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

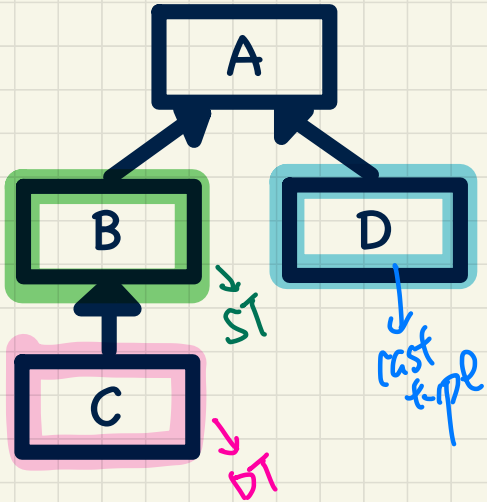
Exercise: Compilable Cast vs. Exception-Free Cast

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

PointV2 ((Object) obj)
String

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

→ cast not compilable ∵ neither upward nor downward



D d = (D) ((A) b)

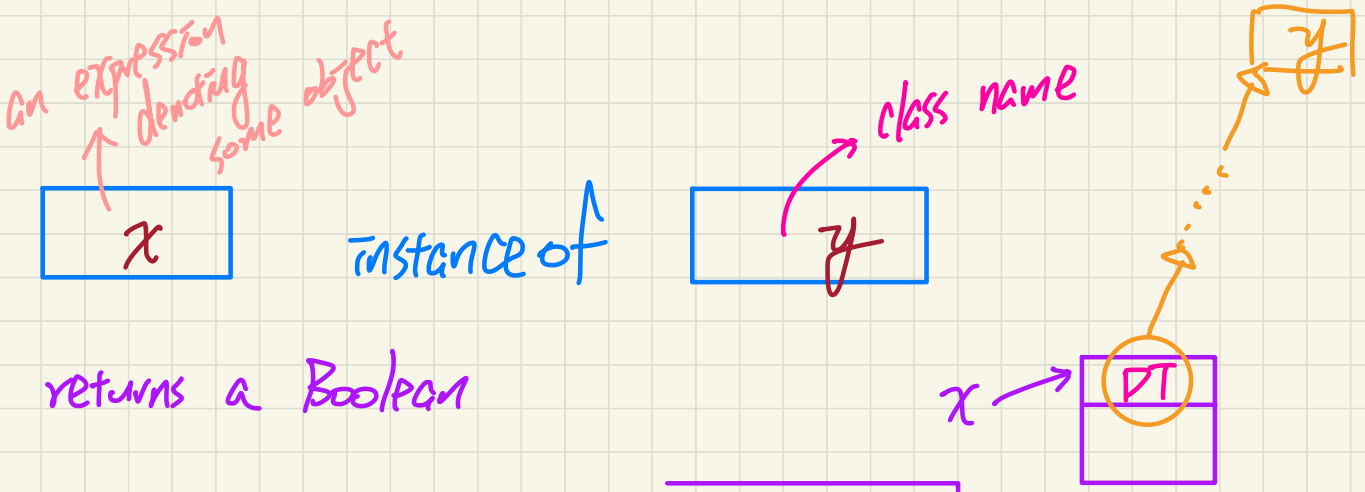
upward

downward

↳ ClassCastException

∵ DT C cannot fulfill exp. of cast type D

not a descendant of



↳ returns a Boolean

↳ true if x's DT can fulfil the exp. of y.

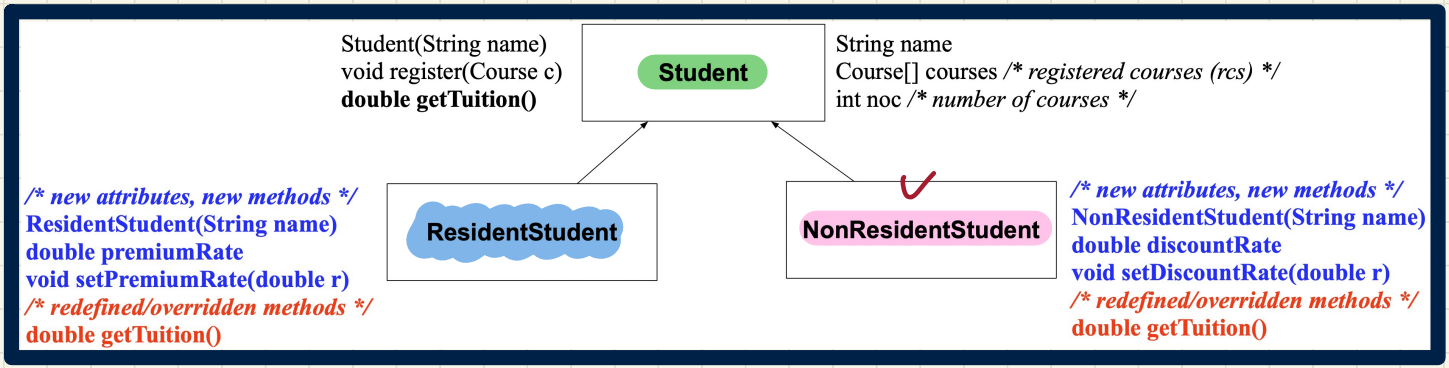
① x's DT is descendant of y

② y is ancestor of x's DT

↳ true means we can cast x to y (y) x

ClassCastException free

Checking Dynamic Types at Runtime (1)

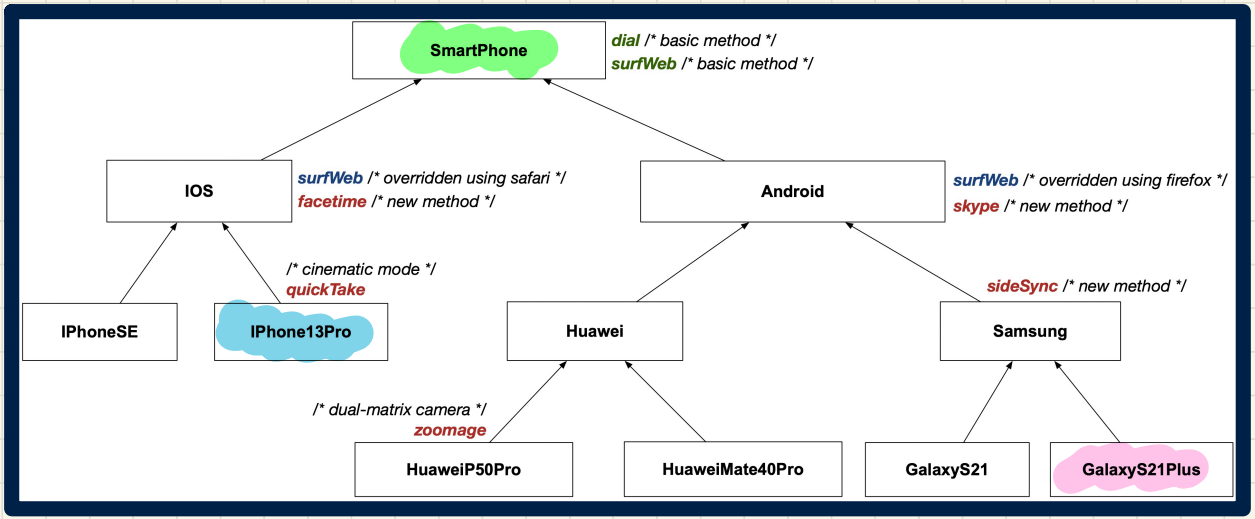



```
1  Student jim = new NonResidentStudent("J. Davis");
2  if (jim instanceof ResidentStudent) {
3    ResidentStudent rs = (ResidentStudent) jim;
4    rs.setPremiumRate(1.5);
5  }
```

↓ downward cast

* Can the DT of jim (NRS) fulfill exp of cast type RS?
  ↳ No → instanceof returns false

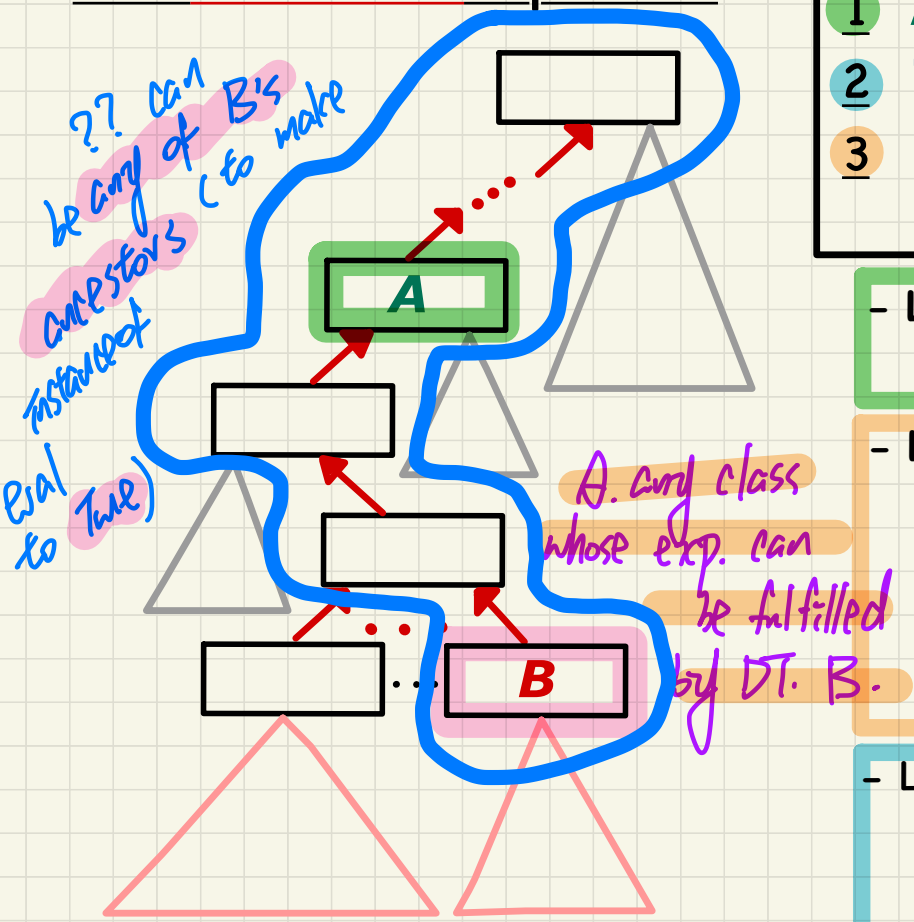
Checking Dynamic Types at Runtime (2)




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

true ⇒ safe to write (IP13Pro) aPhone

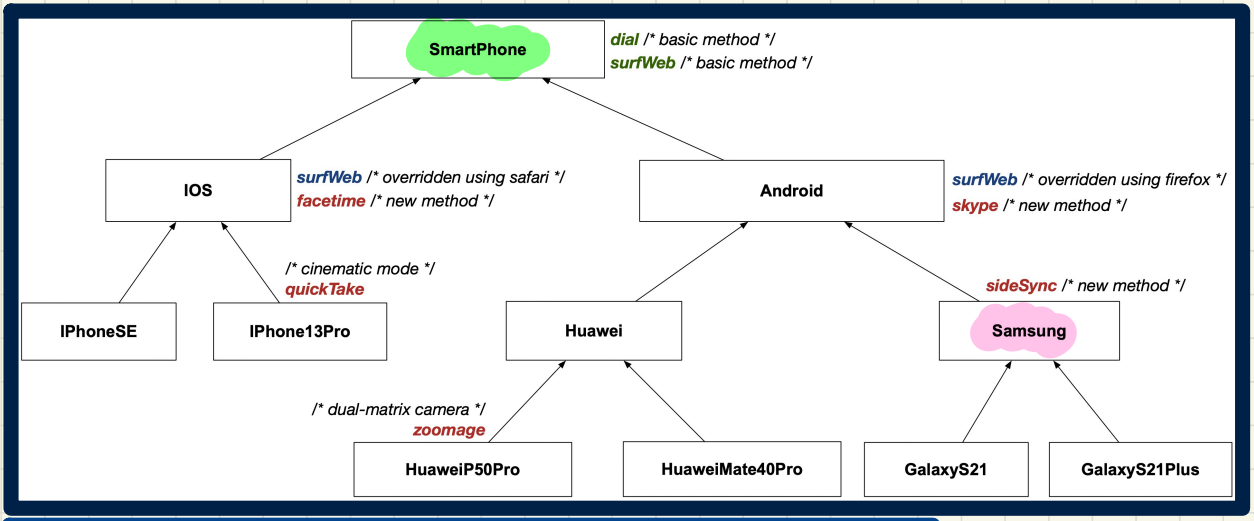
The instanceof Operator




```
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 ??.

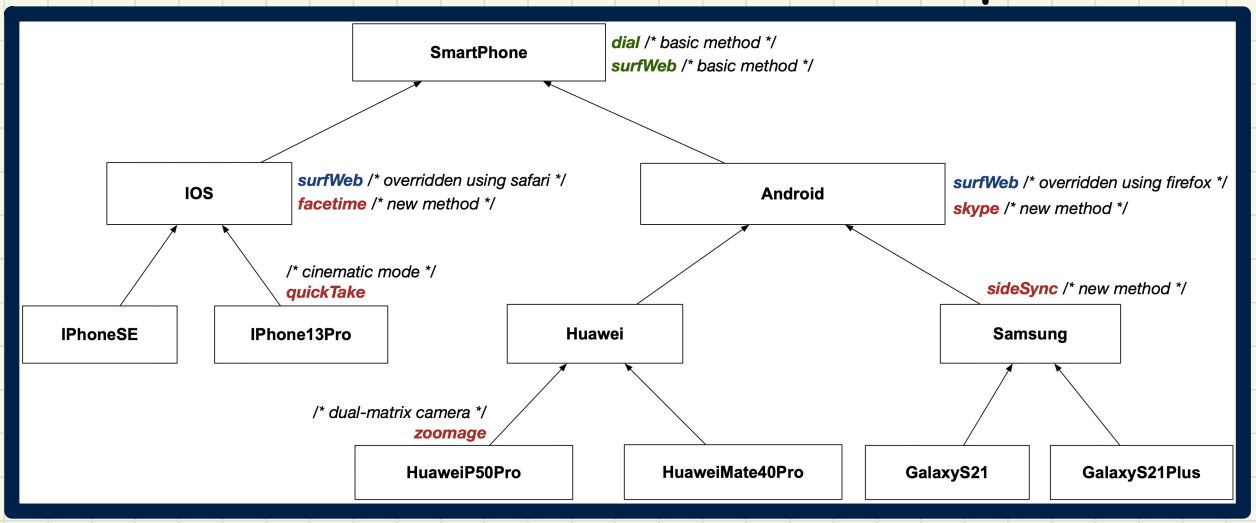
Use of the instanceof Operator




```
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 ??.

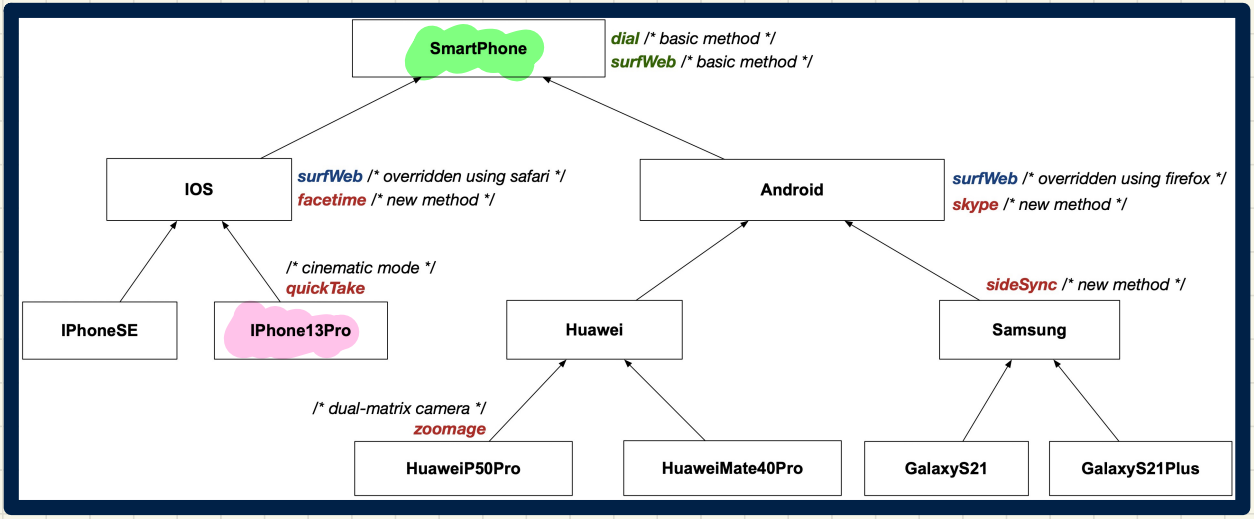
Safe Cast via Use of the instanceof Operator




```
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 }
```

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

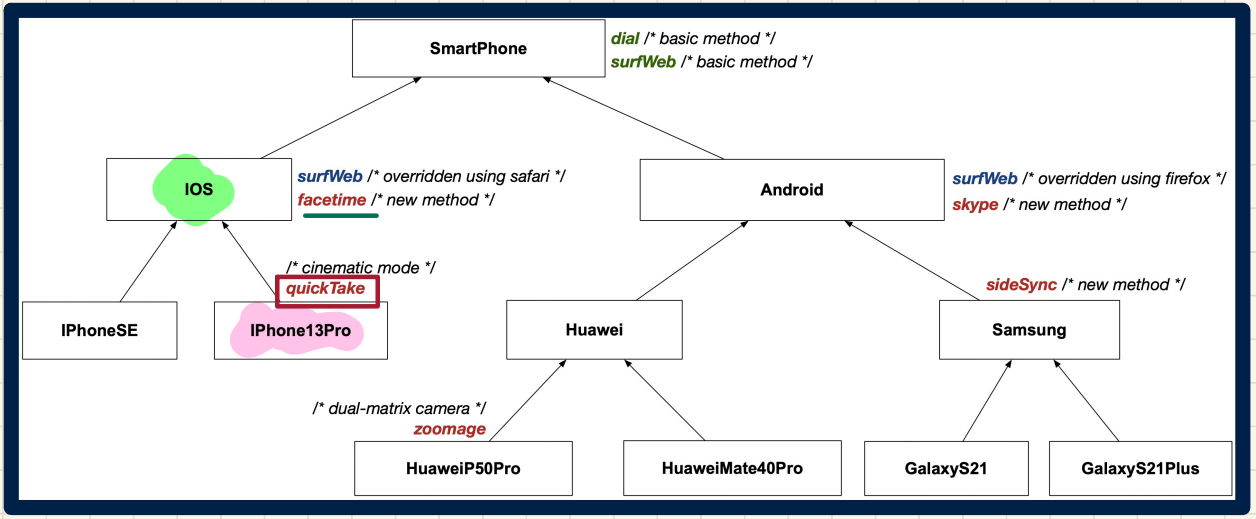
Static Types, Casts, Polymorphism (1)




```
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();
```

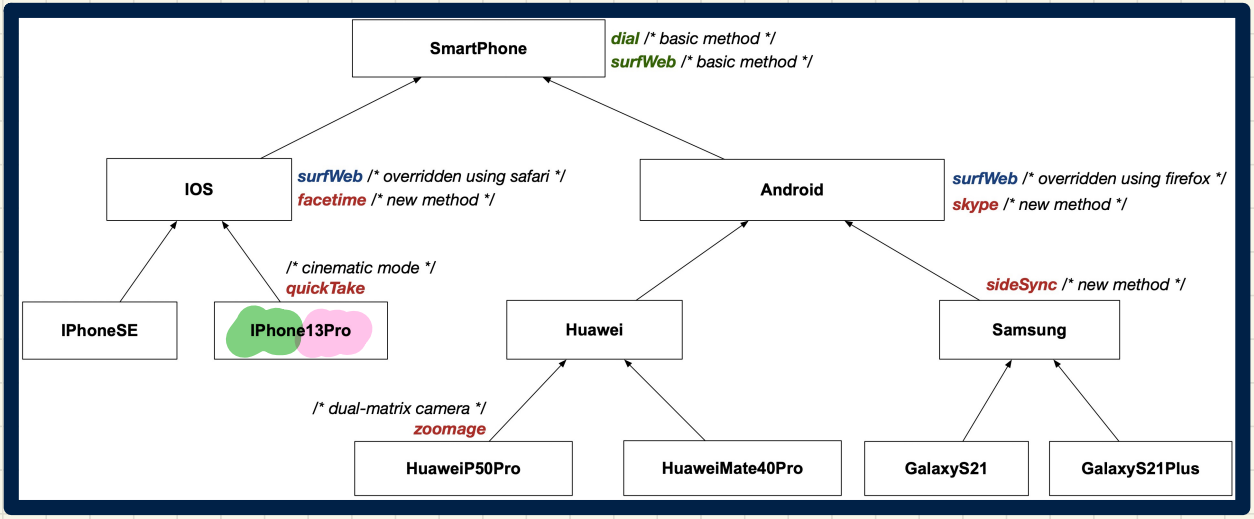
Static Types, Casts, Polymorphism (2)




```
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();
```

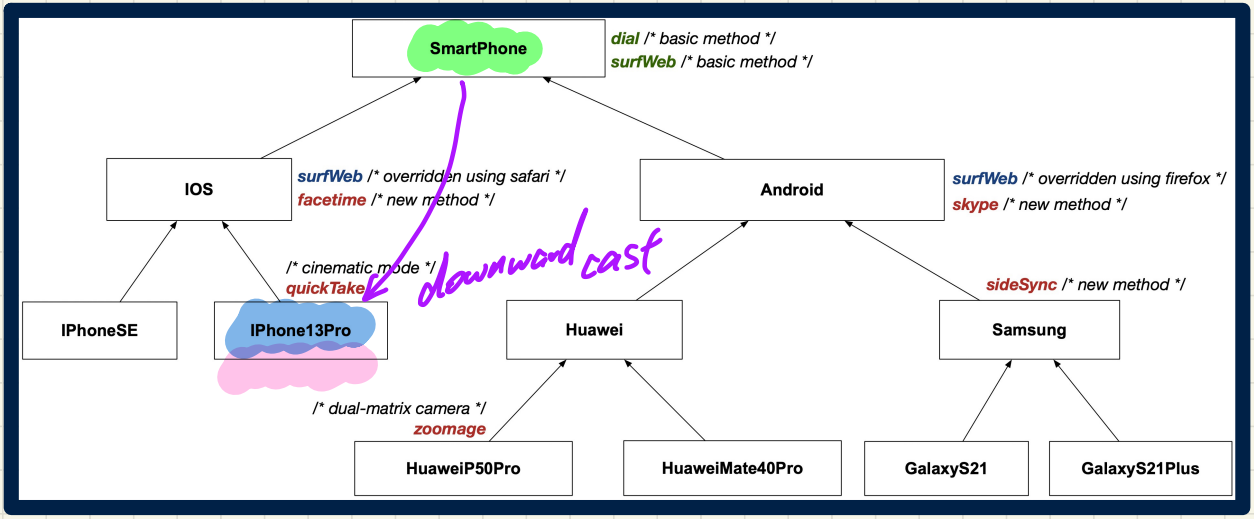
Static Types, Casts, Polymorphism (3)




```
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)

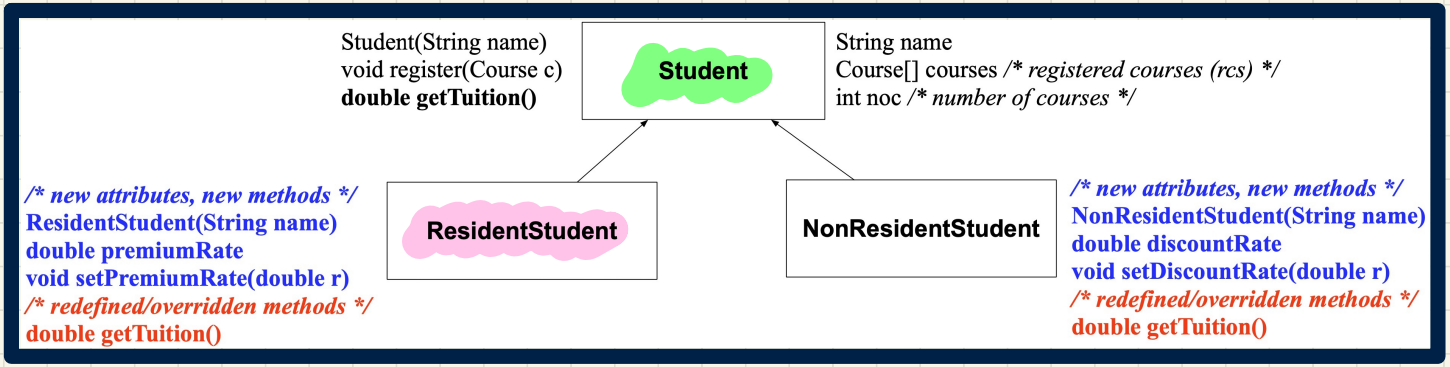



```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

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

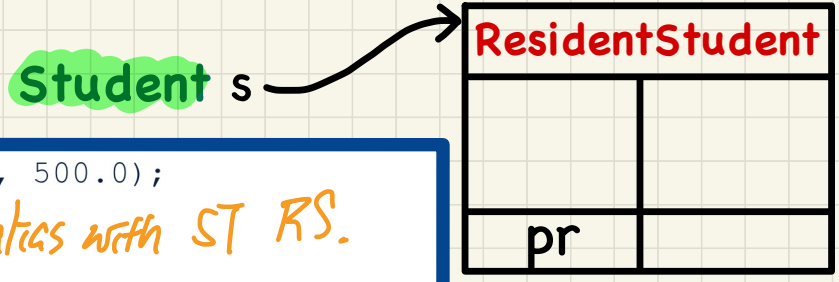
also with ST IP13Pro. (no CCE).

Static Types, Casts, Polymorphism (5)

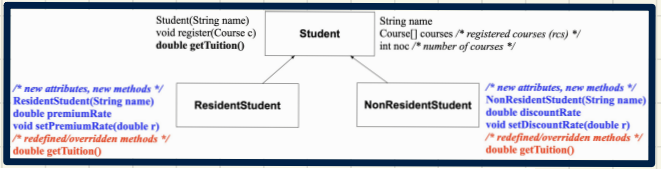



```
Course eecs2030 = new Course("EECS2030", 500.0);
Student s = new ResidentStudent("Jim");
s.register(eecs2030);
if(s instanceof ResidentStudent) {
  ((ResidentStudent) s).setPremiumRate(1.75);
  System.out.println(((ResidentStudent) s).getTuition());
}
```

aligns with ST RS.



Polymorphic Parameters (1)



ST of param. rs

```
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]?

Student.

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

ST: Student ST: RS

call by value: rs = 0 = param arg.

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

sms.addRS(o) → ST of arg. o should be a descendant of ST of param rs (RS).

Polymorphic Parameters (2)

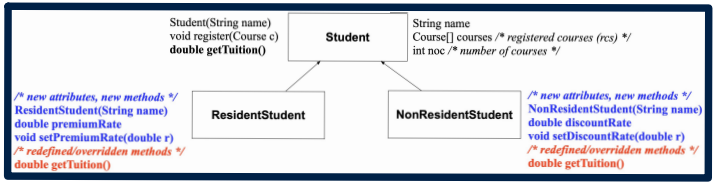
```
1  class StudentManagementSystem {
2    Student [] ss; /* ss[i] has static type Student */ int c;
3    void addRS(ResidentStudent rs) { ss[c] = rs; c ++; }
4    void addNRS(NonResidentStudent nrs) { ss[c] = nrs; c++; }
5    void addStudent(Student s) { ss[c] = s; c++; } }
```

call by value: rs = ?

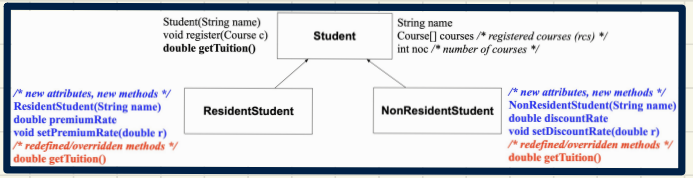
```
Student s1 = new Student();
Student s2 = new ResidentStudent();
Student s3 = new NonResidentStudent();
ResidentStudent rs = new ResidentStudent();
NonResidentStudent nrs = new NonResidentStudent();
StudentManagementSystem sms = new StudentManagementSystem();
sms.addRS(s1);   ✗
sms.addRS(s2);   ✗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

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

call by value: s = ?



Casting Arguments



void addRS(ResidentStudent rs)

sms.addRS((ResidentStudent) s) compiles?

```
1 Student s = new Student("Stella");
2 /* s' ST: Student; s' DT: Student */
3 StudentManagementSystem sms = new StudentManagementSystem();
4 sms.addRS(s); ×
```

ClassCastException?

```
1 Student s = new NonResidentStudent("Nancy");
2 /* s' ST: Student; s' DT: NonResidentStudent */
3 StudentManagementSystem sms = new StudentManagementSystem();
4 sms.addRS(s); ×
```

ClassCastException?

```
1 Student s = new ResidentStudent("Rachael");
2 /* s' ST: Student; s' DT: ResidentStudent */
3 StudentManagementSystem sms = new StudentManagementSystem();
4 sms.addRS(s); ×
```

ClassCastException?

sms.addRS((ResidentStudent) nrs) compiles?

```
1 NonResidentStudent nrs = new NonResidentStudent();
2 /* ST: NonResidentStudent; DT: NonResidentStudent */
3 StudentManagementSystem sms = new StudentManagementSystem();
4 sms.addRS(nrs); ×
```