Lecture 20 - Nov 21

Inheritance

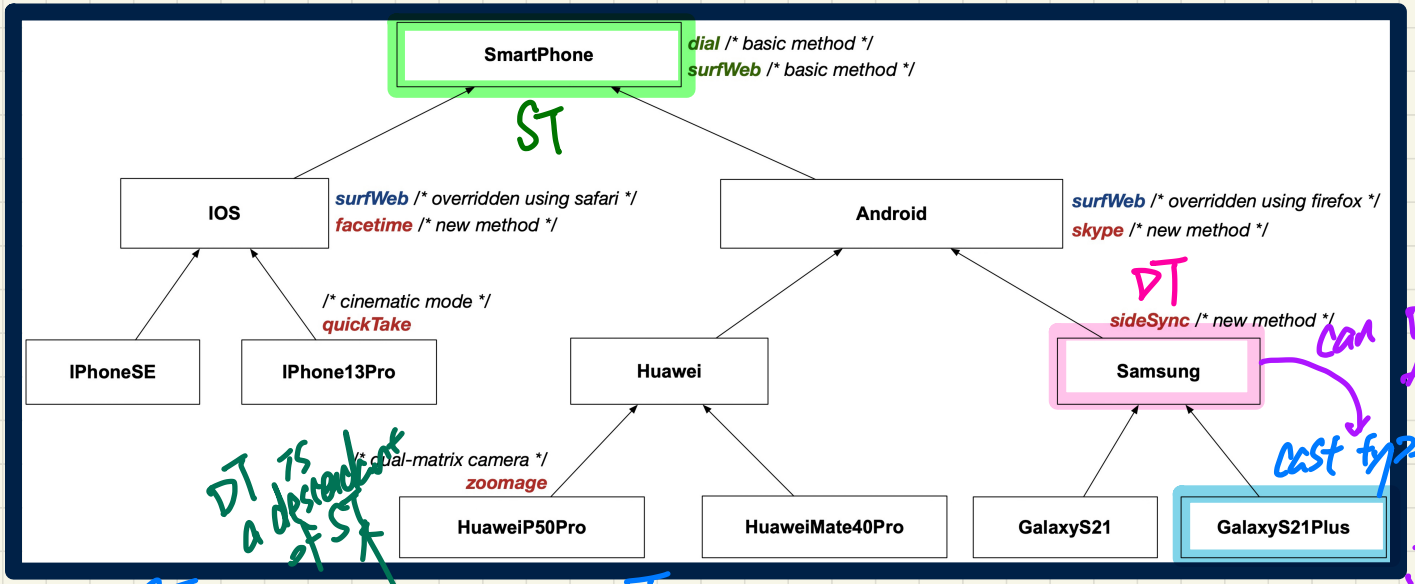
Type Cast: Compilable vs. Exception-Free
Checking DTs: instanceof Operator
Polymorphic Method Parameters

basis for ProgTest3

Announcements

- Lab4 due soon! tmrw.
- WT3 and ProgTest3 approaching...

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



Can DT fulfill cast type? No!

DT is a descendant of ST

∴ DT is not a descendant of cast type.

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

① compile? ✓ downward

② ClassCastException?

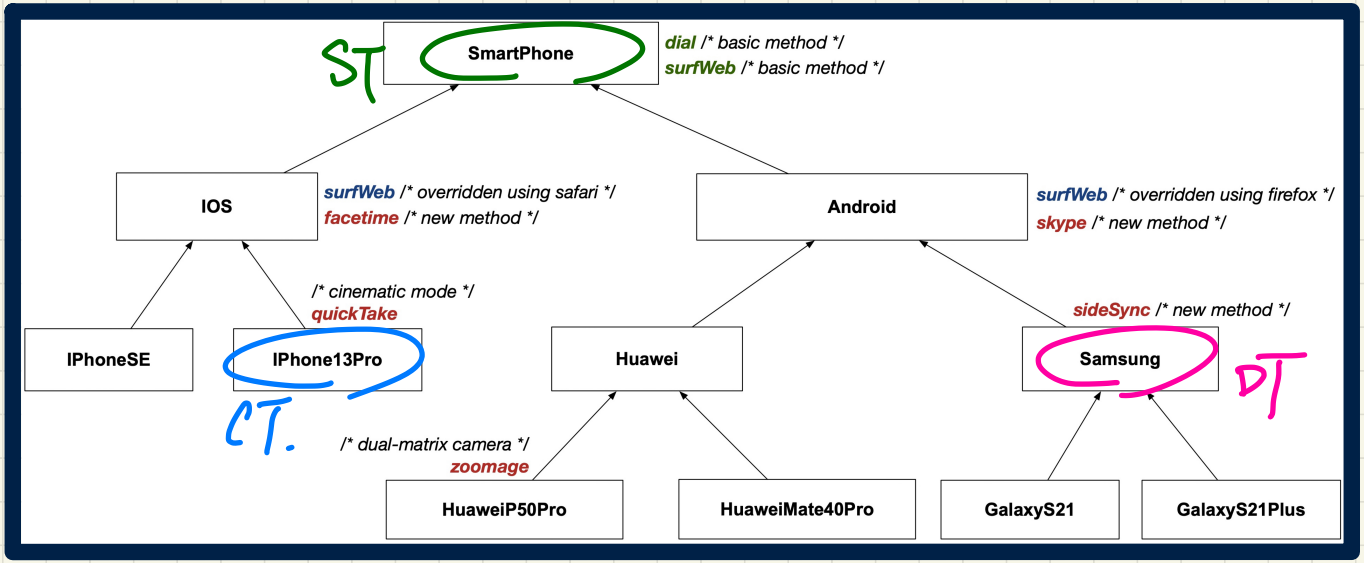
Compiler: ∴ cast type is a descendant of the ST.

Compilable? **ClassCastException at runtime?**

checked by instanceof operator

Can the DT fulfill the cast type?

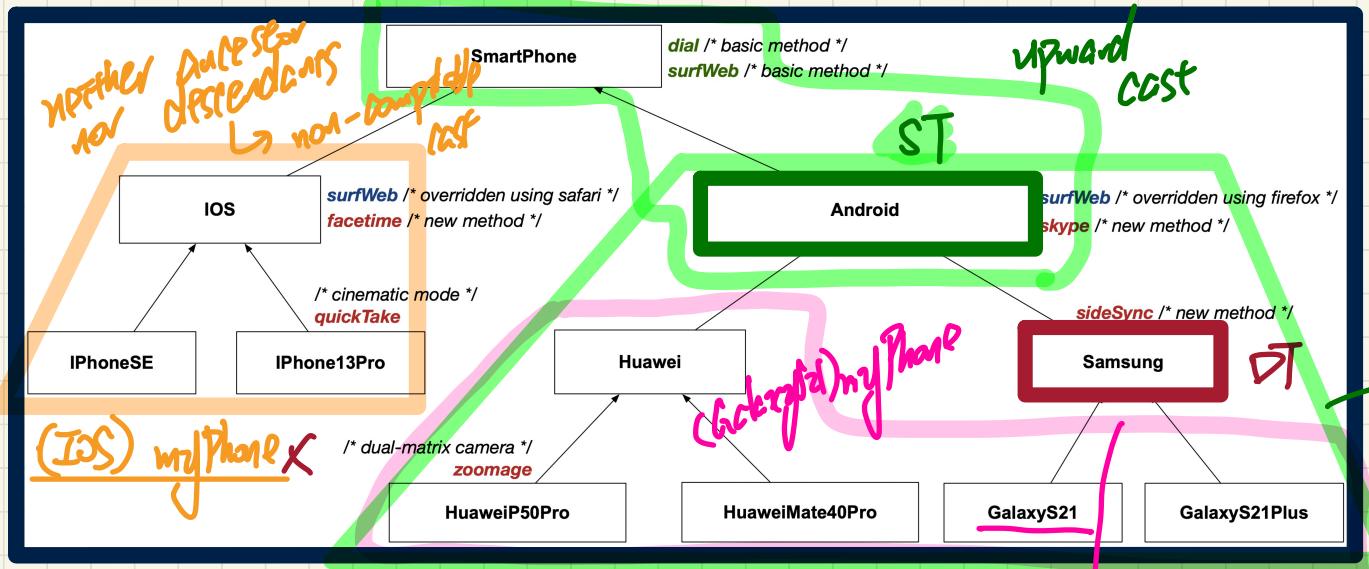
Exercise: Compilable Type Cast? Fail at Runtime? (2)




```
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

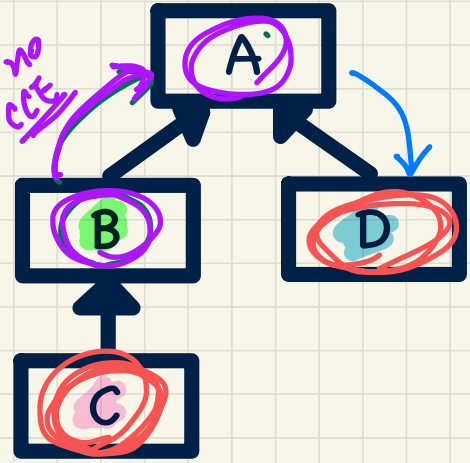
ClassCastException

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;
```

non-compilable cost.

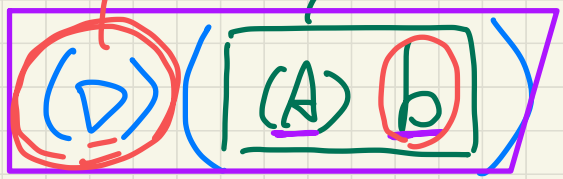


no CCE

at runtime

CCE : DT of b (which is C) is not a descendant of the cast type (D).

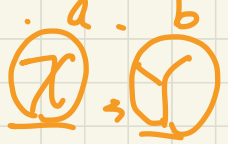
ST: A



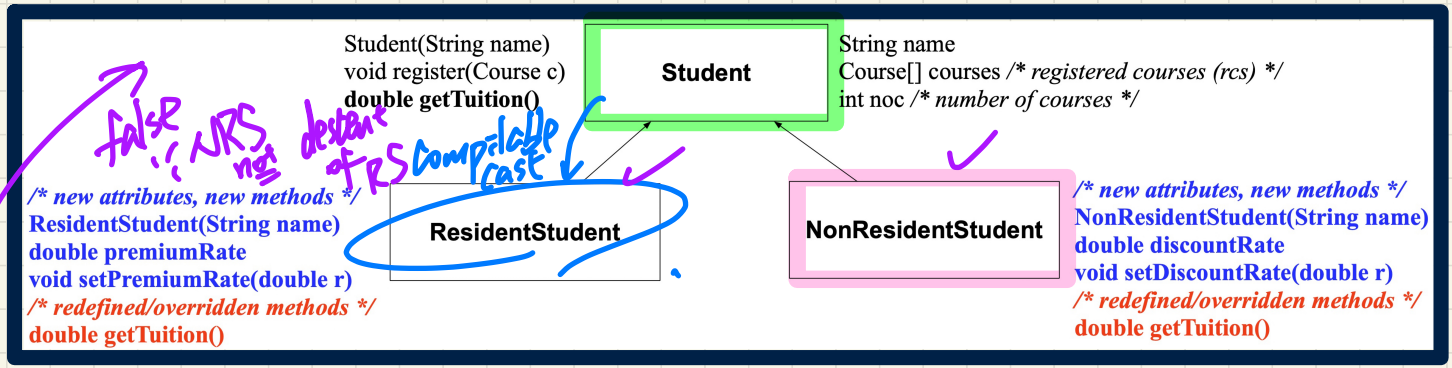
Compilable!

given any two classes X, Y

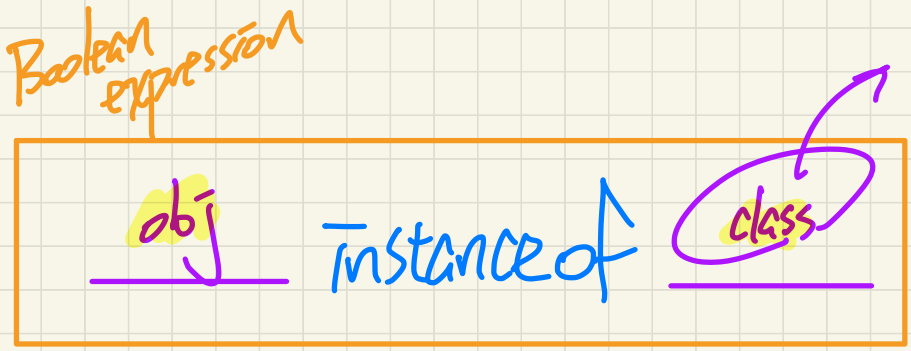
(Y) (Object) a



Checking Dynamic Types at Runtime (1)



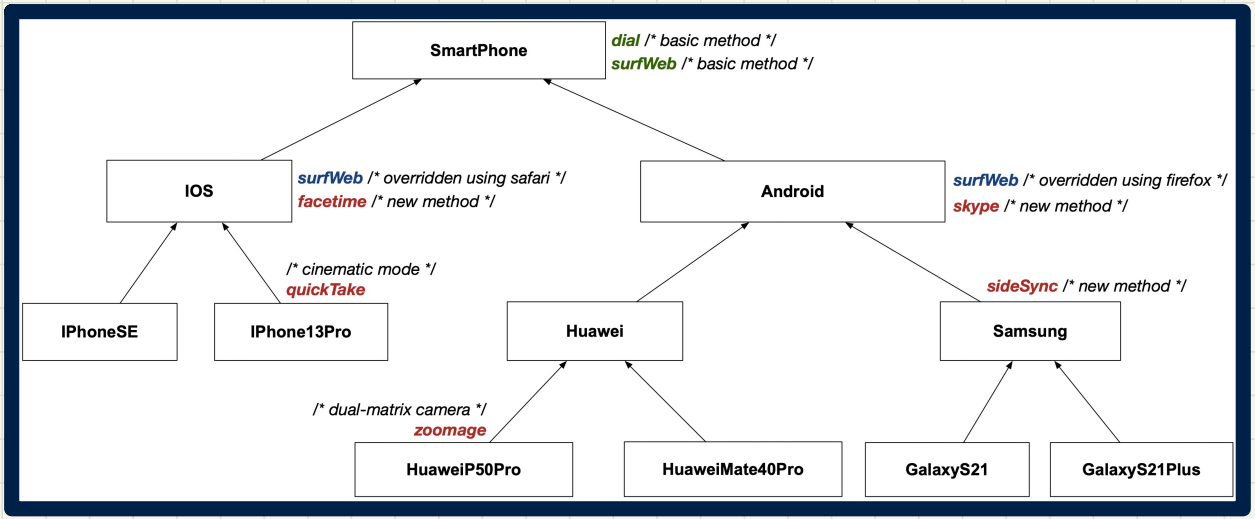

```
1  Student jim = new NonResidentStudent("J. Davis");
2  if (jim instanceof ResidentStudent) {
3    ResidentStudent rs = (ResidentStudent) jim;
4    rs.setPremiumRate(1.5);
5  }
```



↳ True if: ① DT of obj is a descendant of class

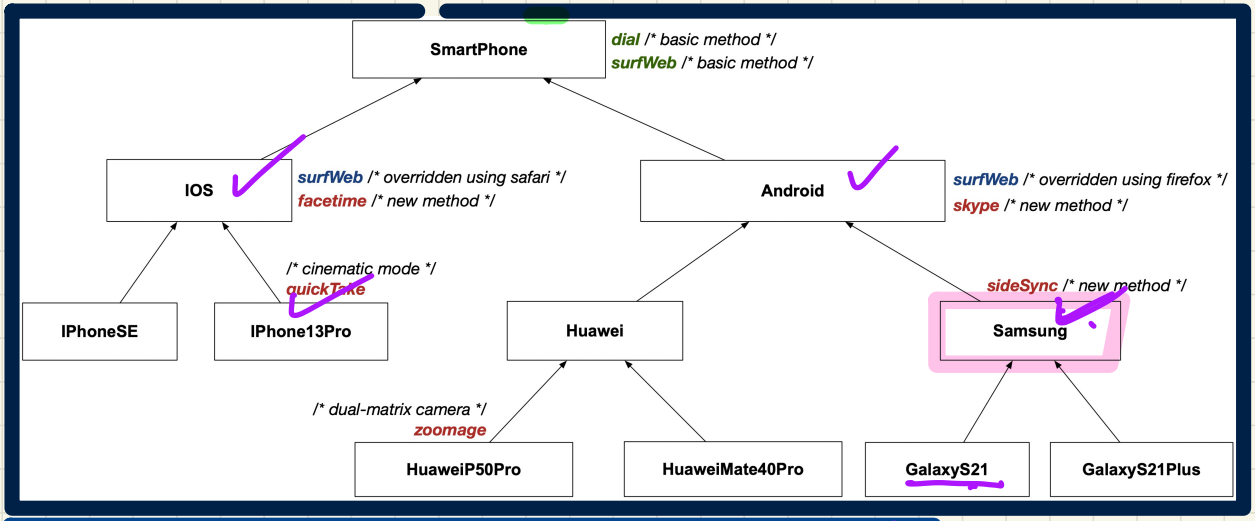
② DT of obj can fulfill expectation of class

Checking Dynamic Types at Runtime (2) (exercise)




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

Use of the instanceof Operator



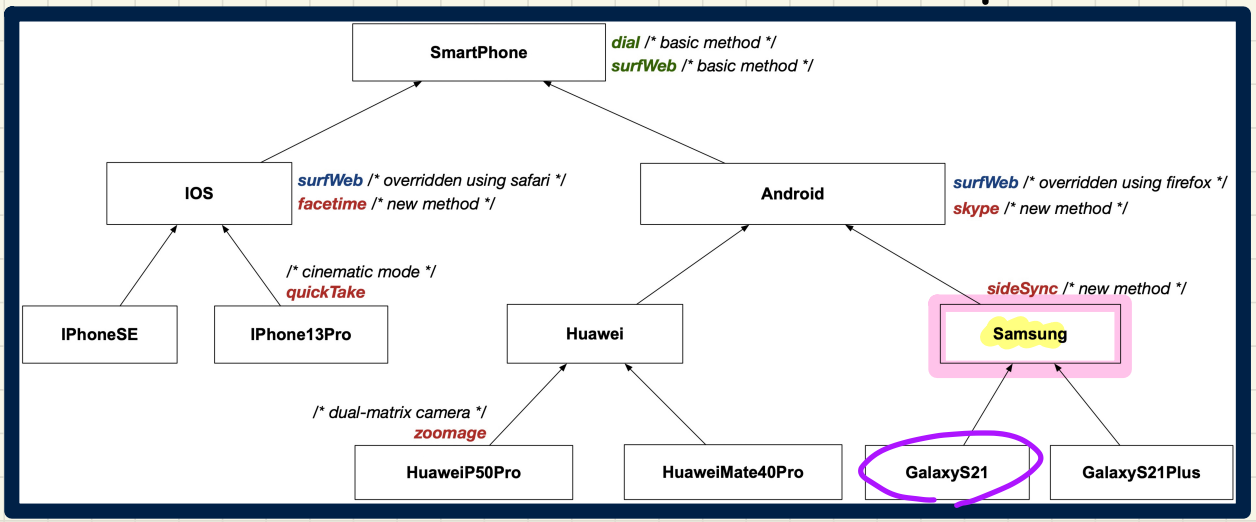

```
SmartPhone myPhone = new Samsung();
println(myPhone instanceof Android);
println(myPhone instanceof Samsung);
println(myPhone instanceof GalaxyS21);
println(myPhone instanceof IOS);
println(myPhone instanceof IPhone13Pro);
```

T
T
F
F
F

DT of myPhone (Samsung) not present of GalaxyS21

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

Safe Cast via Use of the instanceof Operator



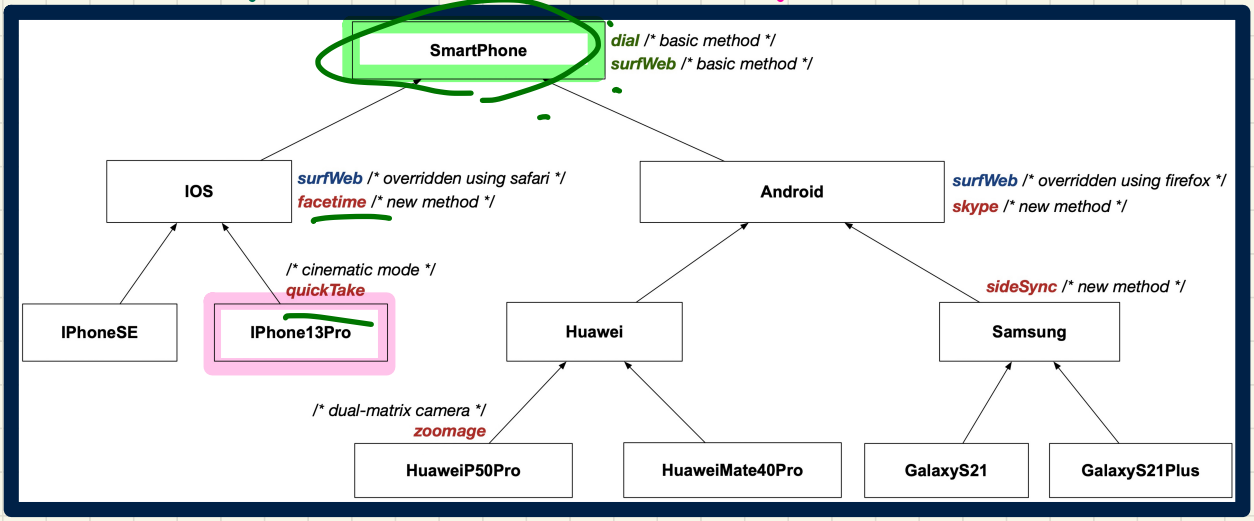

```
1  SmartPhone myPhone = new Samsung();
2  /* ST of myPhone is SmartPhone; DT of myPhone is Samsung */
3  if(myPhone instanceof Samsung) {
4    Samsung samsung = (Samsung) myPhone;
5  }
6  if(myPhone instanceof GalaxyS21Plus) {
7    GalaxyS21Plus galaxy = (GalaxyS21Plus) myPhone;
8  }
9  if(myphone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

cast done without CCE!

cast not reached ∵ instanceof evaluates to F.

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

Static Types, Casts, Polymorphism (1)

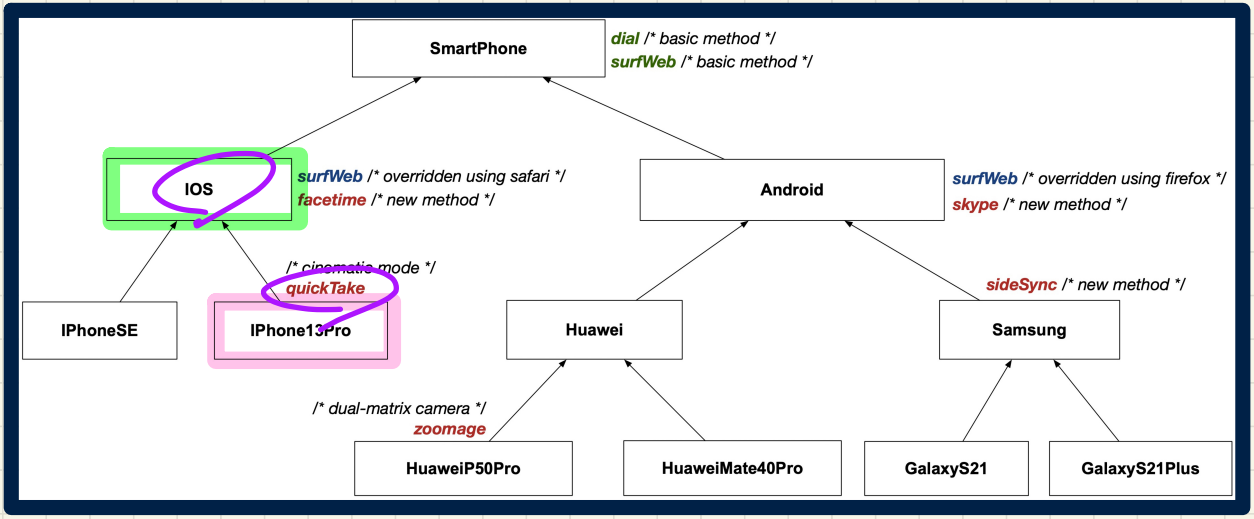



```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

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

ST: SmartPhone

Static Types, Casts, Polymorphism (2)

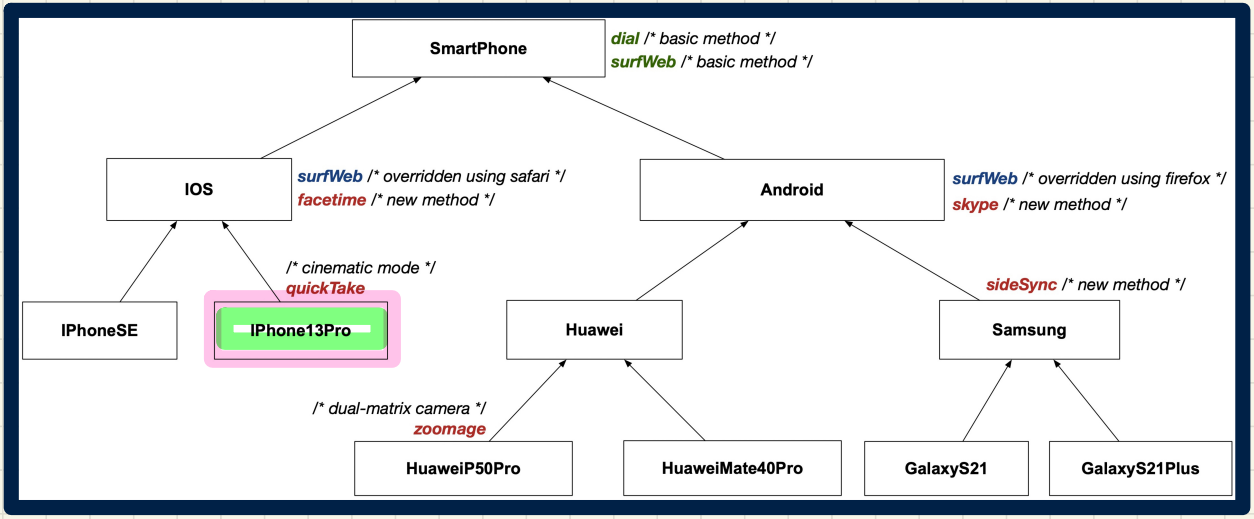



```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

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

✓ ✓ → ST IOS ✗

Static Types, Casts, Polymorphism (3)

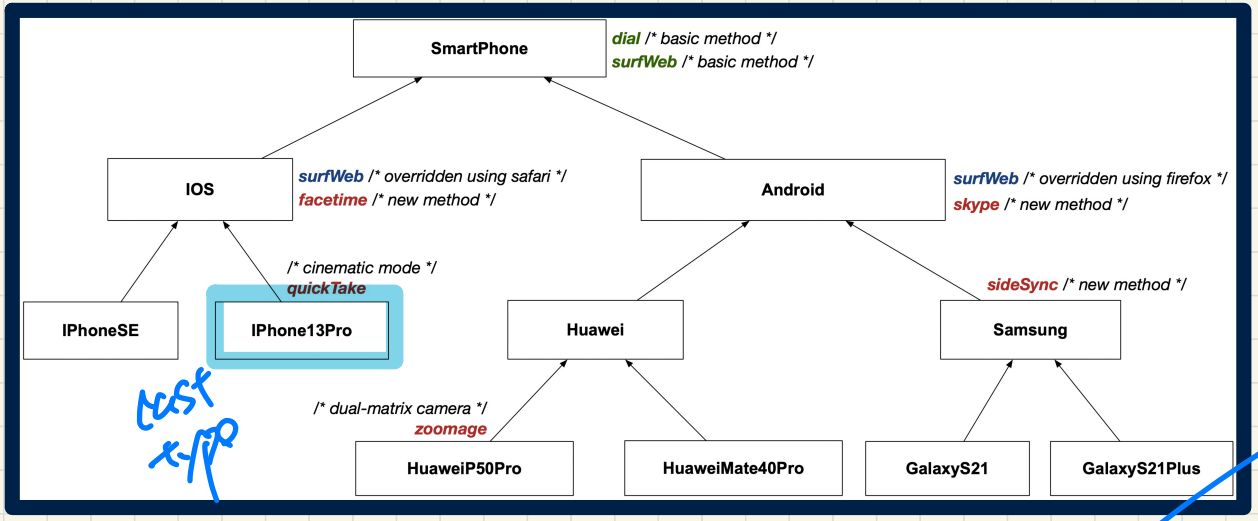



```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

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

allowed by ST.

Static Types, Casts, Polymorphism (4)



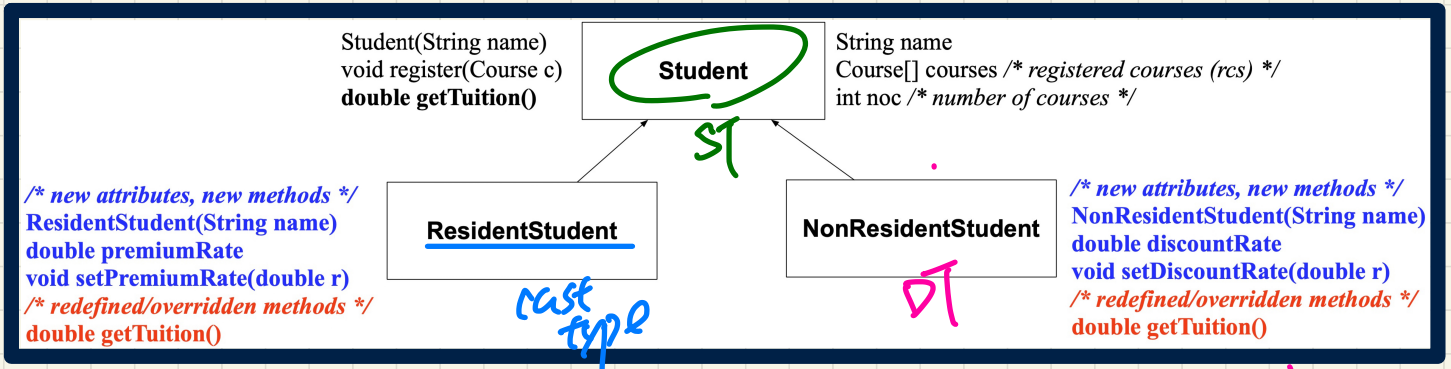

```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

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

case type
↑ a downcast
↳ no CCE.
DT

creating an alias of ST IPhone13Pro.

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());
}
```

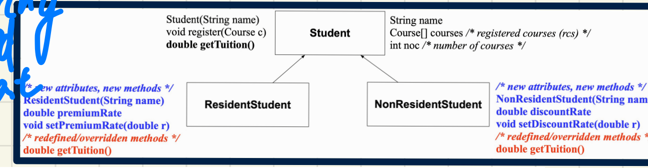
NonResidentStudent → False

Student s → ResidentStudent

pr	

Polymorphic Parameters (1)

① it has an associated type. hierarchy
② each element in array has declared type Student




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

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

Student

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

ST: Student → ss[c]; ST: RS → rs

valid ∵ ST of RHS descendant of ST of LHS.

call by value: rs = o → para. = arg.

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

sms.addRS(o)

what should be the type of o?

valid: ST of arg. o should be a descendant of para. rs