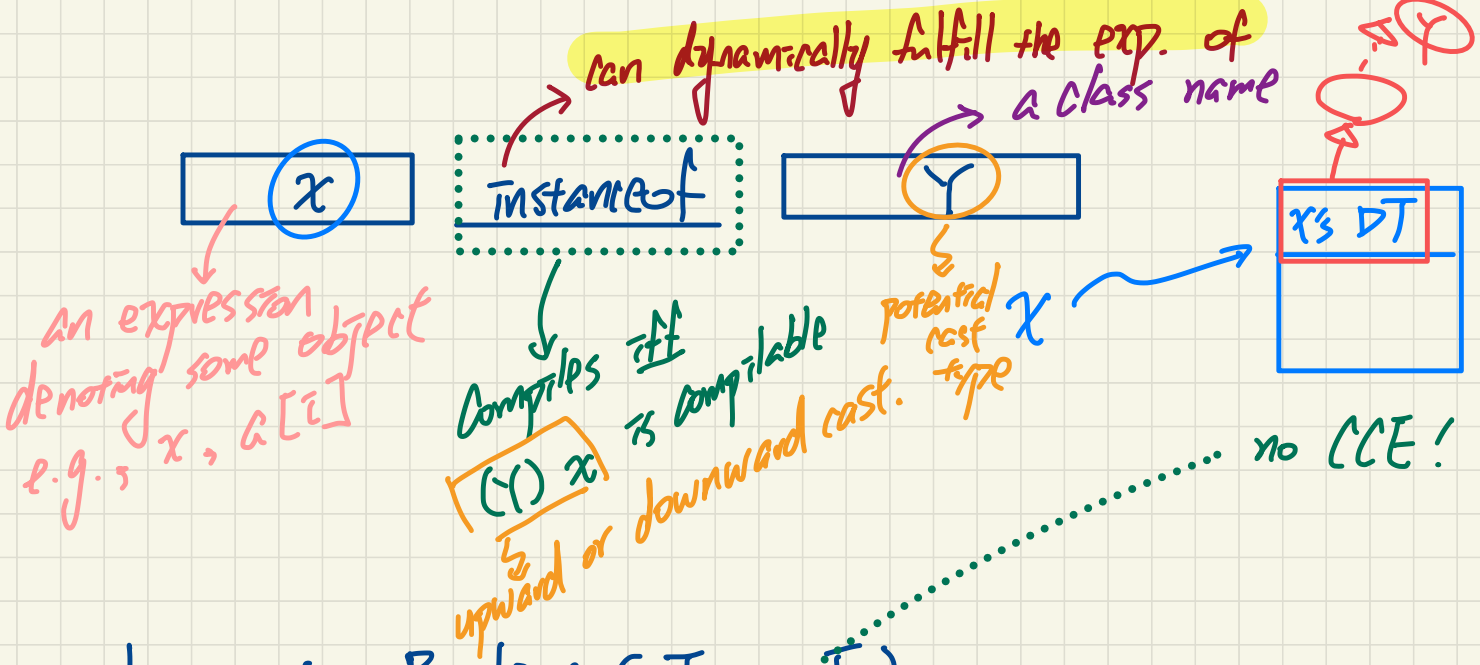
Lecture 20 - Nov 17

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



↳ evaluates to Boolean (T or F)

↳ True if (1) DT of x can fulfill exp(Y)
(2) DT of x is descendant of Y
(3) Y is an ancestor of x's DT

false otherwise

↳ if true then it's safe to execute (Y) x

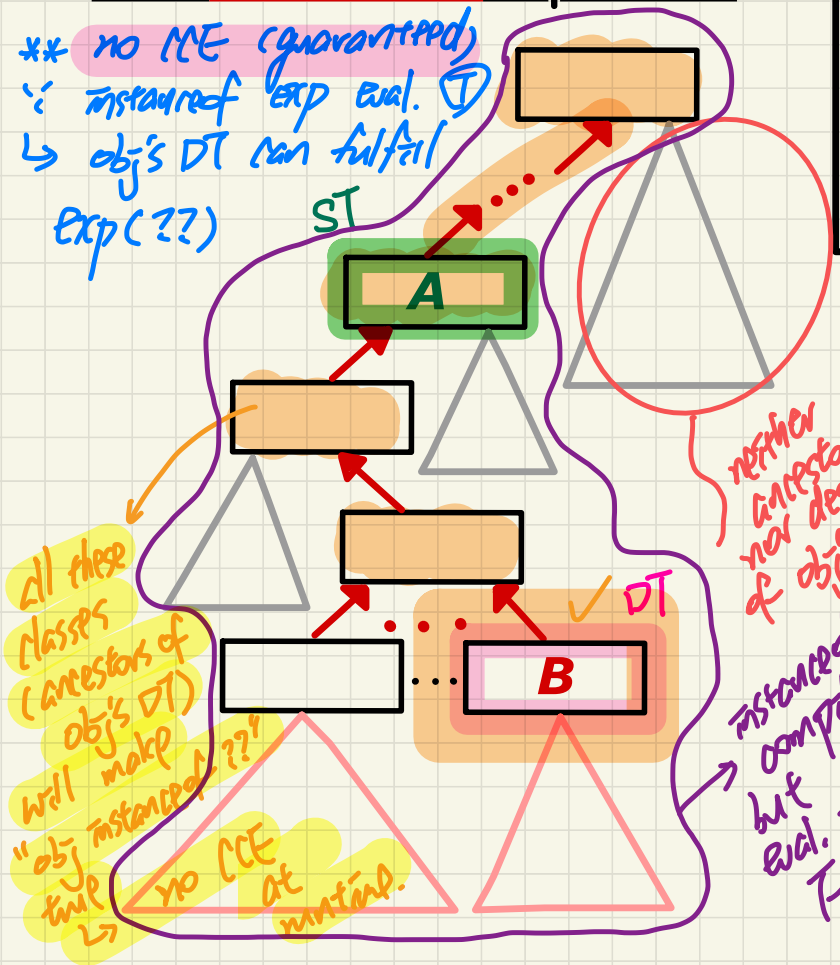
The instanceof Operator

* True iff obj's DT can fulfill exp(??)

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

* assume: obj instanceof ?? eval to True.

** no CCE (guaranteed)
∵ instanceof exp eval. T
↳ obj's DT can fulfill exp(??)

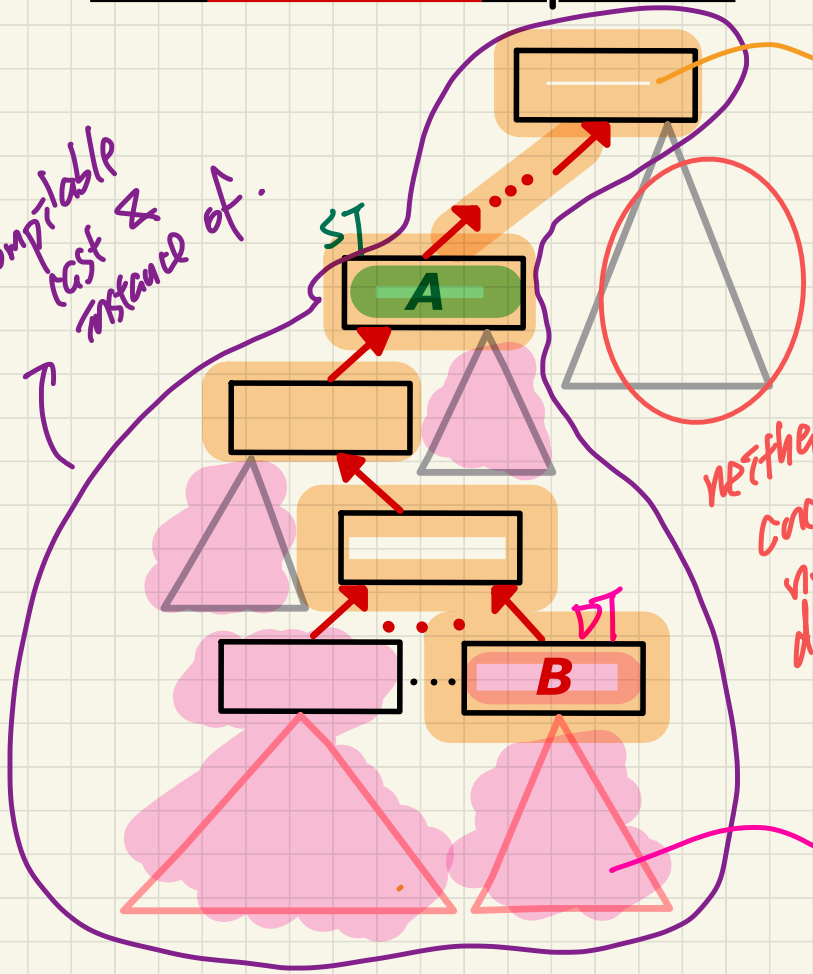


- L1 compiles if B can fulfill expectations of A.

- 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 ??.

The instanceof Operator



obj instanceof ?? → true.

```
A obj = new B;
if( obj instanceof ?? )
```

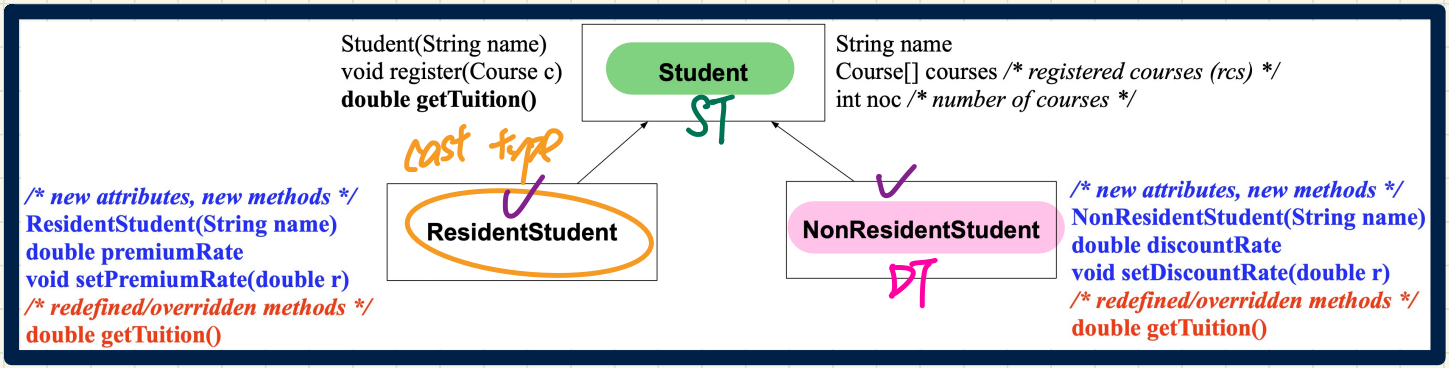
↳ compilable if ?? is either ancestor or descendant of A.

neither ancestor nor descendants (non compilable)

compilable cast & instanceof.

not ancestors of DT B → instanceof evals. to false.

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  }
```

false

bypassed → CCE prevented.

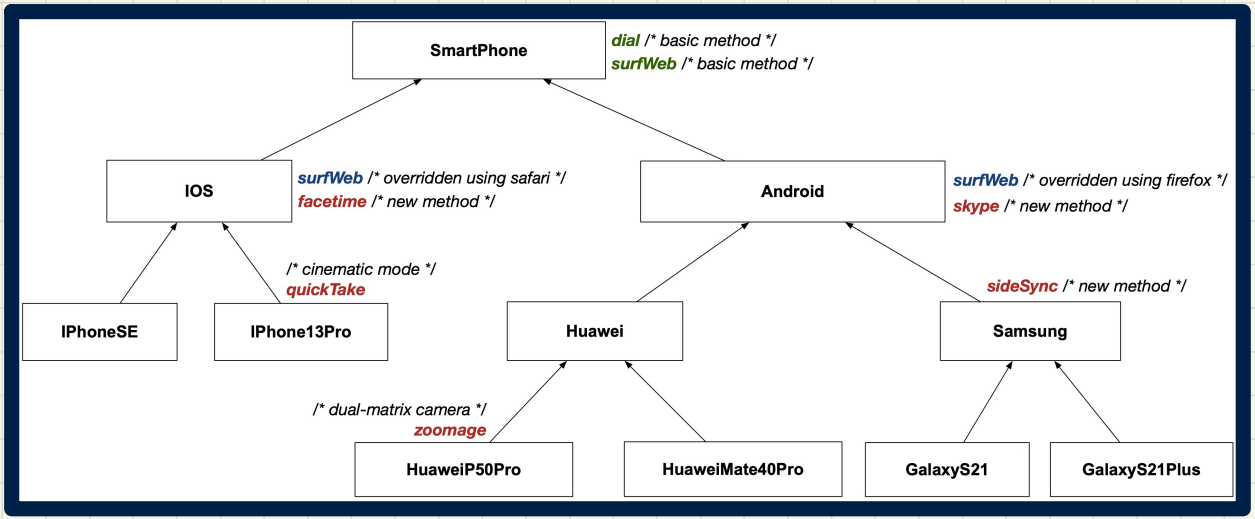
ST: S

1. instanceof exp. (RS) jim. compiles
2. eval. to false -: jim's DT not descendant of RS

1. compiles -: downward cast
2. CCE -: DT of jim (NRS) cannot fulfil the cast type (RS).

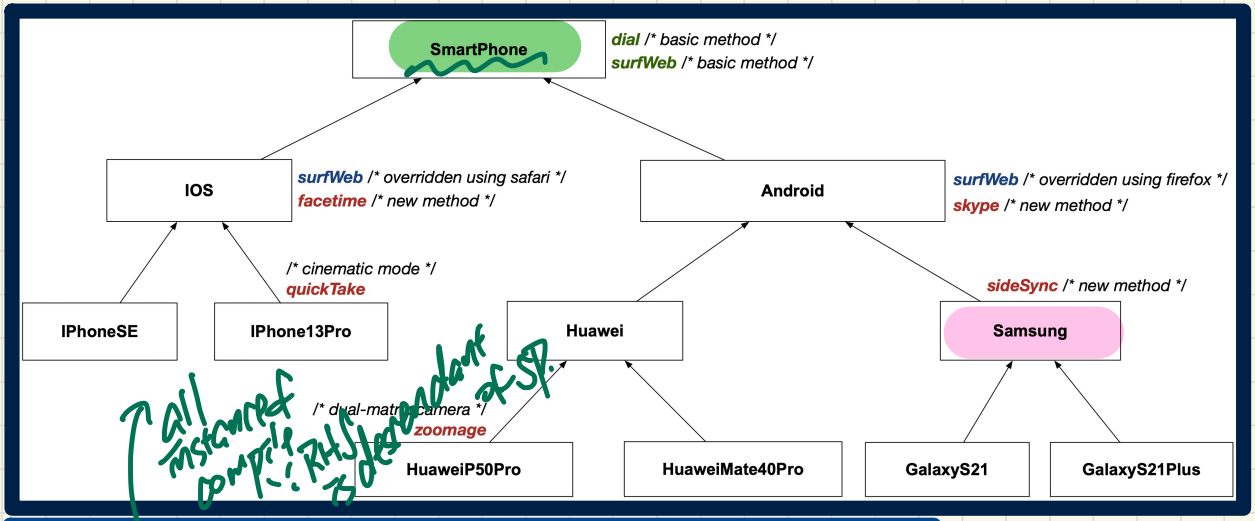
Checking Dynamic Types at Runtime (2)

Exercise




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

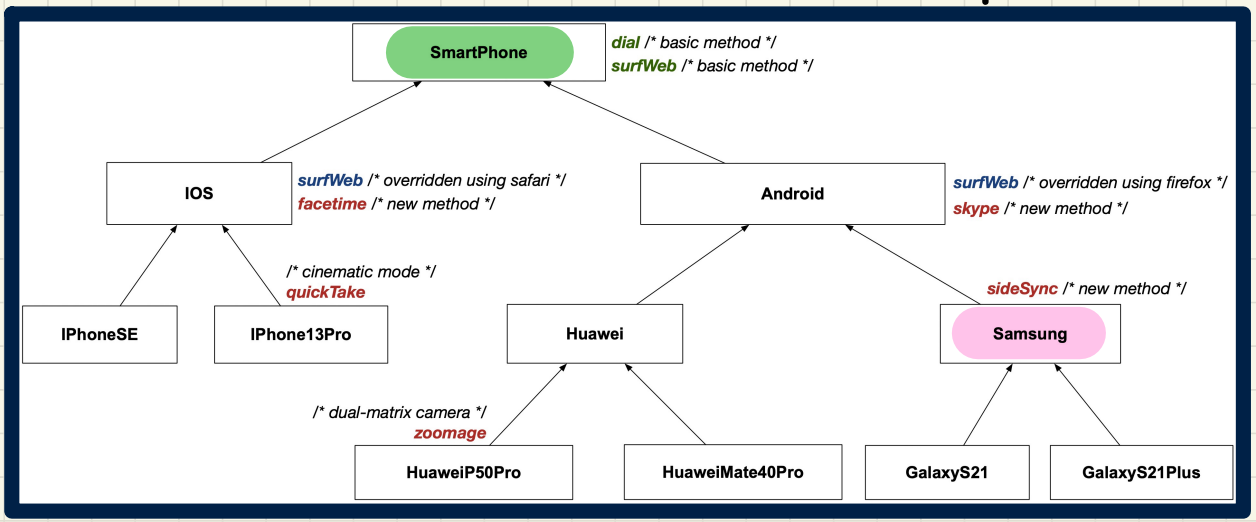
Use of the instanceof Operator




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

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 }
```

DT

no CCE!

False

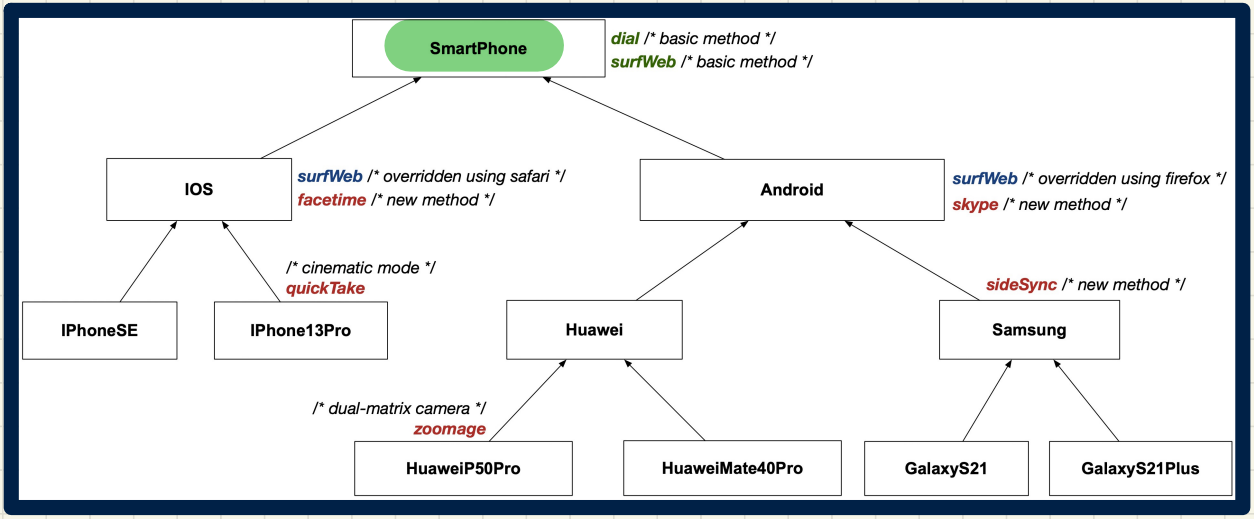
False

bypassed

prevent CCE.

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

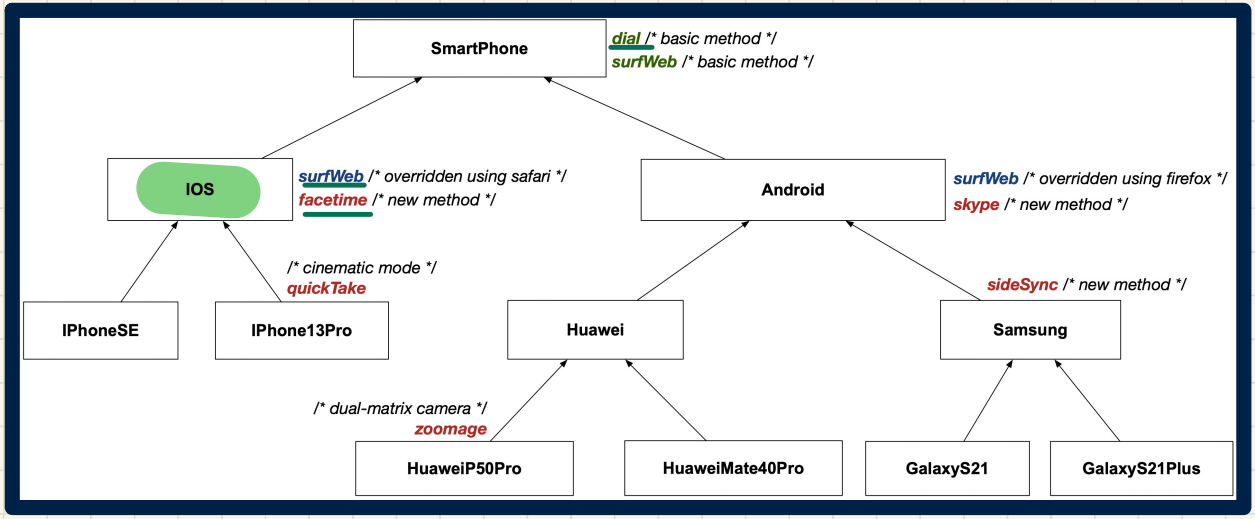
Static Types, Casts, Polymorphism (1)




```
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)

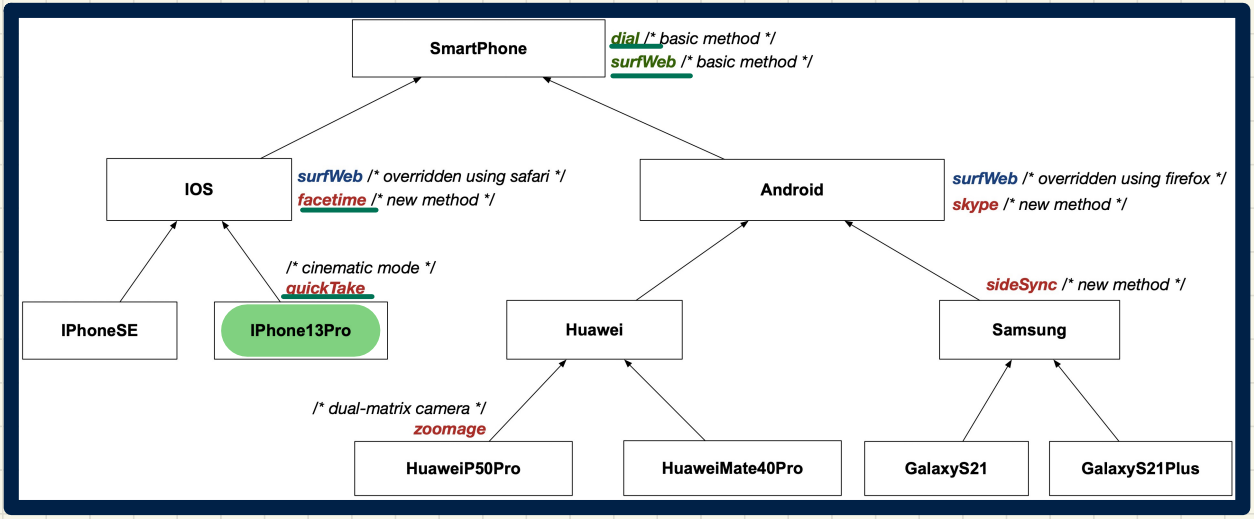



```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

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

ST: IOS.

Static Types, Casts, Polymorphism (3)

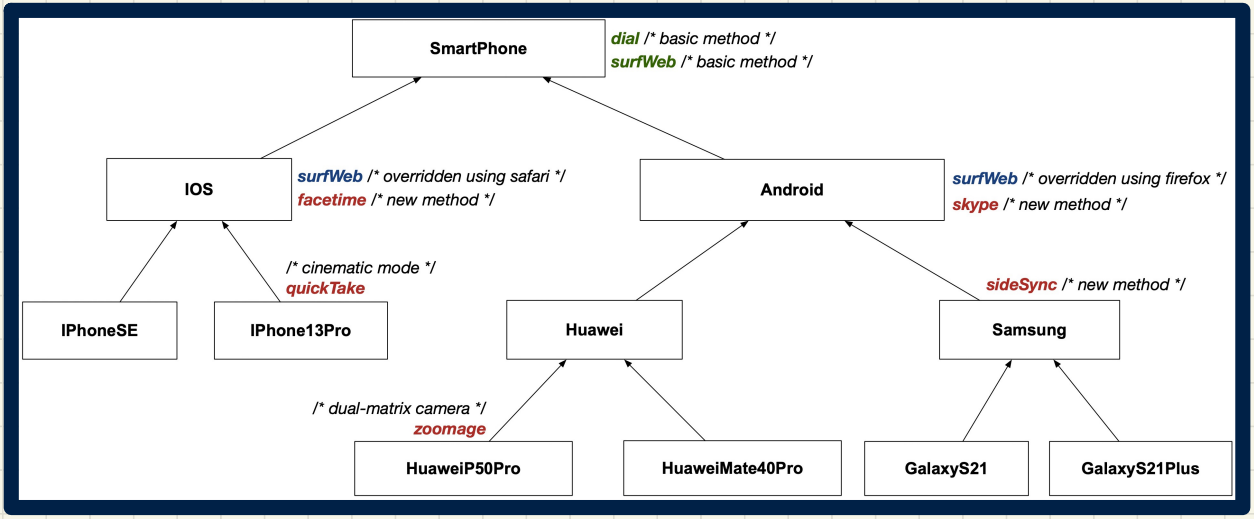



```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

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

ST: IP13Pro.

Static Types, Casts, Polymorphism (4)

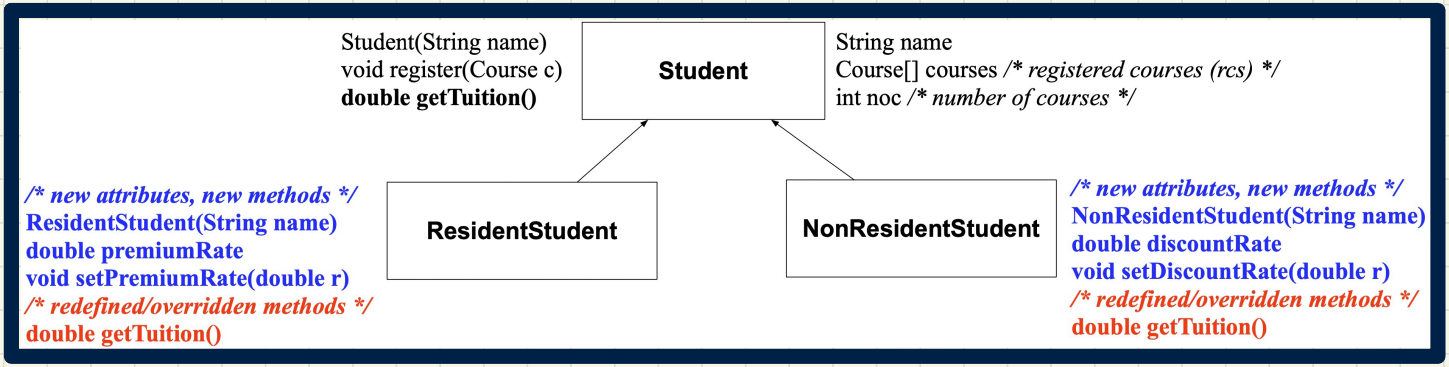



```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

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

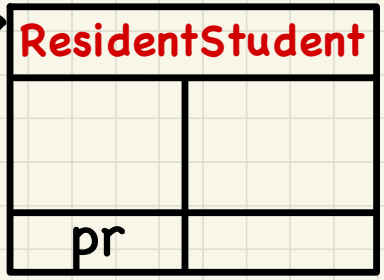
type cast exp. of ST IP13Pro.

Static Types, Casts, Polymorphism (5)



CCE here guaranteed C: instance of check

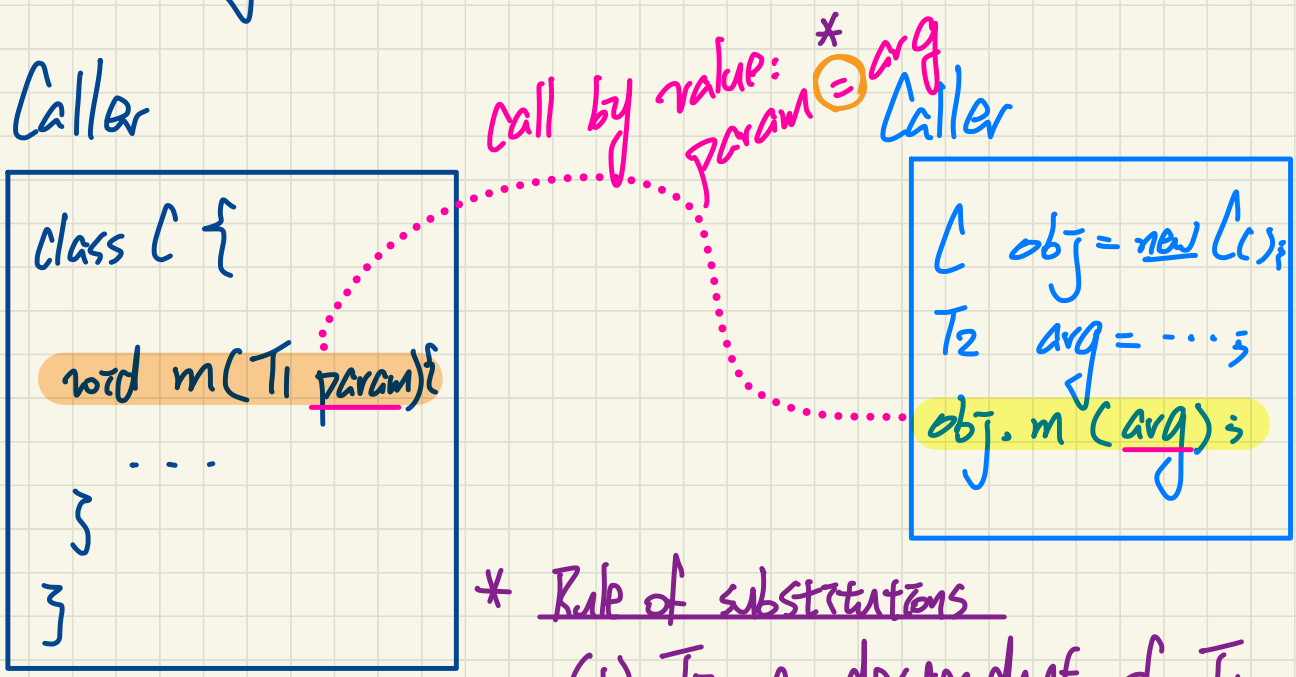
Student s




```
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

Call by Value

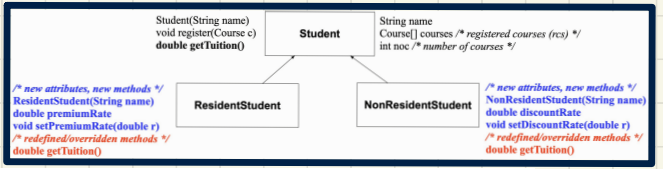


* Rule of substitutions

(1) T_2 a descendant of T_1

(2) T_2 fulfills exp(T_1)

Polymorphic Parameters (1)



ST of each element in the array.

```
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: S. ST: S. ST: S.

call by value: rs = o

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

ST: RS — ST must fulfill: RS (descendant of) — ST: Student — ST: RS — Yes ∵ RS descendant of Student.

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

ST: SMS ← sms.addRS(o) → in order for this method call to compile what should be ① about argument "o"?

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++; } }
```

```
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);
```