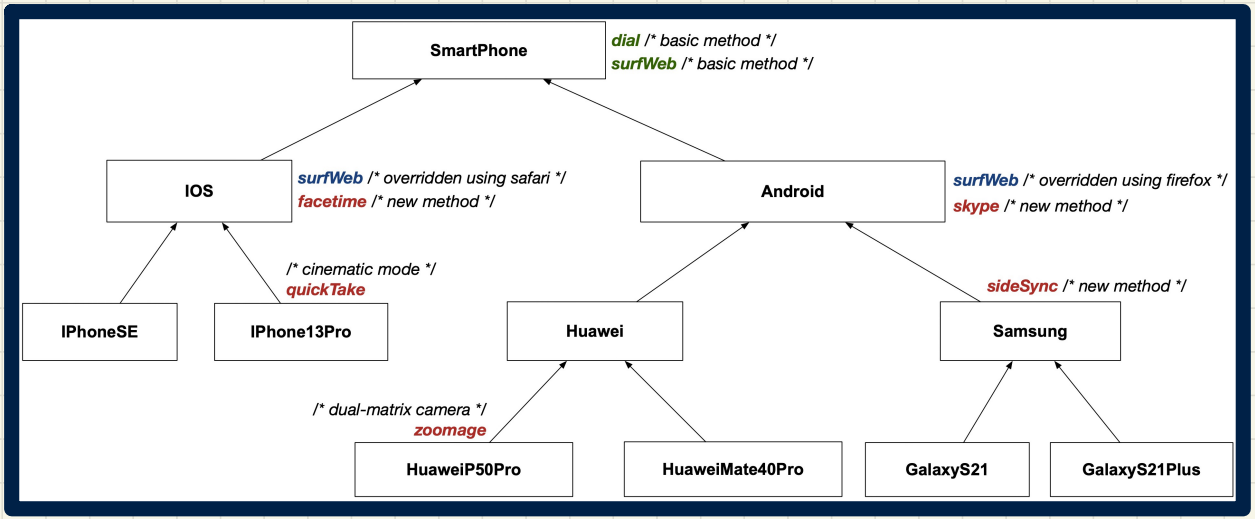
evaluating to false avoids the CCE

would trigger CCE

Can the DT of jim (NRS) fulfill exp. of RS ?

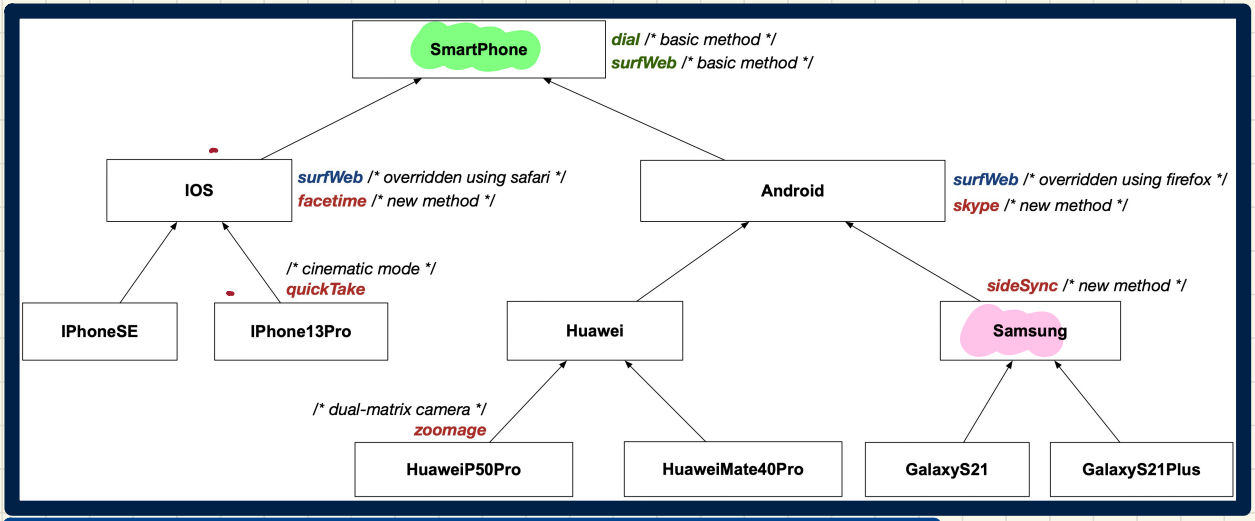
False :(NRS is not a descent of RS.

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  }
```

Use of the instanceof Operator

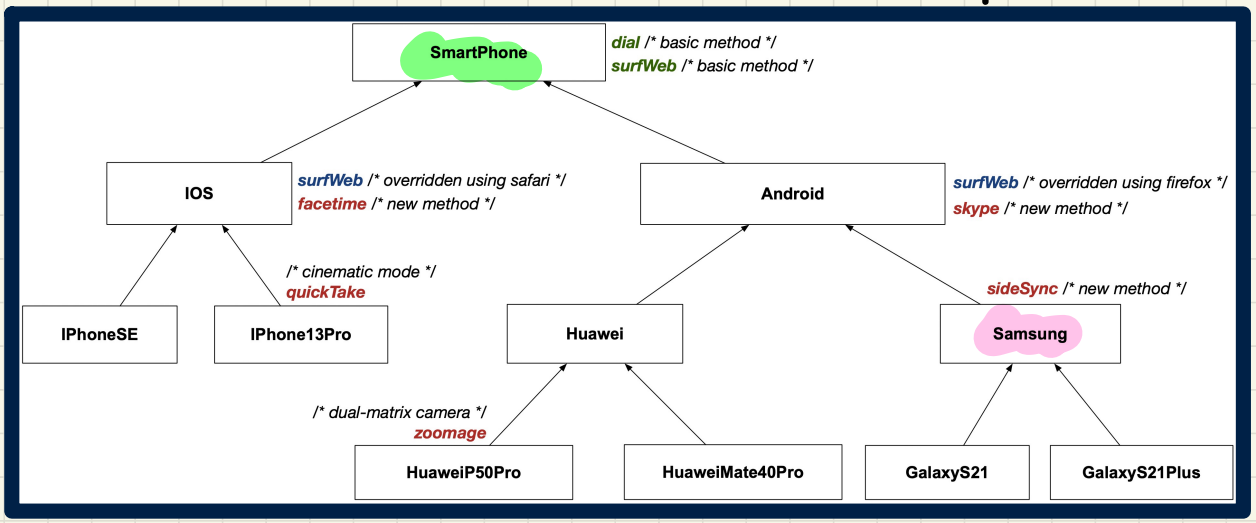



```
SmartPhone myPhone = new Samsung();
println(myPhone instanceof Android);
println(myPhone instanceof Samsung);
println(myPhone instanceof GalaxyS21);
println(myPhone instanceof IOS);
println(myPhone instanceof IPhone13Pro);
```

T T T F F

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

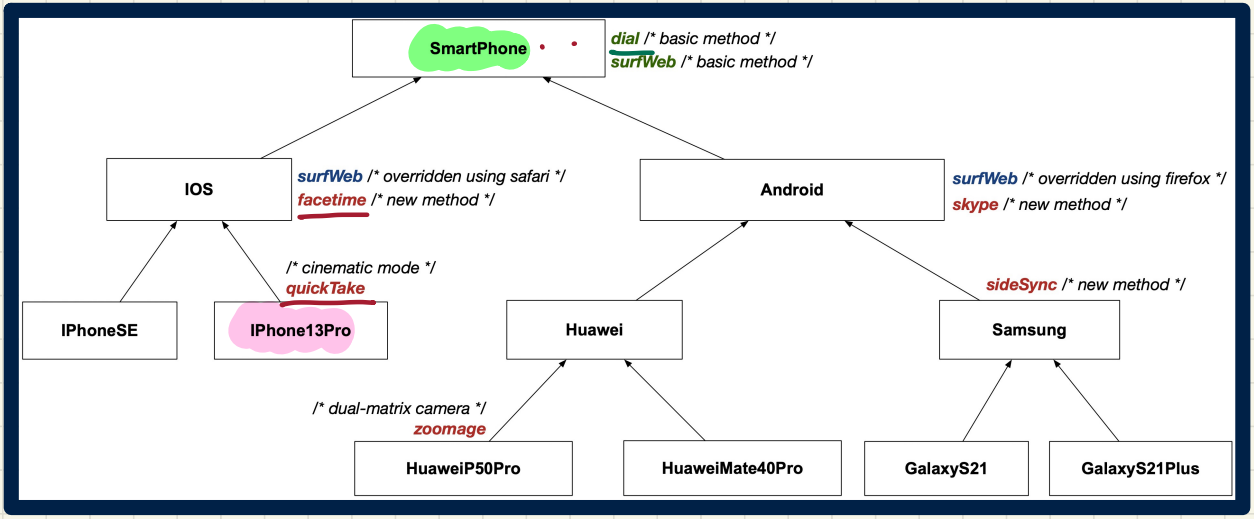
Safe Cast via Use of the instanceof Operator




```
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 }
```

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

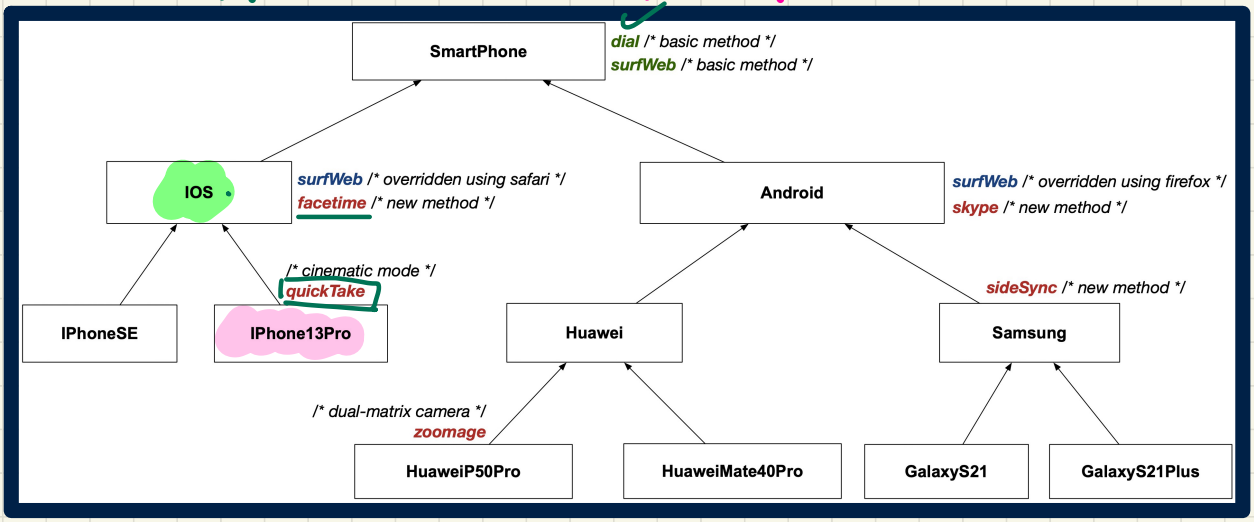
Static Types, Casts, Polymorphism (1)




```
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();
```

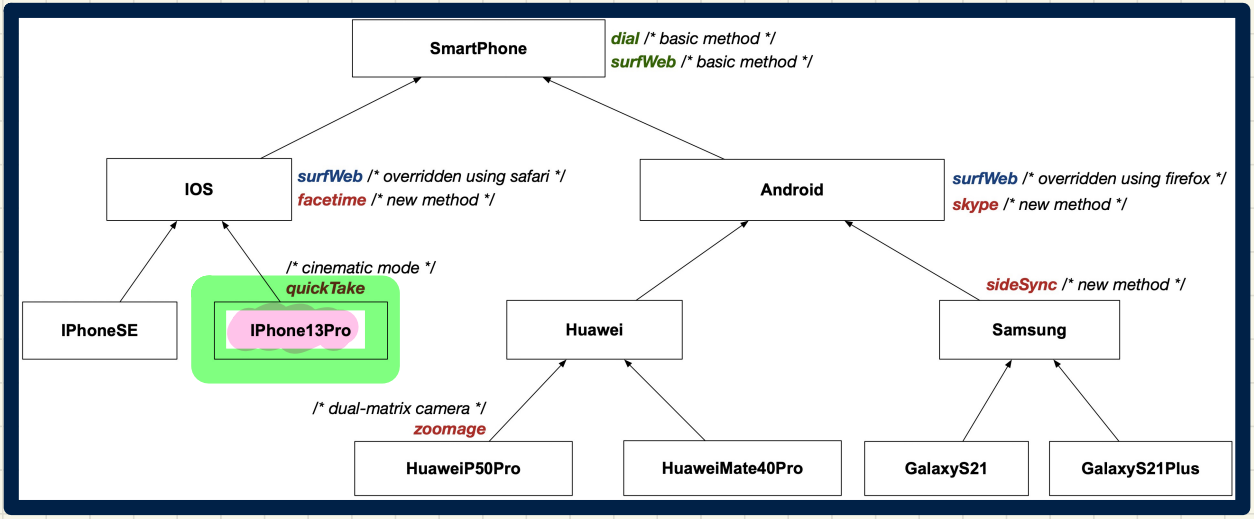
Static Types, Casts, Polymorphism (2)




```
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();
```

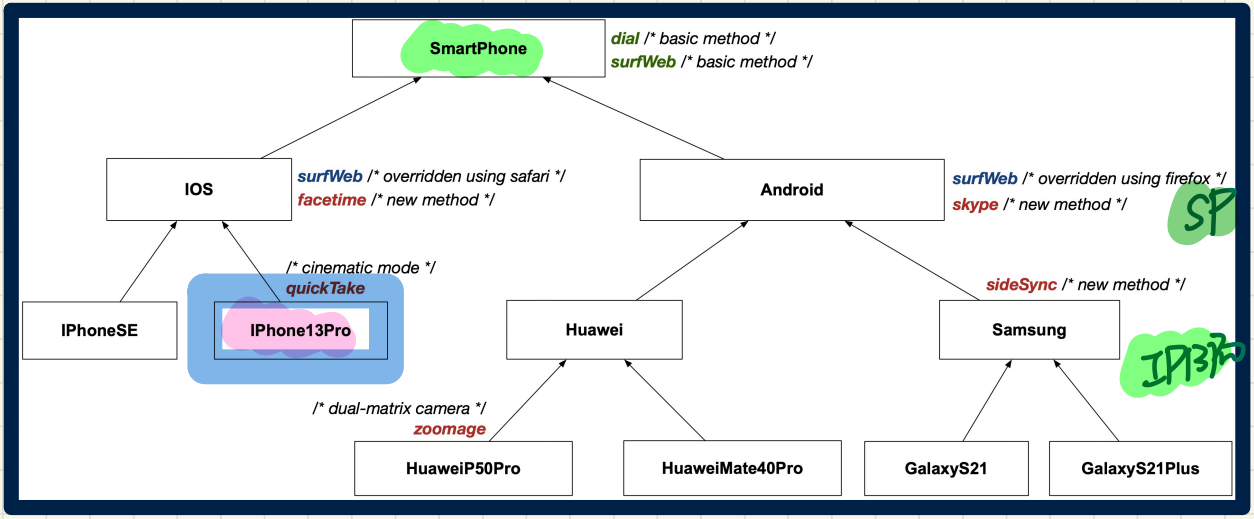
Static Types, Casts, Polymorphism (3)

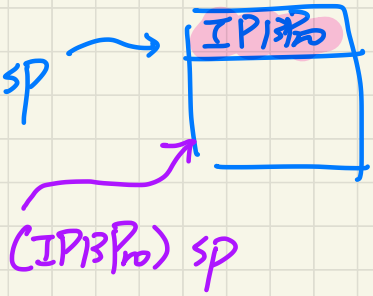



```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

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

Static Types, Casts, Polymorphism (4)

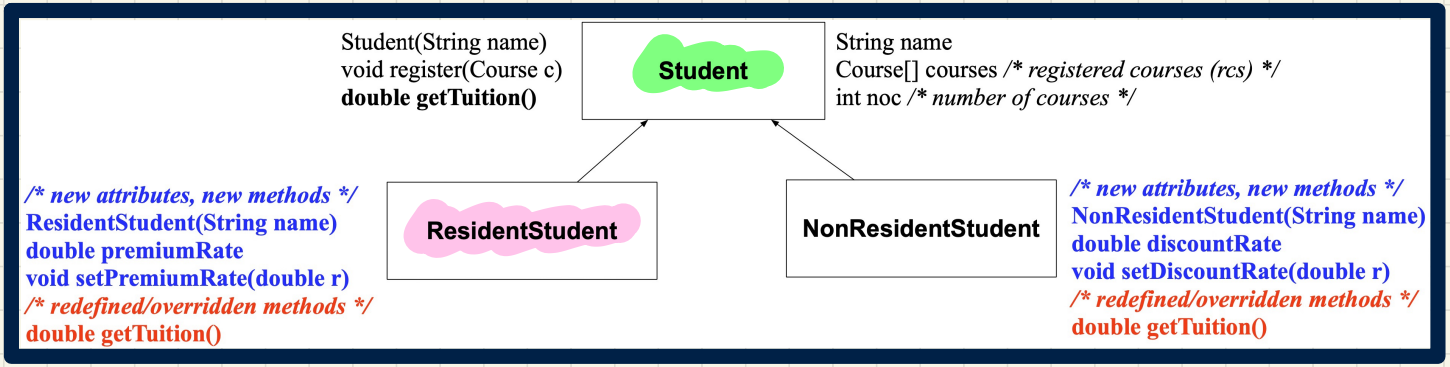





```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

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

Static Types, Casts, Polymorphism (5)

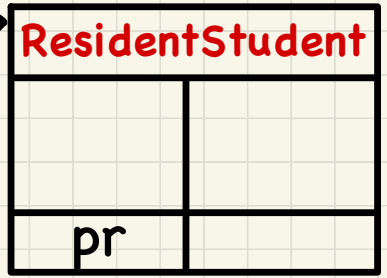



```
Course eecs2030 = new Course("EECS2030", 500.0);
Student s = new ResidentStudent("Jim");
s.register(eecs2030);
if(s instanceof ResidentStudent) {
  ((ResidentStudent) s).setPremiumRate(1.75);
  System.out.println(((ResidentStudent) s).getTuition());
}
```

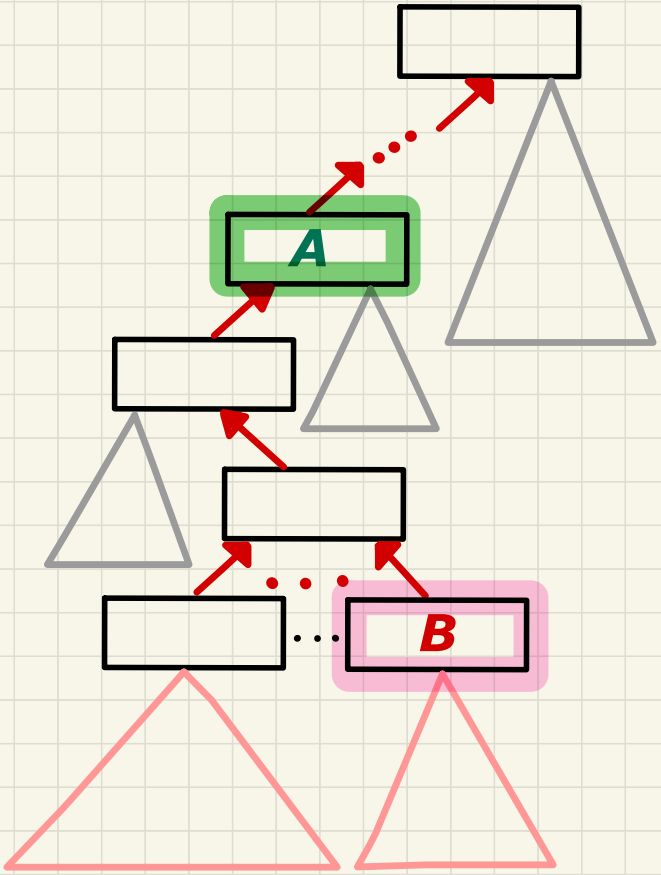
true.

No CCE!

Student s



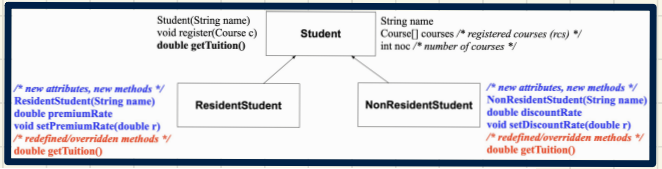
The instanceof Operator




```
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 ??.

Polymorphic Parameters (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]?

call by value: rs = 0; ↳ ST of arg should be a descendant of ST of param. rs.

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

∵ ST of param rs (RS) is a descendant of ST of ss[c] (Stud.)

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

sms.addRS(o)