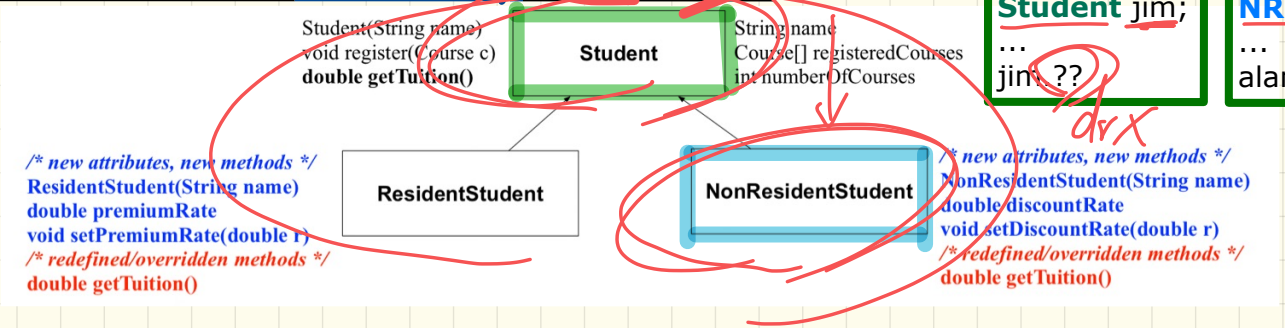
# Lecture 18

## Monday November 11

# Static Types determine Expectations

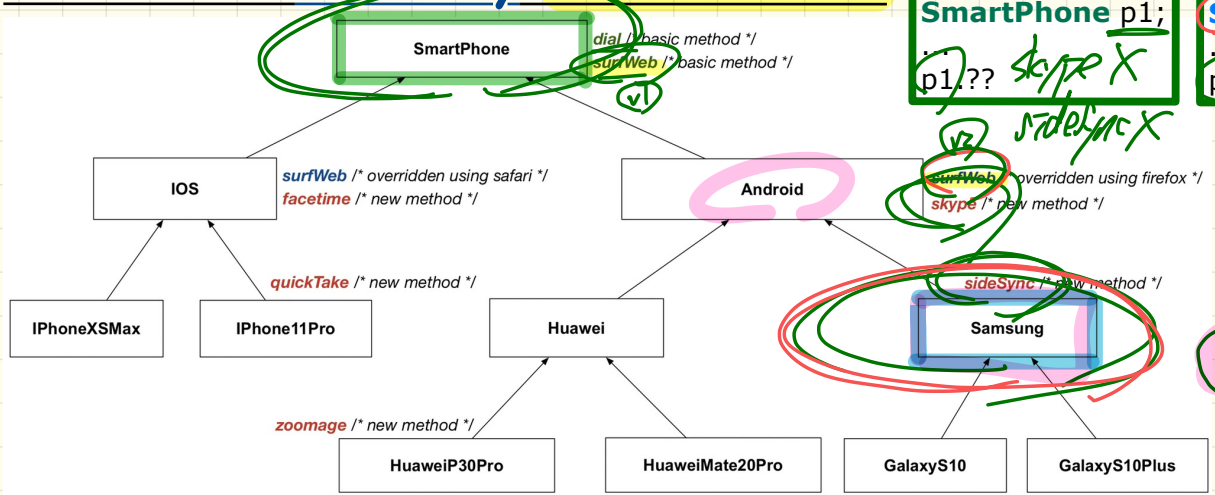
## Inheritance Hierarchy: Students




**Declare:**
**Student** jim;
…
jim.??

**Declare:**
**NRS** alan;
…
alan.??

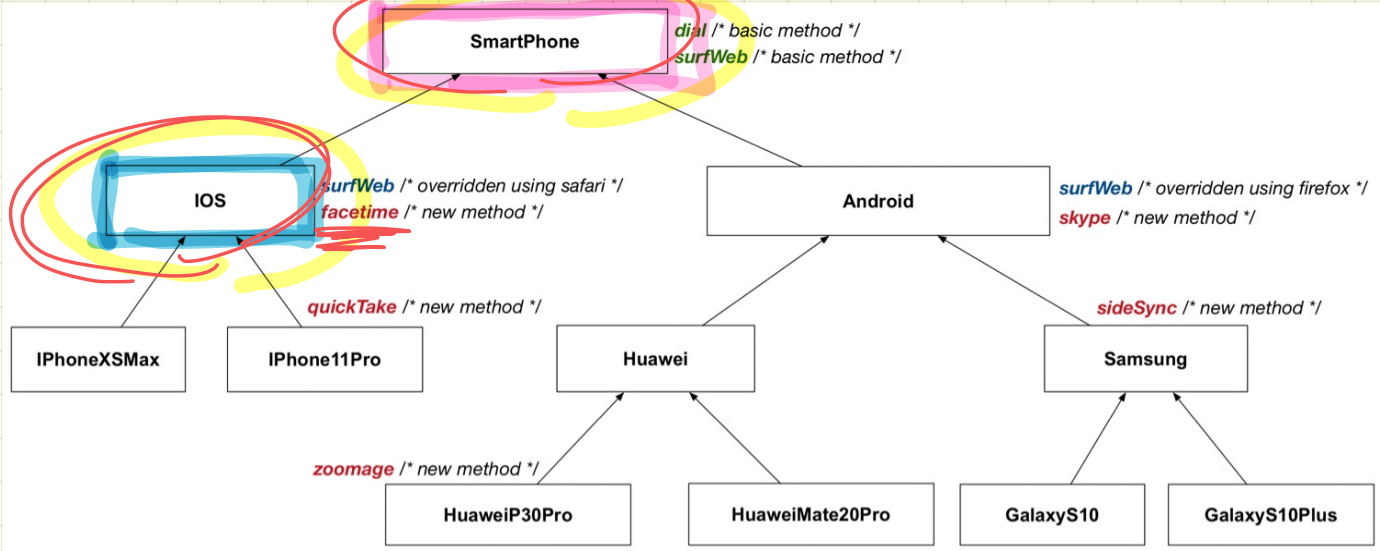
## Inheritance Hierarchy: Smart Phones




**Declare:**
**SmartPhone** p1;
…
p1.??

**Declare:**
**Samsung** p2;
…
p2.??

# Rules of Substitutions




**Declarations:**

**IOS** sp1;

**SmartPhone** sp2;

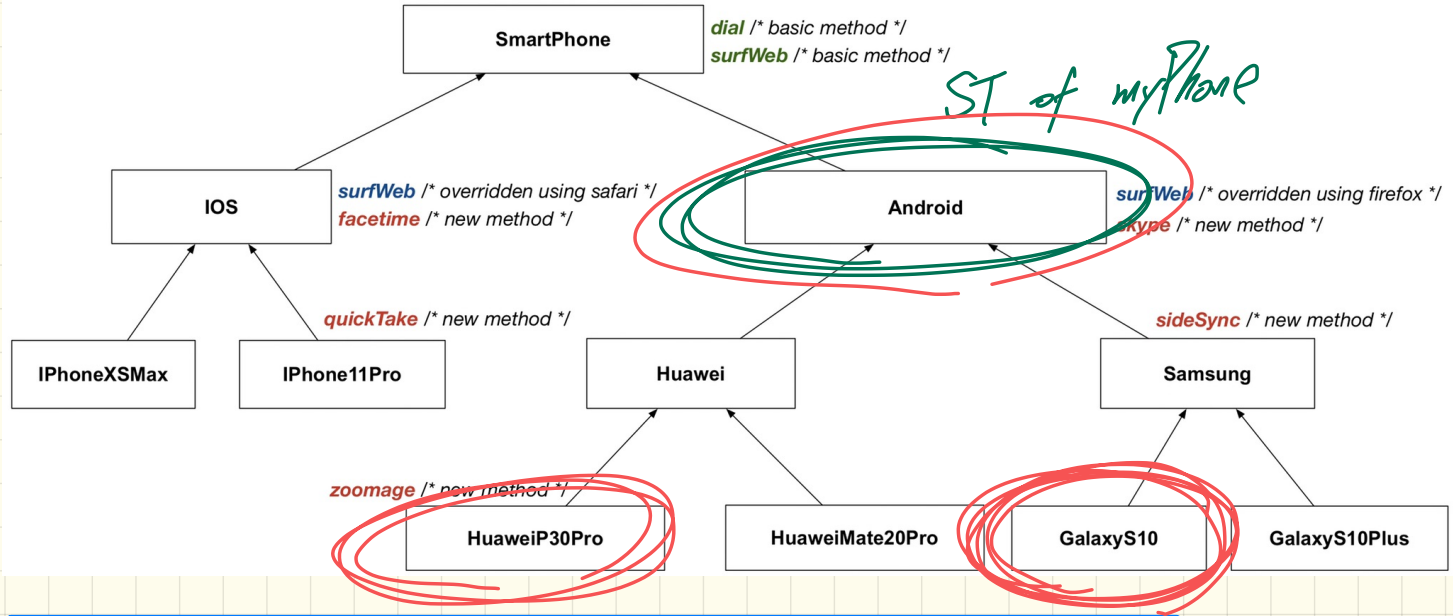
**Substitutions:**

sp1 = sp2;

Can ST of sp2 fulfill what's expected on ST of sp1?

NO ;( facetime not support on sp2.

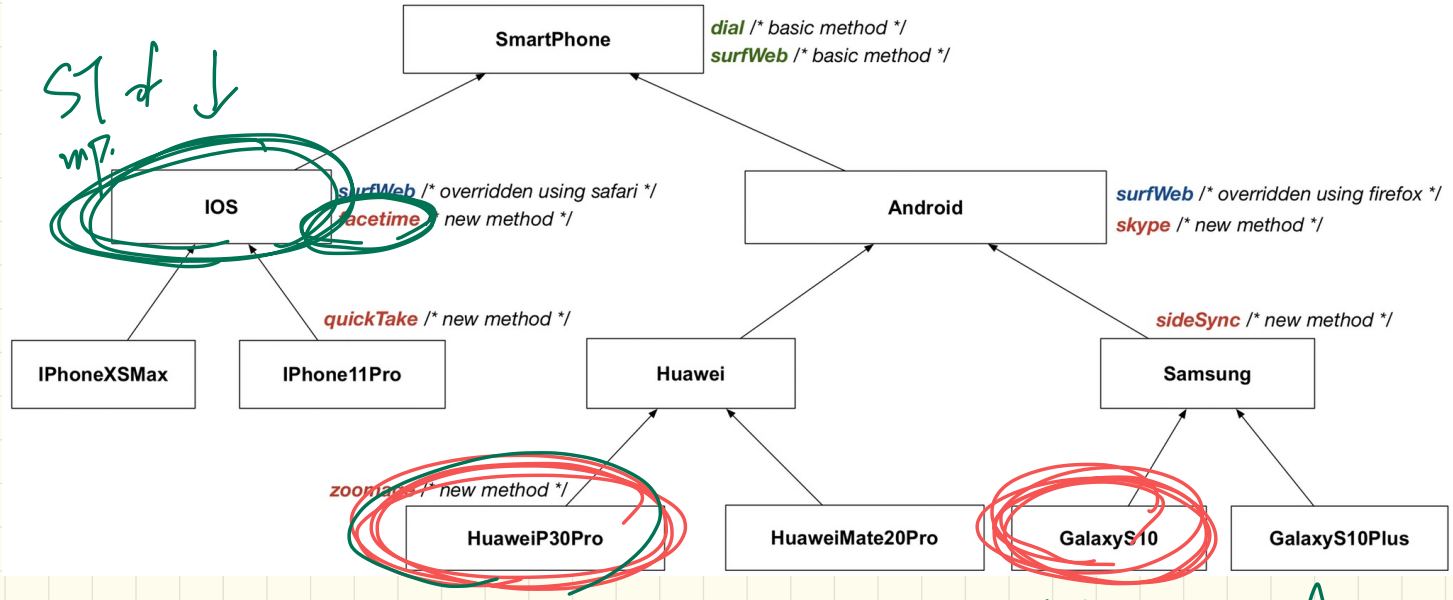
# Change of Dynamic Type: Exercise (1)




## Exercise 1:

```
Android myPhone = new HuaweiP30Pro(...);
myPhone = new GalaxyS10(...);
```

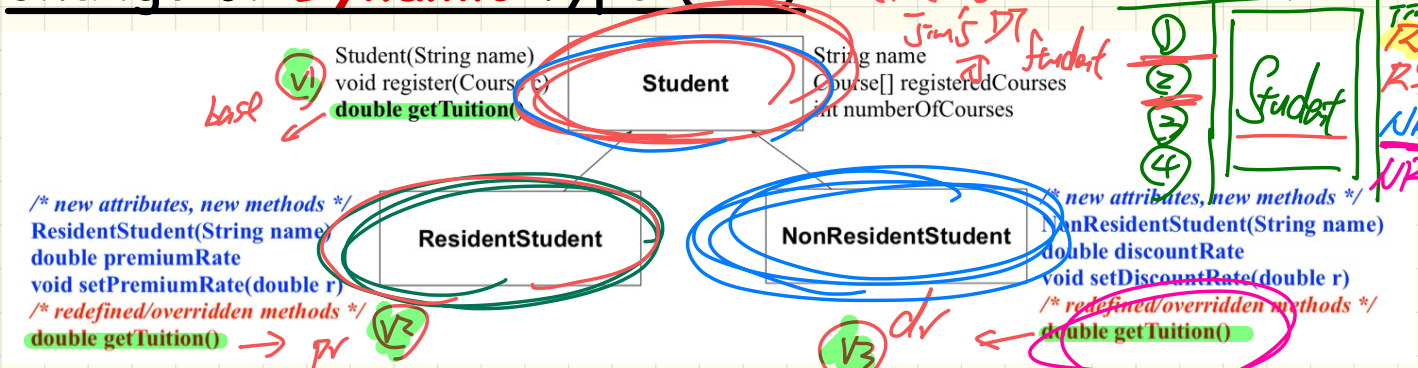
# Change of Dynamic Type: Exercise (2)




## Exercise 2:

```
IOS myPhone = new HuaweiP30Pro(...);
myPhone = new GalaxyS10(...);
```

# Change of Dynamic Type (2.1)

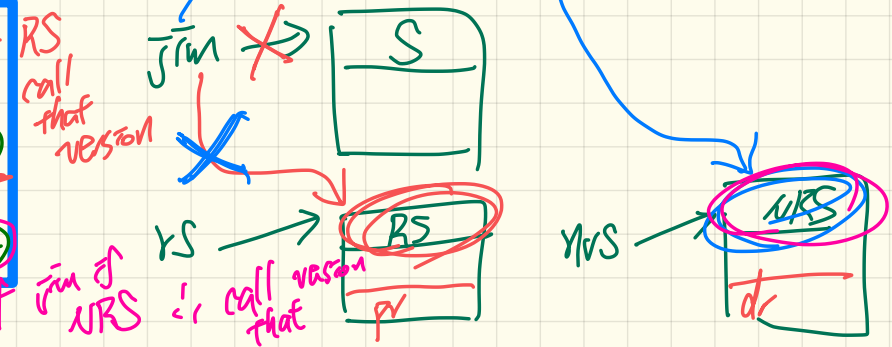



Given:

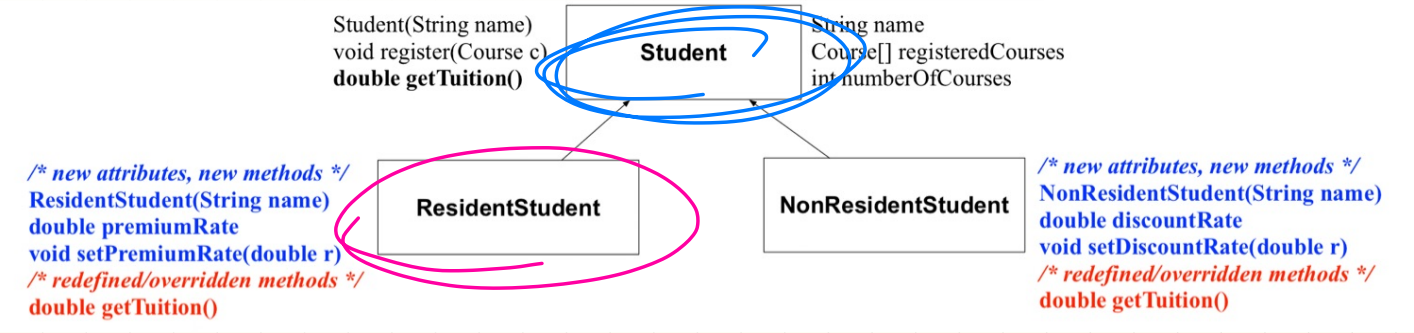
```
Student jim = new Student(...);
ResidentStudent rs = new ResidentStudent(...);
NonResidentStudent nrs = new NonResidentStudent(...);
```

Example 1:

```
jim = rs;
println(jim.getTuition());
jim = nrs;
println(jim.getTuition());
```



# Change of Dynamic Type (2.2)



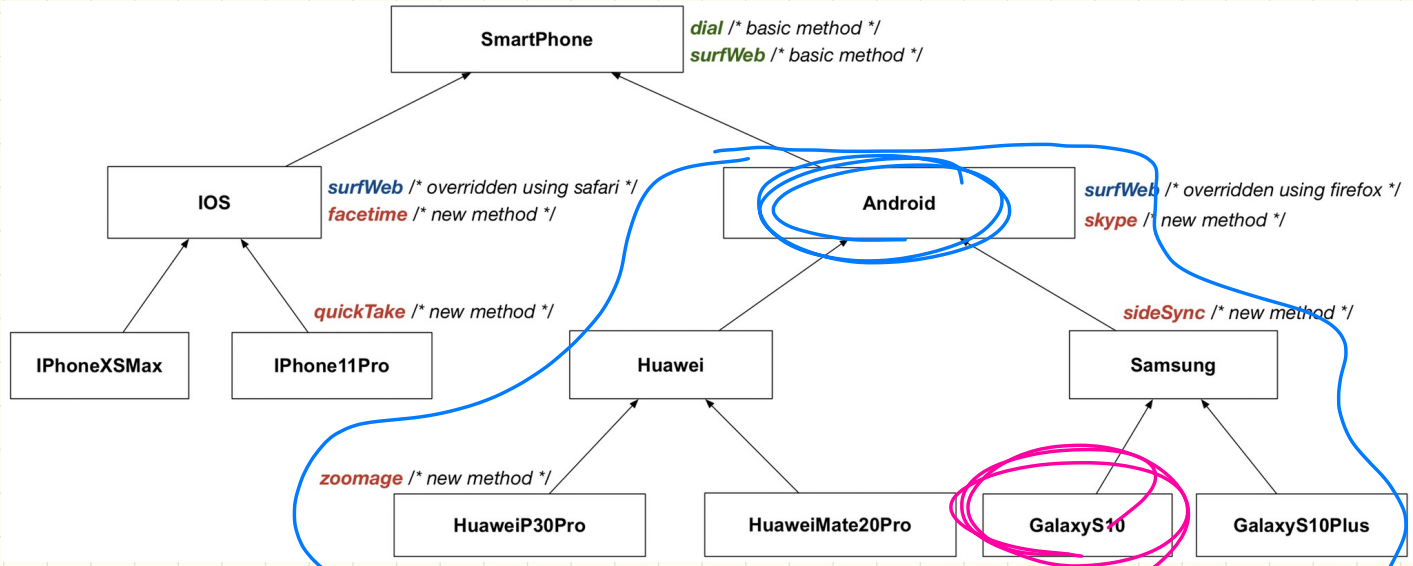

**Given:**

```
Student jim = new Student(...);
ResidentStudent rs = new ResidentStudent(...);
NonResidentStudent nrs = new NonResidentStudent(...);
```

**Example 2:**

```
rs = jim;
println(rs.getTuition());
nrs = jim;
println(nrs.getTuition());
```

cannot be executed if the previous line does not compile.



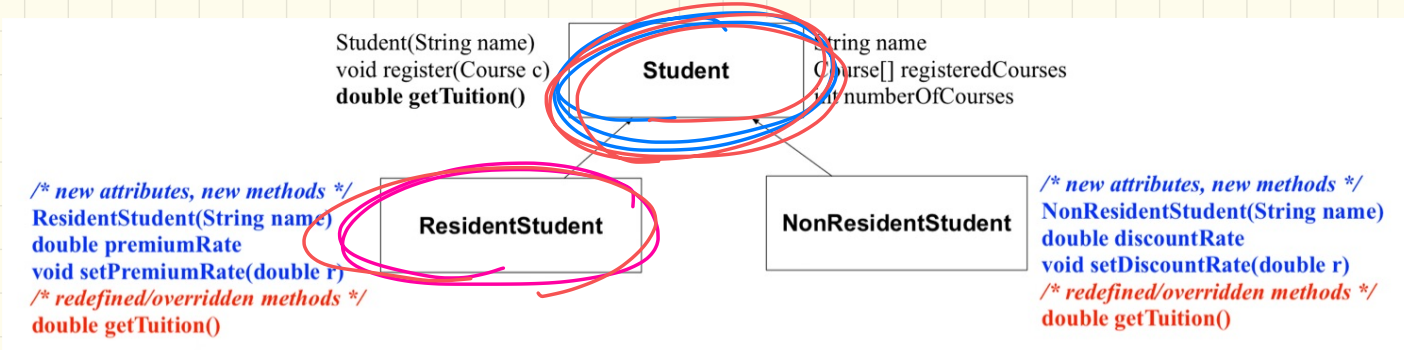

Polymorphism

Android myPhone = ?;

new GS10();

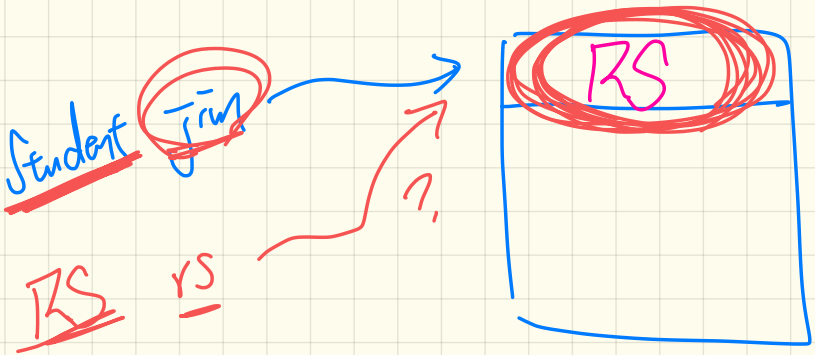
all valid types of objects for substituting Android.

# Type Cast: Motivation

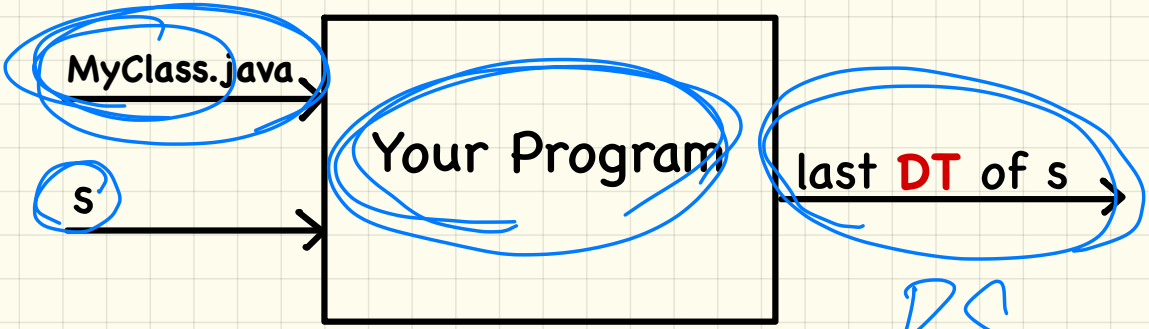



```
1  Student jim = new ResidentStudent("J. Davis");
2  ResidentStudent rs = jim;
3  rs.setPremiumRate(1.5);
```

dynamically points to RS



# An A+ Challenge: Inferring the DT of a Variable




```
class MyClass {
  main (...)
    Student s = ...;
    ...
    s = new ResidentStudent(..);
  }
}
```

RS

undecidable

B

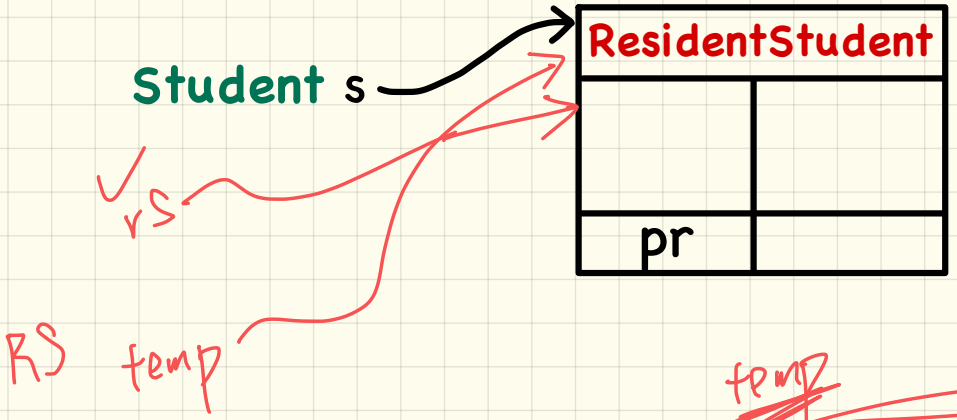
while ( ) {

}

i =

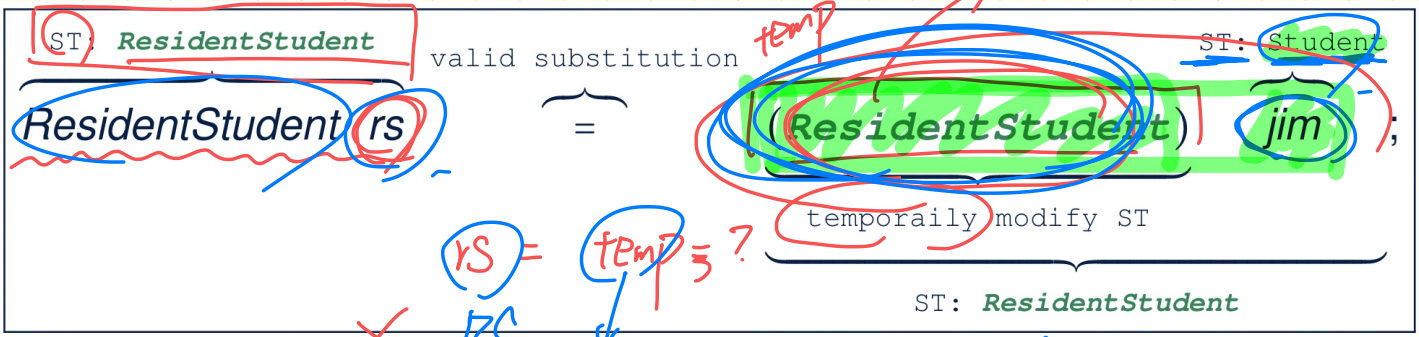
# Anatomy of a Type Cast (1)

```
Student s = new ResidentStudent("Jim");
```




```
ResidentStudent rs = (ResidentStudent) s;
```

# Anatomy of a Type Cast (2)




rs = temp = ?

rs = jim; ✗

RS

ST: RS

Student jim

ST

DT

RS

PT

RS rs

RS temp

Putting a cast.

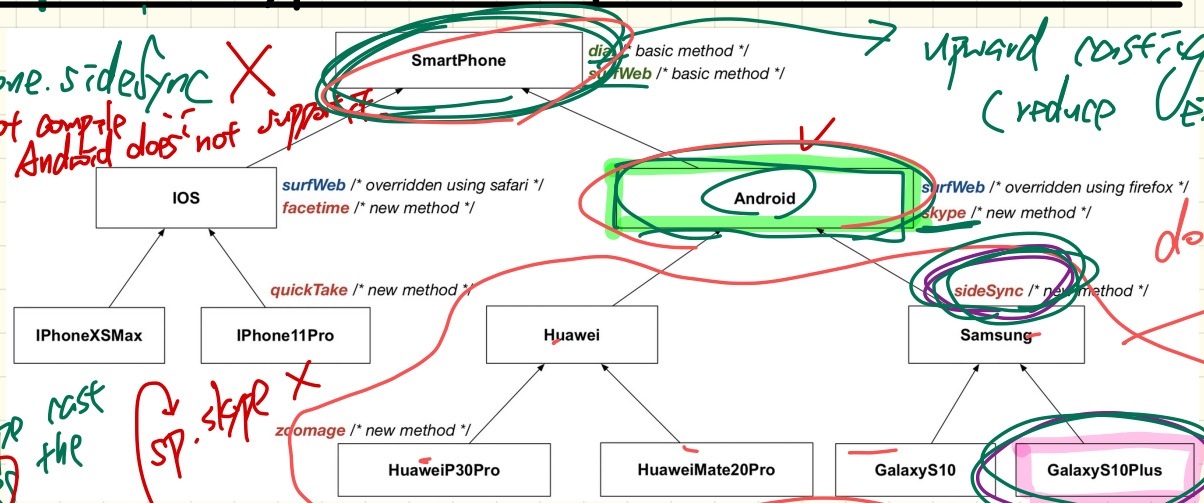
1. No new object created
2. ST of jim not modified
3. a temp. alias of ST RS is created

# Type Cast

↳ change the expectation

1. does the cast compile?
2. if the cast compiles, does it cause an Exception (ClassCastExc.)

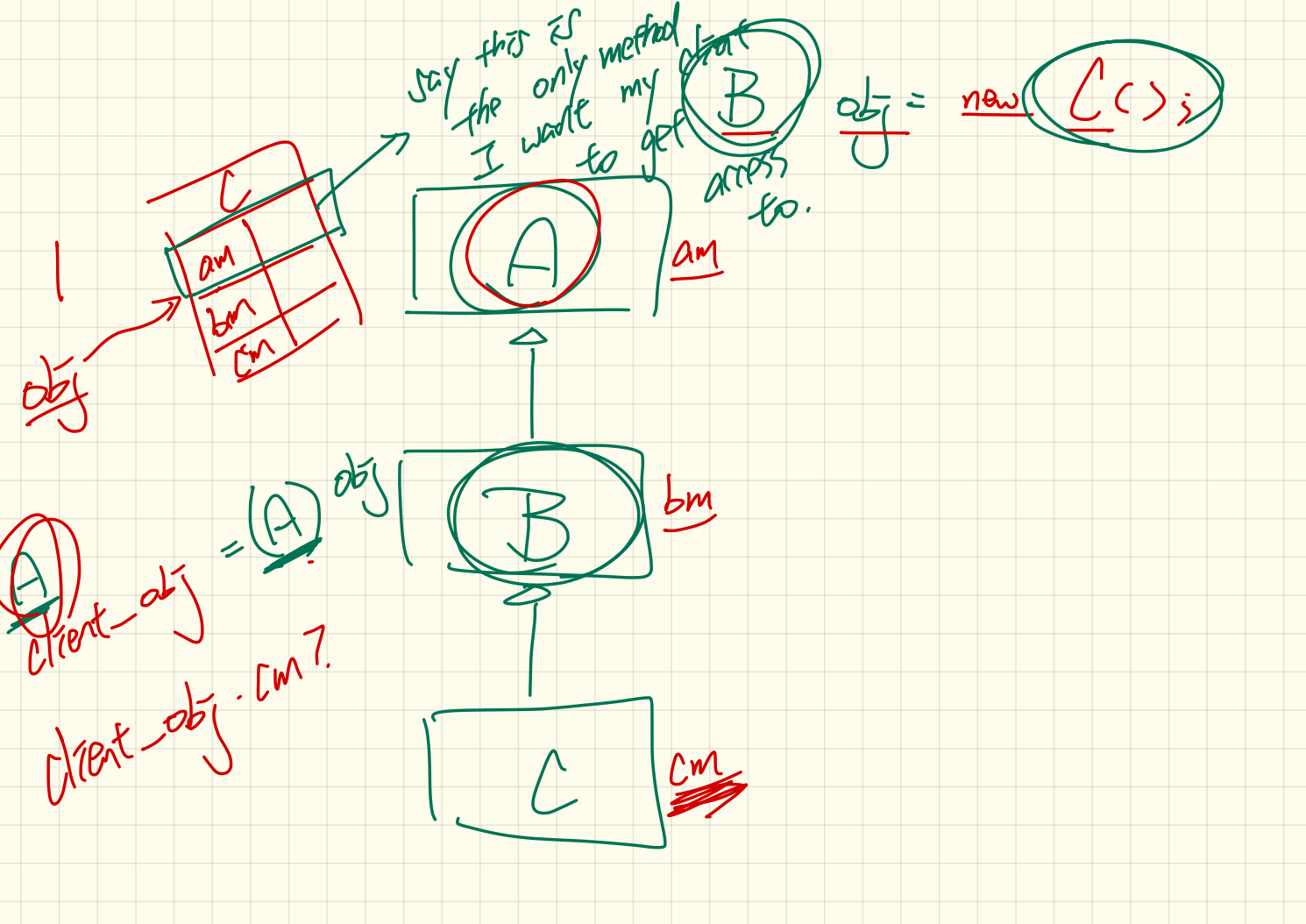
# Compilable Type Casts: Upwards vs. Downwards




## Expectations

| | sp | myPhone | ga |
|---|---|---|---|
| dial | ✓ | ✓ | ✓ |
| surfWeb | ✓ | ✓ | ✓ |
| skype | ✗ | ✓ | ✓ |
| sideSync | ✗ | ✗ | ✓ |
| facetime | ✗ | ✗ | ✗ |
| quickTake | ✗ | ✗ | ✗ |
| zoomage | ✗ | ✗ | ✗ |

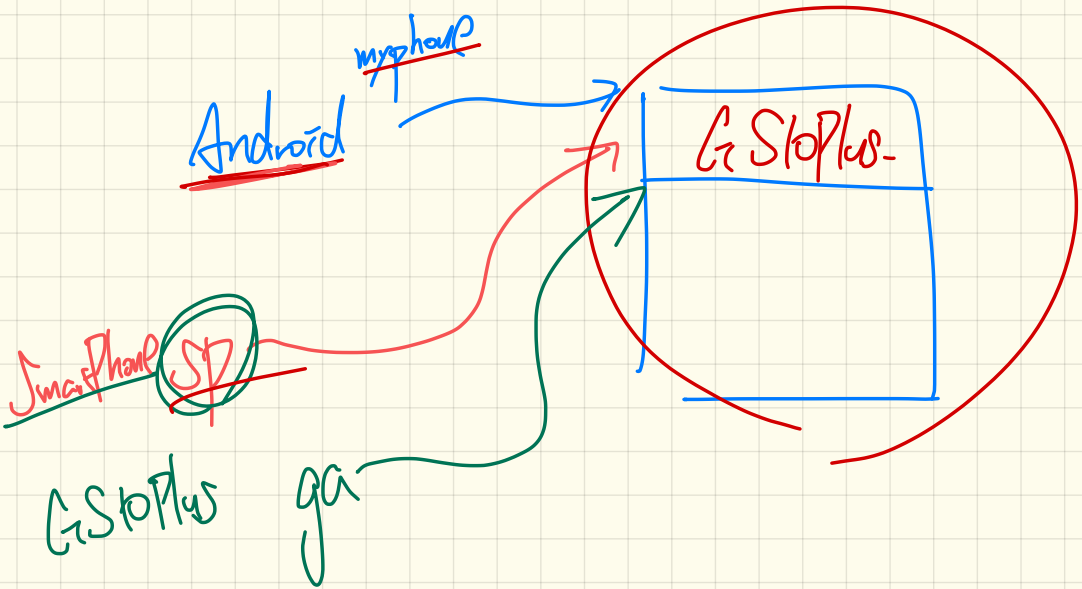
```
Android myPhone = new GalaxyS10Plus();
SmartPhone sp = (SmartPhone) myPhone;
GalaxyS10Plus ga = (GalaxyS10Plus) myPhone;
```


Say this is the only method I want my client to get access to.
B
obj = new C();
C
am
bm
cm
obj
A
am
B
bm
C
cm
A client_obj = (A) obj
client_obj.cm?

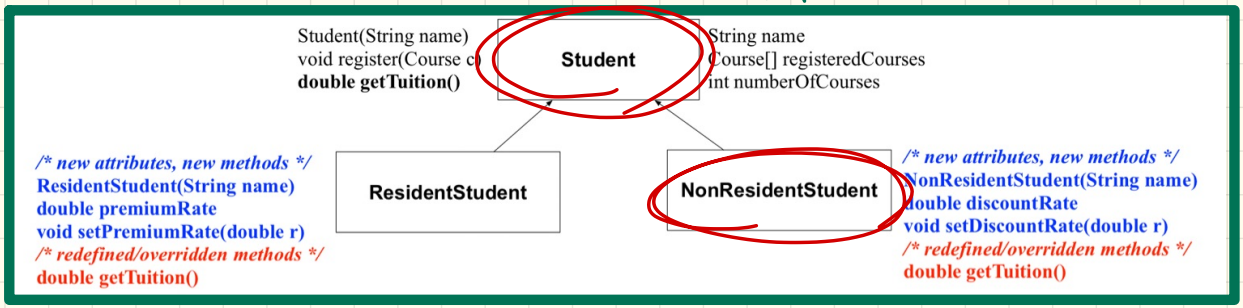
```
Android myPhone = new GalaxyS10Plus();

SmartPhone sp = (SmartPhone) myPhone;

GalaxyS10Plus ga = (GalaxyS10Plus) myPhone;
```



# Compilable Type Cast May Fail at Runtime (1)




```
1 Student jim = new NonResidentStudent("...");
2 ResidentStudent rs = ((ResidentStudent) jim);
3 rs.setPremiumRate(1.5);
```

cast version of jim of ST: RS

ST: Student

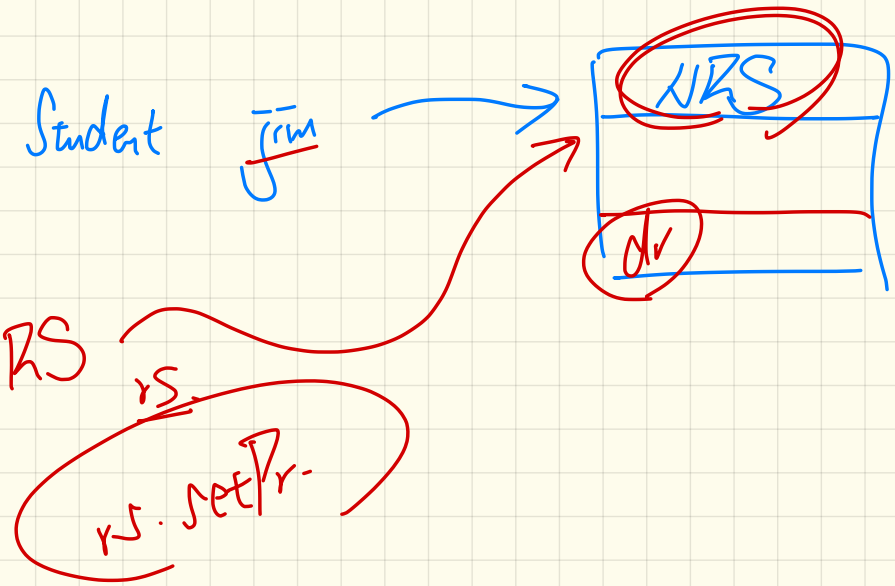
ResidentStudent rs = jim; X

rs = cast version ✓

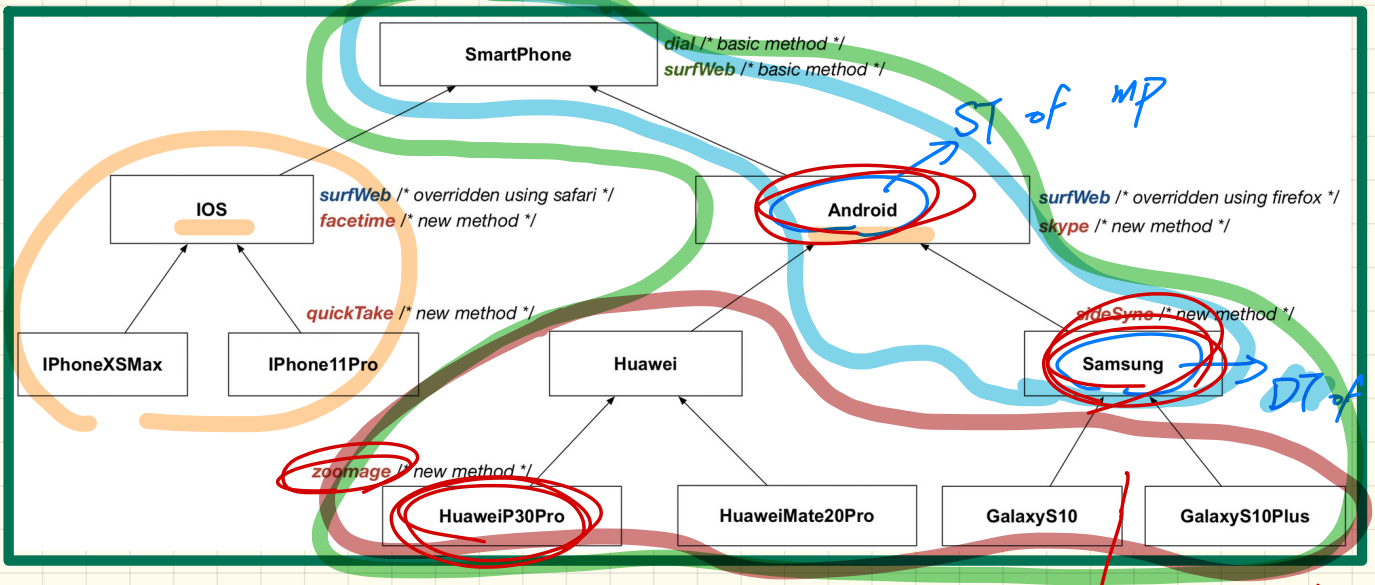
valid downward casting!

```
1  Student jim = new NonResidentStudent("J. Davis");
2  ResidentStudent rs = (ResidentStudent) jim;
3  rs.setPremiumRate(1.5);
```

class cast exception.



# Compilable Cast vs. Exception-Free Cast




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

downward casting but DT of SP cannot support it.

Compilable Casts

Exception-Free Casts

Non-Compilable Casts

ClassCastException