Lecture 19 - Nov. 14

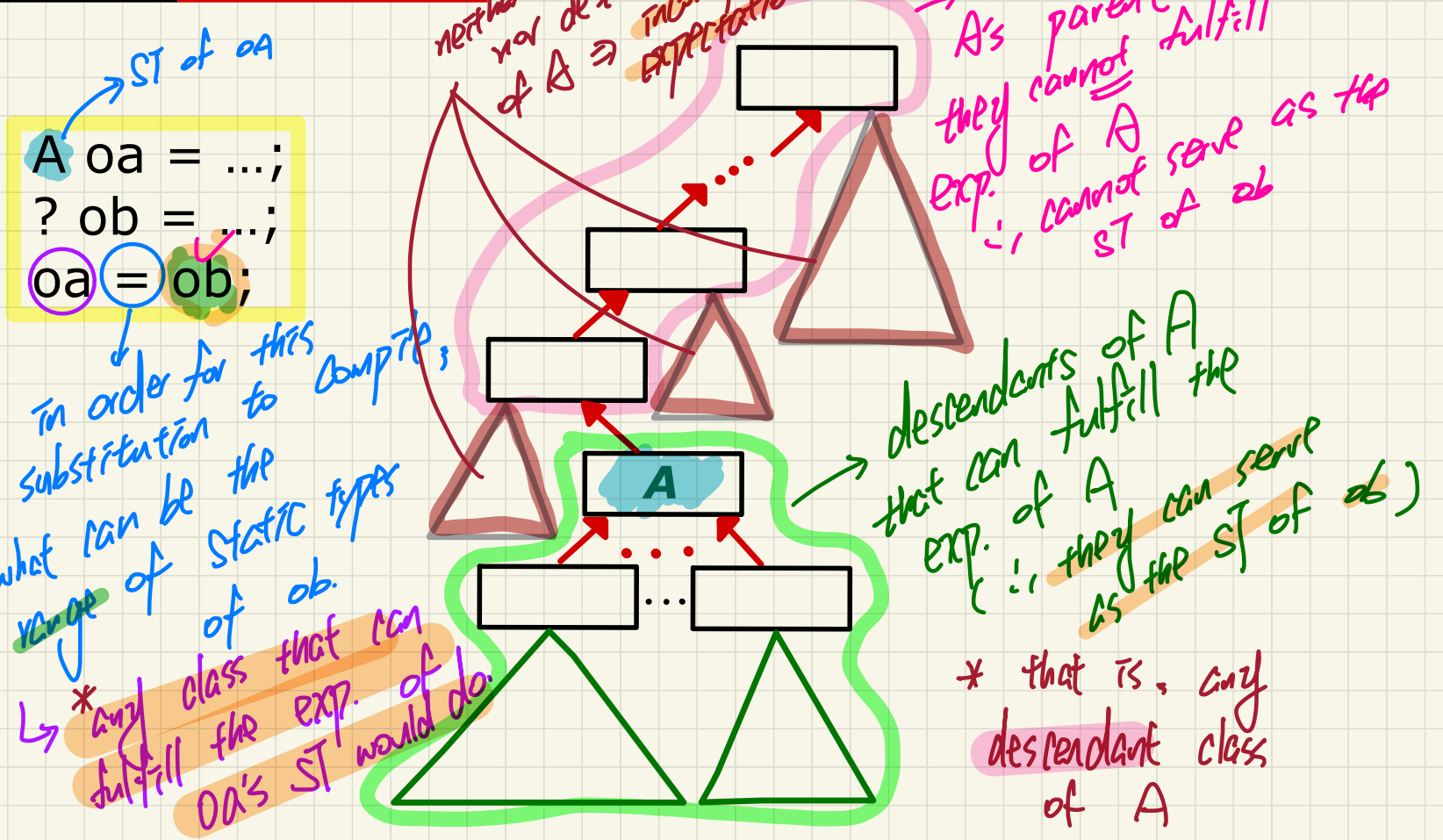
Inheritance

Polymorphism vs. Dynamic Binding
Type Casts: Named vs. Anonymous
Casts: Compilable vs. ClassCastException

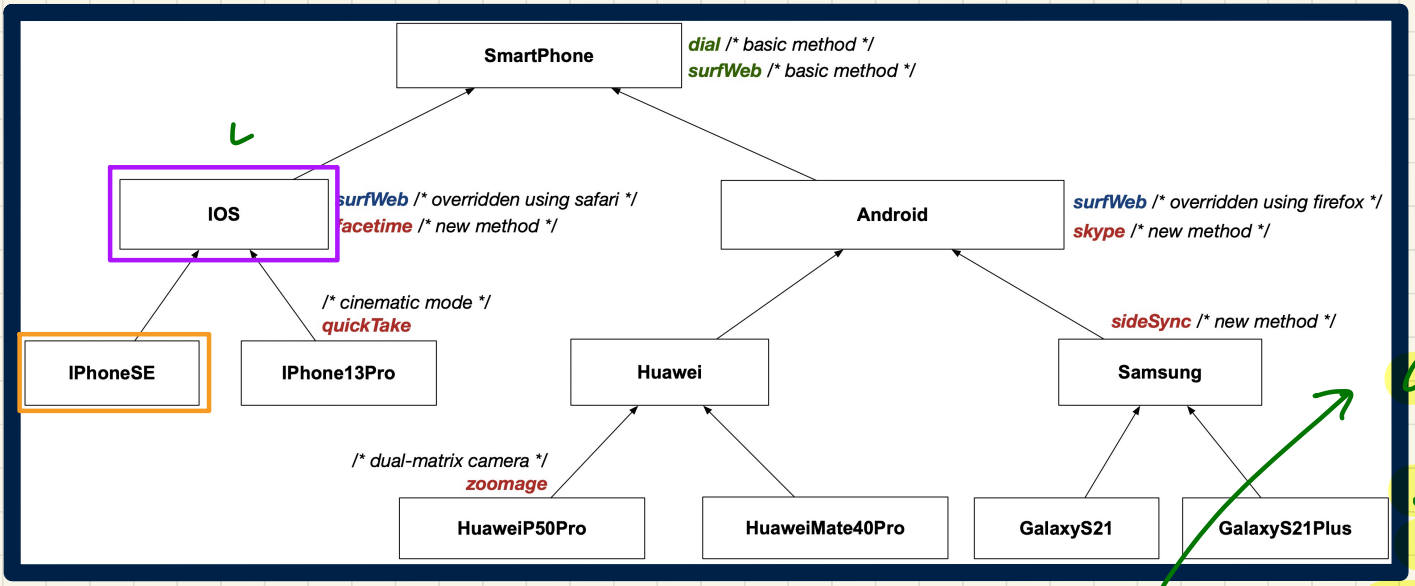
Announcements/Reminders

- **WrittenTest2** results to be released by Monday
- **Lab4** due tomorrow at noon
- **Lab5** to be released tomorrow
- **ProgTest3** next Wednesday, November 20
  + **Lab4** grading tests
  + **Lab4** solution video
- **Bonus** Opportunity coming: Formal Course Evaluation

Rules of Substitutions



Rules of Substitutions (1)



Declarations:

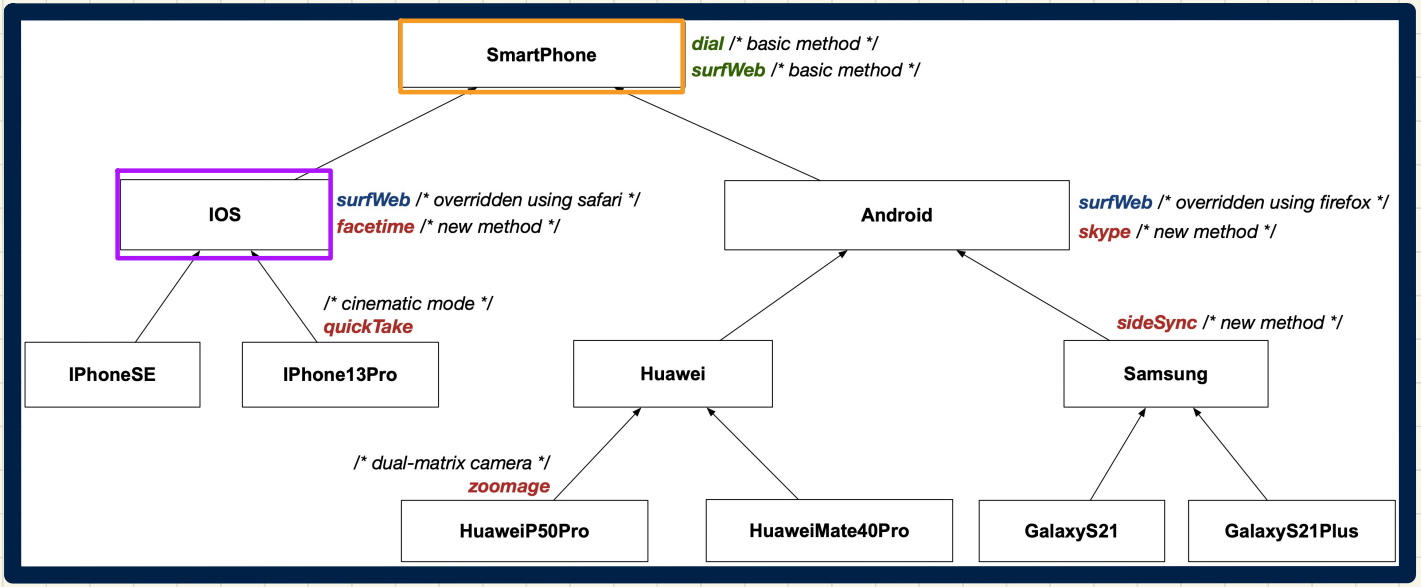
```
IOS sp1;
IPhoneSE sp2;
IPhone13Pro sp3;
```

Substitutions:

```
sp1 = sp2;
sp1 = sp3;
```

Compiles: sp2's ST IPSE is a descendant of sp1's ST IOS

Rules of Substitutions (2)



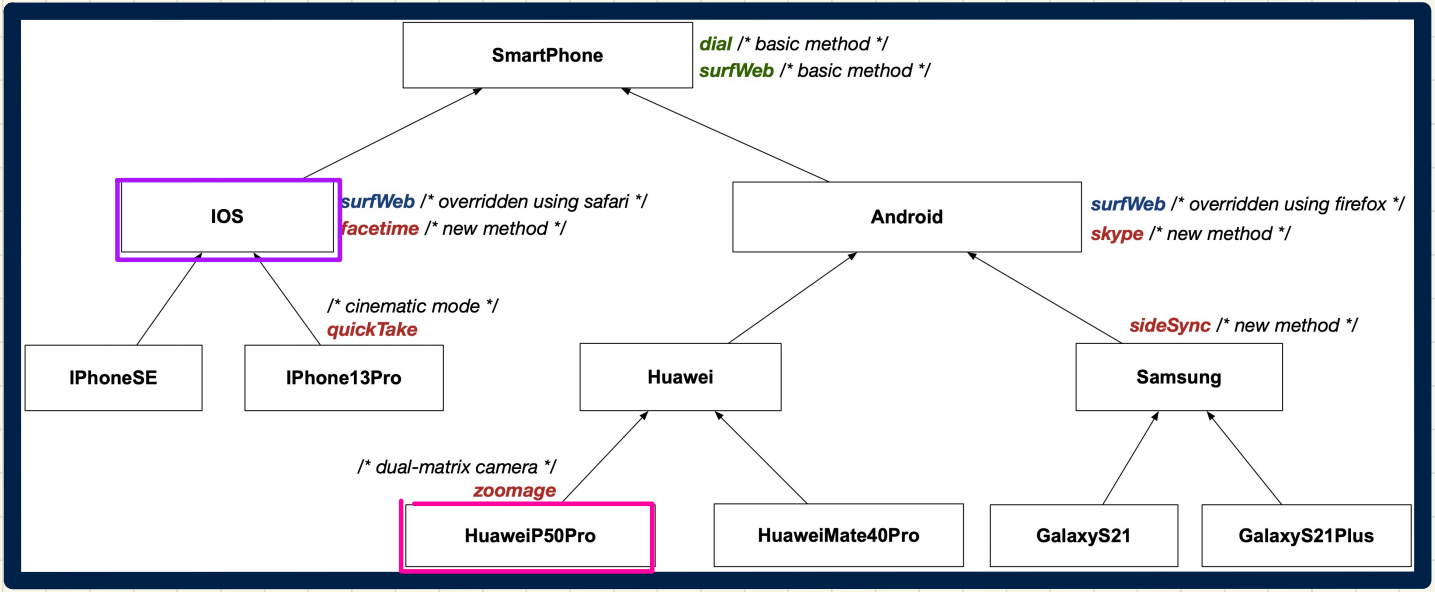
Declarations:

```
IOS sp1;
SmartPhone sp2;
```

Substitutions:

```
sp1 ✗= sp2;
```

Rules of Substitutions (3)



Declarations:

IOS sp1;

HuaweiP50Pro sp2;

Substitutions:

sp1 $\overset{X}{=}$ sp2;

ST of sp1 (IOS)

ST of sp2 (HuaweiP50Pro) is neither an ancestor nor descendant of

Visualization: Static Type vs. Dynamic Type

Declaration:

Student s;

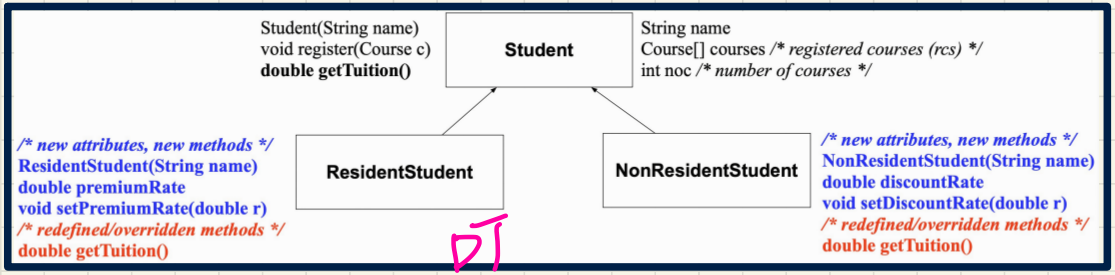
ST

Substitution:

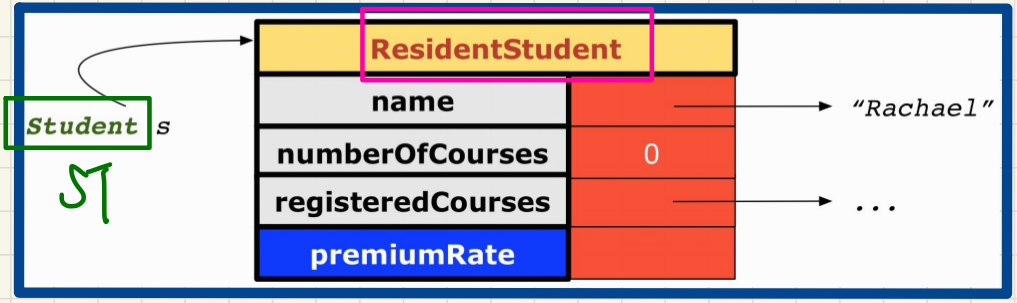
s = **new ResidentStudent**("Rachael");

DT

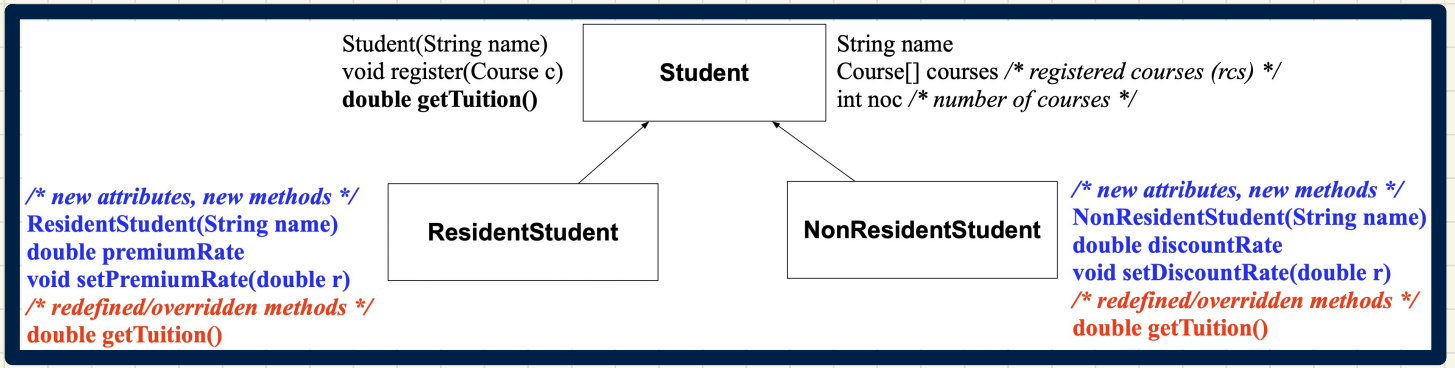
> **Static Type**: Expectation
>
> **Dynamic Type**: Accumulation of Code



DT



Change of Dynamic Type (1.1)



Example 1:

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

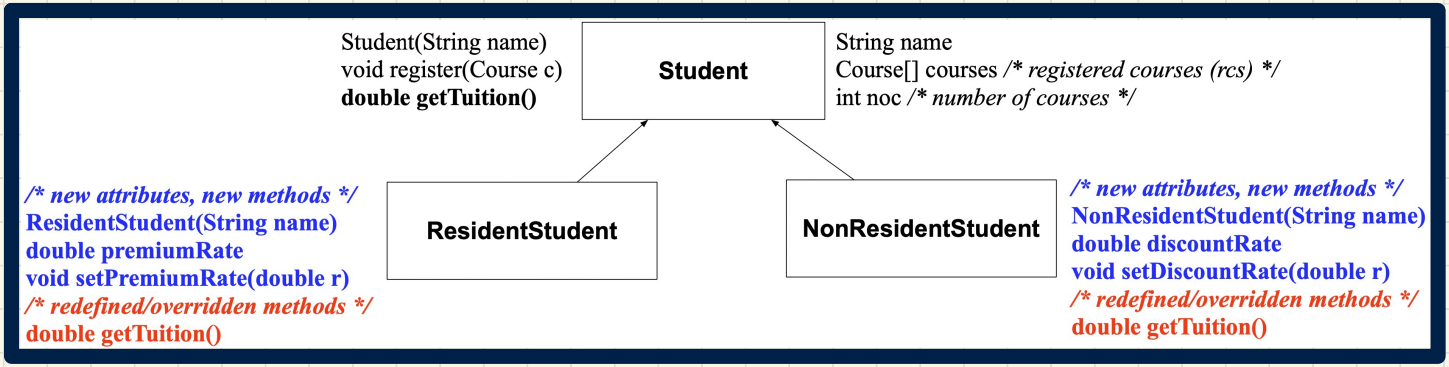
compiles if RS can fulfill the exp

of jim's ST (Student)
(RS is a descendant of Student)

DT of jim: RS

DT of jim: NRS

Change of Dynamic Type (1.2)

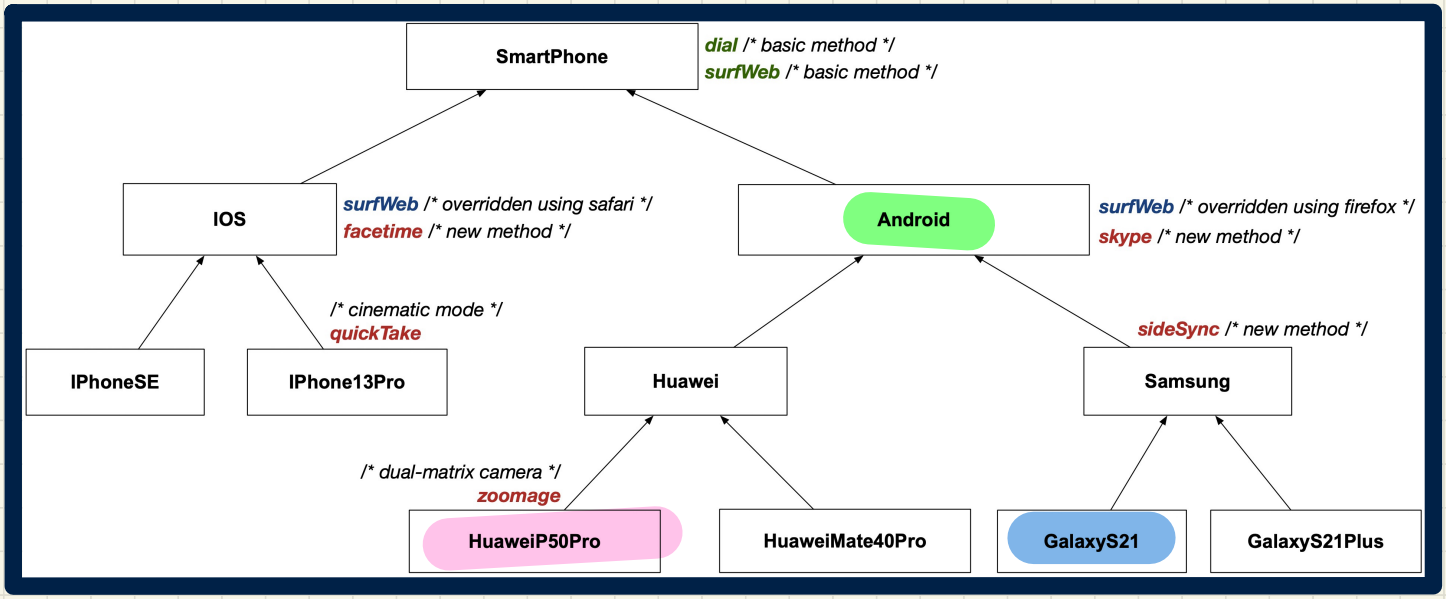


Example 2:

ResidentStudent jeremy = **new Student**(...);

∵ Student is not a descendant of ST of jeremy (RS).

Change of Dynamic Type: Exercise (1)



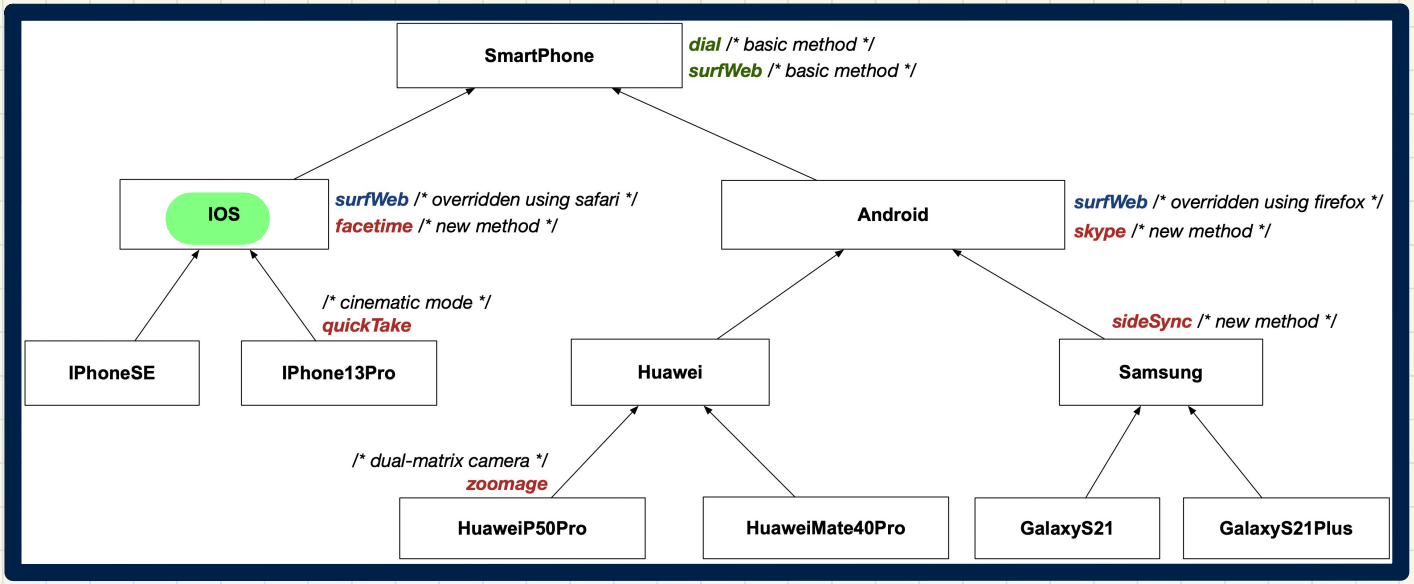
Exercise 1:

```
Android myPhone = new HuaweiP50Pro(...);
myPhone = new GalaxyS21(...);
```

DT: HP50P

DT: GS21

Change of Dynamic Type: Exercise (2)

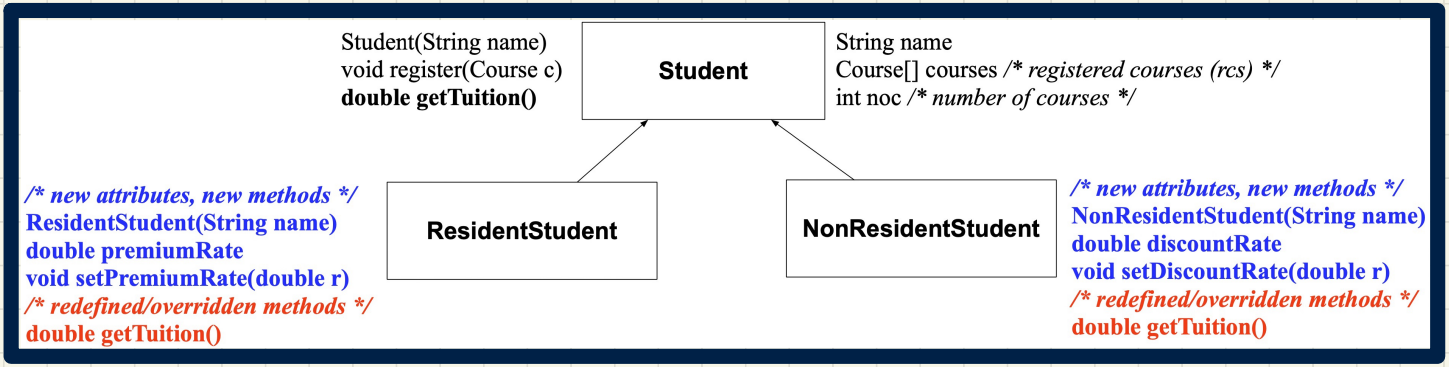


Exercise 2:

IOS myPhone = new HuaweiP50Pro(...); ✗

myPhone = new GalaxyS21(...); ✓

Change of Dynamic Type (2.1)



Given:

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

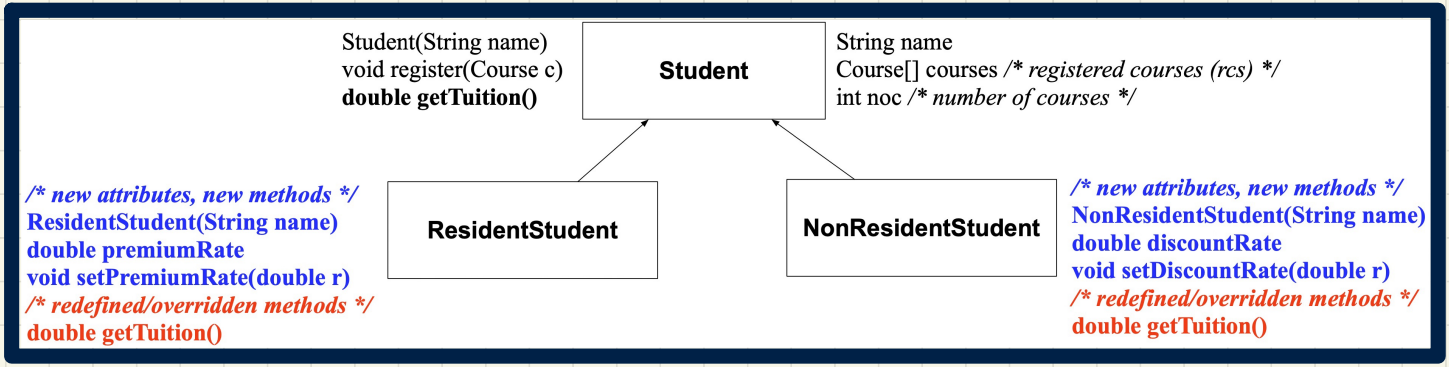
Example 1:

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

DT: RS → version from RS (rv)

DT: NRS → version from NRS (nrv).

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());
```

Polymorphism and Dynamic Binding

Polymorphism:

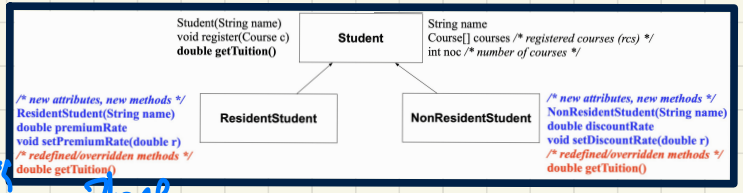
An object's **static type** may allow **multiple** possible **dynamic types**.

⇒ Each **dynamic type** has its **version** of method.

Dynamic Binding:

An object's **dynamic type** determines the **version** of method being invoked.

```
Student jim = new ResidentStudent(...);
jim.getTuition();
jim = new NonResidentStudent(...);
jim.getTuition();
```



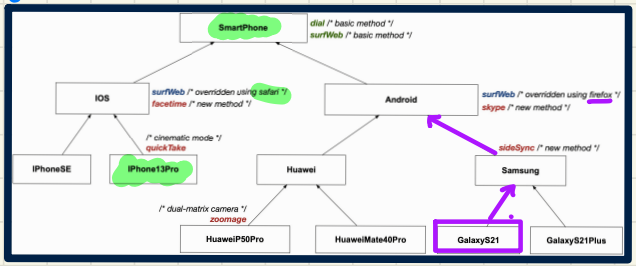
polymorphism: sp1 can have multiple possible DTs: descendants of SmartPhone

```
SmartPhone sp1 = new IPhone13Pro(...);
SmartPhone sp2 = new GalaxyS21(...);
sp1.surfWeb();
sp1 = sp2;
sp1.surfWeb();
```

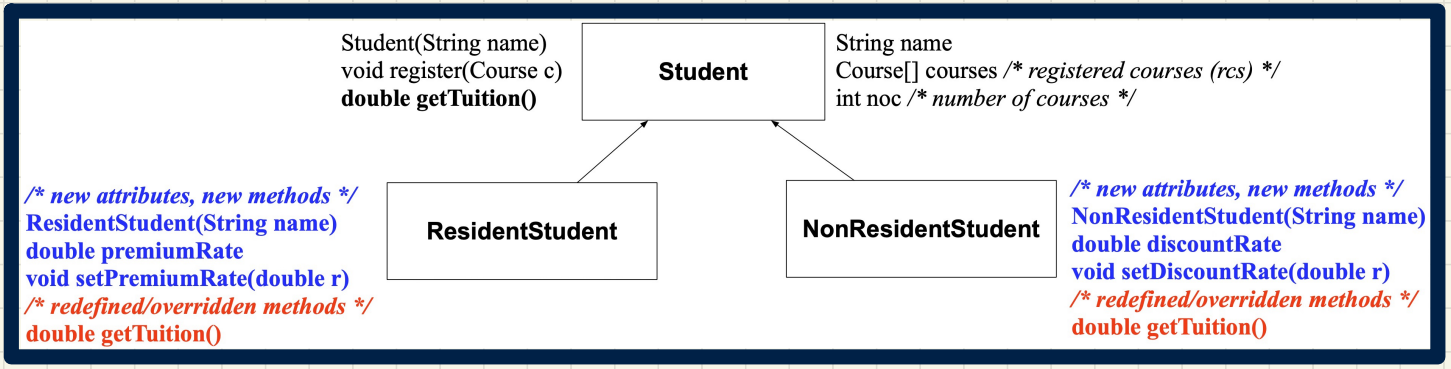
DT: IP13Pro

changes sp1's DT from IP13Pro to GS21

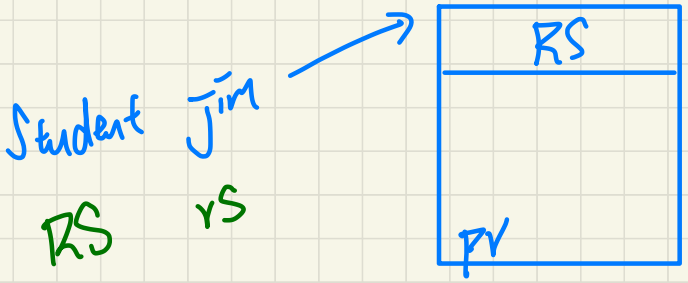
DT: GS21



Type Cast: Motivation

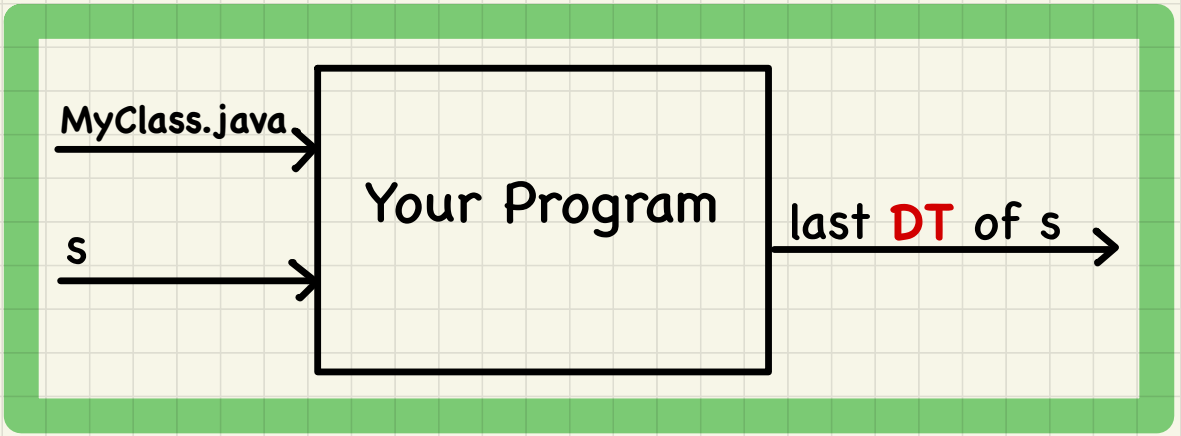



```
1  Student jim = new ResidentStudent("J. Davis");
2  ResidentStudent rs = jim;
3  rs.setPremiumRate(1.5);
```



An A+ Challenge: Inferring the DT of a Variable

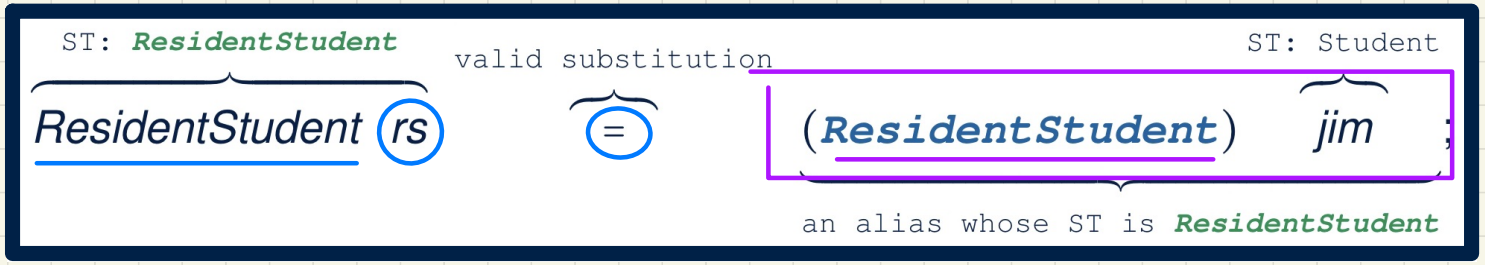
undecidable

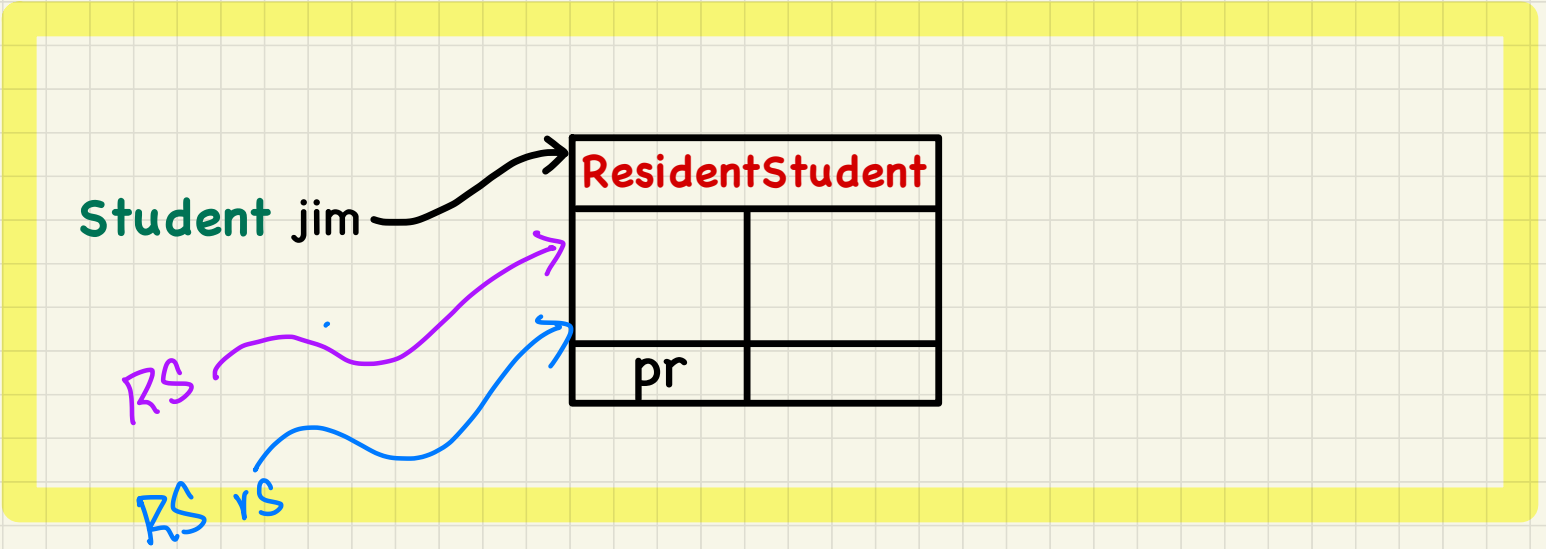



```
class MyClass {
   main (...)
      Student s = ...;
      ...
      s = new ResidentStudent(...);
   }
}
```

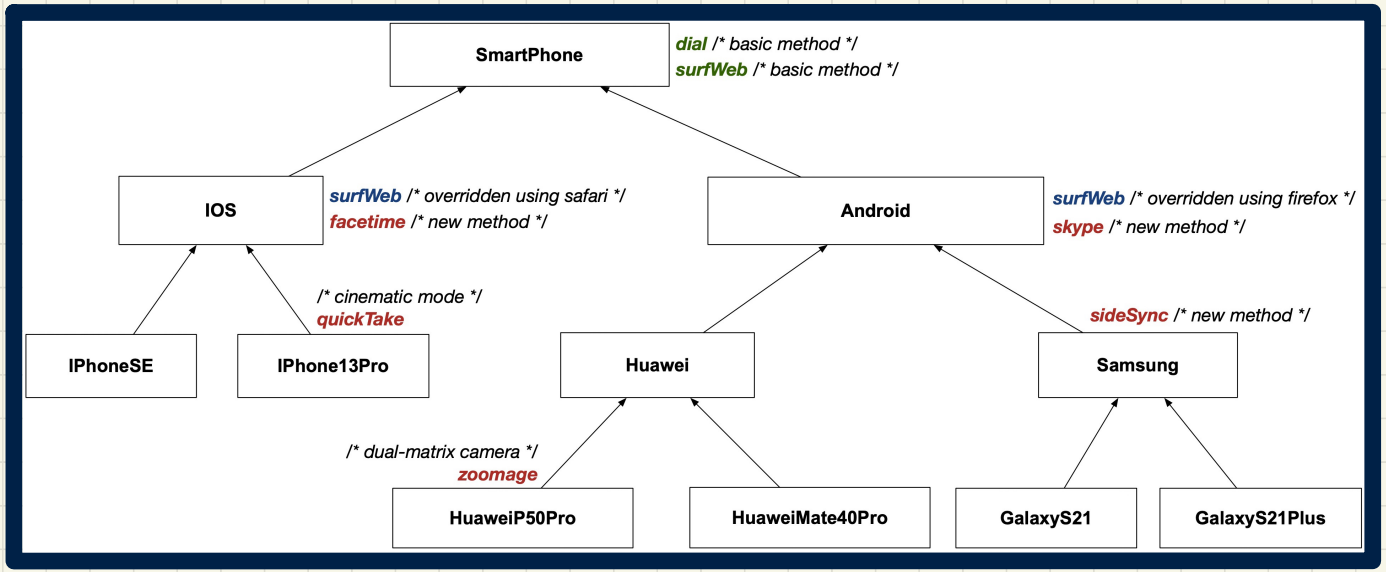
Anatomy of a Type Cast

Student jim = **new ResidentStudent**("Jim");





Type Cast: Named vs. Anonymous



Named Cast: Use intermediate variable to store the cast result.

```
SmartPhone aPhone = new IPhone13Pro();
IOS forHeeyeon = (IPhone13Pro) aPhone;
forHeeyeon.facetime();
```

Anonymous Cast: Use the cast result directly.

```
SmartPhone aPhone = new IPhone13Pro();
((IPhone13Pro) aPhone).facetime();
```

alias of ST: IP13Pro

Exercise

```
SmartPhone aPhone = new IPhone13Pro();
(IPhone13Pro) aPhone.facetime();
```

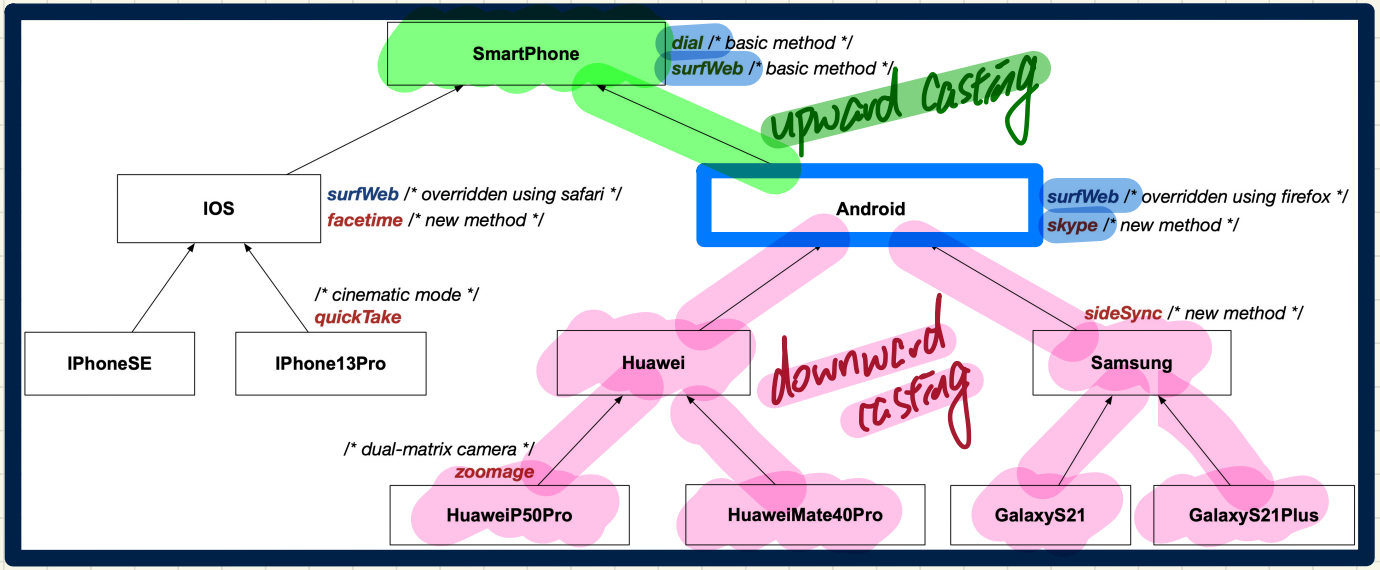
not compilable
∵ facetime not part of the exp of aPhone's ST (SP)

Compilable Casts: Upwards vs. Downwards

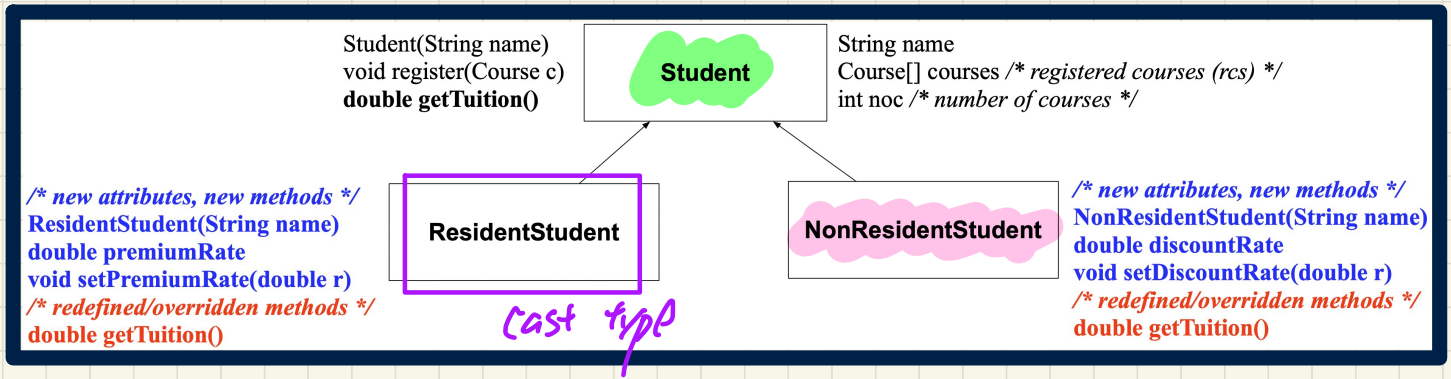
```
Android myPhone = new GalaxyS21Plus();

SmartPhone sp = (SmartPhone) myPhone;

GalaxyS21Plus ga = (GalaxyS21Plus) myPhone;
```

Expectations

	sp	myPhone	ga
dial			
surfWeb			
skype			
sideSync			
facetime			
quickTake			
zoomage			



Compilable Type Cast May Fail at Runtime (1)




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

→ at RT: ClassCastException

2. DT NRS cannot fulfill the exp. of rs (RS) downward cast
↳ ClassCastException ↳ compiles.

Runtime

1. rs is expected to be used as RS