Lecture 19 - Nov 16

Inheritance

Type Casting: Upward vs. Downward
Danger of Casts: ClassCastException

Announcements

- Lab4 released

Recap: Static Types vs. Dynamic Types

```
C1 v1 = new C3(...);
C2 v2 = new C4(...);
v1.m();
v2.m();
v1 = v2;
v1.m();
v2.m();
```

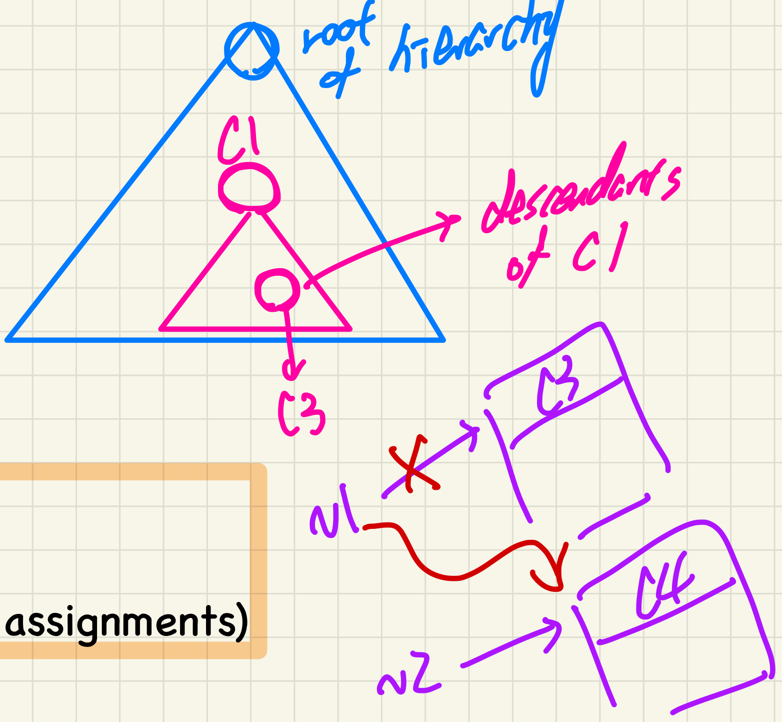
static types

DT of v1 is C3

C3 should be a descendant of C1
DT ST
↳ C3 can fulfil C1's expectation

DT of v1 changed to C4

v2's ST should be a descendant of v1's ST



Exercises on Eclipse:

+ SMS (variable assignments)
+ Smart Phones (hierarchy + variable assignments)

variable assignments   version of methods involved.

Polymorphism and Dynamic Binding

Polymorphism:

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

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

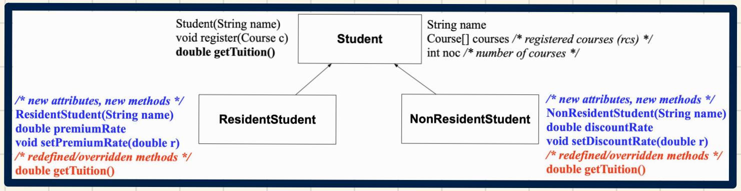
Dynamic Binding:

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

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

DT: RS

DT: NRS


Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] courses /* registered courses (rcs) */
int noc /* number of courses */
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

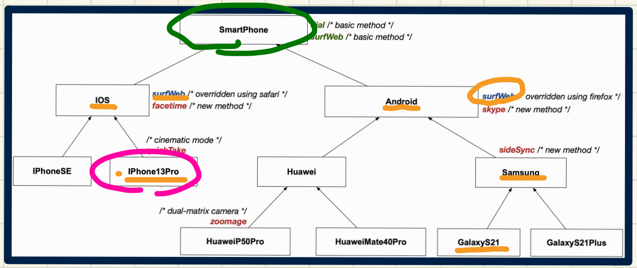
```
SmartPhone sp1 = new IPhone13Pro(...);
SmartPhone sp2 = new GalaxyS21(...);
sp1.surfWeb();
sp1 = sp2;
sp1.surfWeb();
```

→ DT of sp1 is IPhone13Pro.

→ DT: GalaxyS21

→ DT of sp1 is GalaxyS21

DT will become the DT of sp2.



Anatomy of a Type Cast

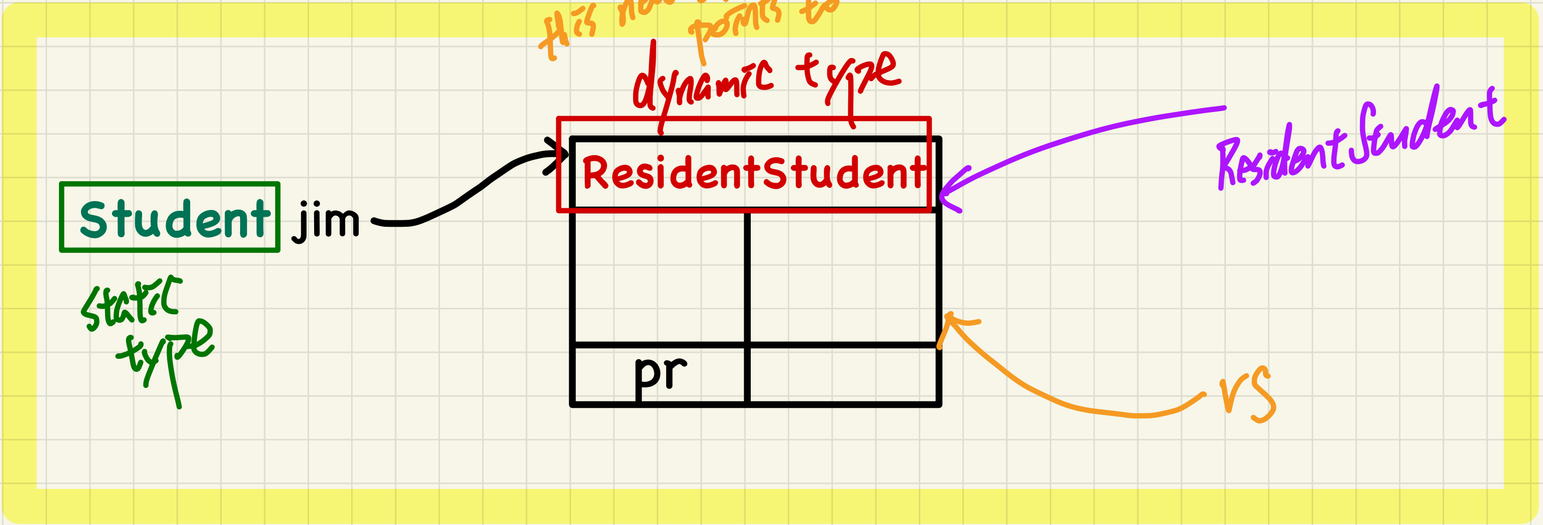
Student jim = **new** **ResidentStudent**("Jim");   RS.

type of the cast expression corresponds to the cast class: RS.

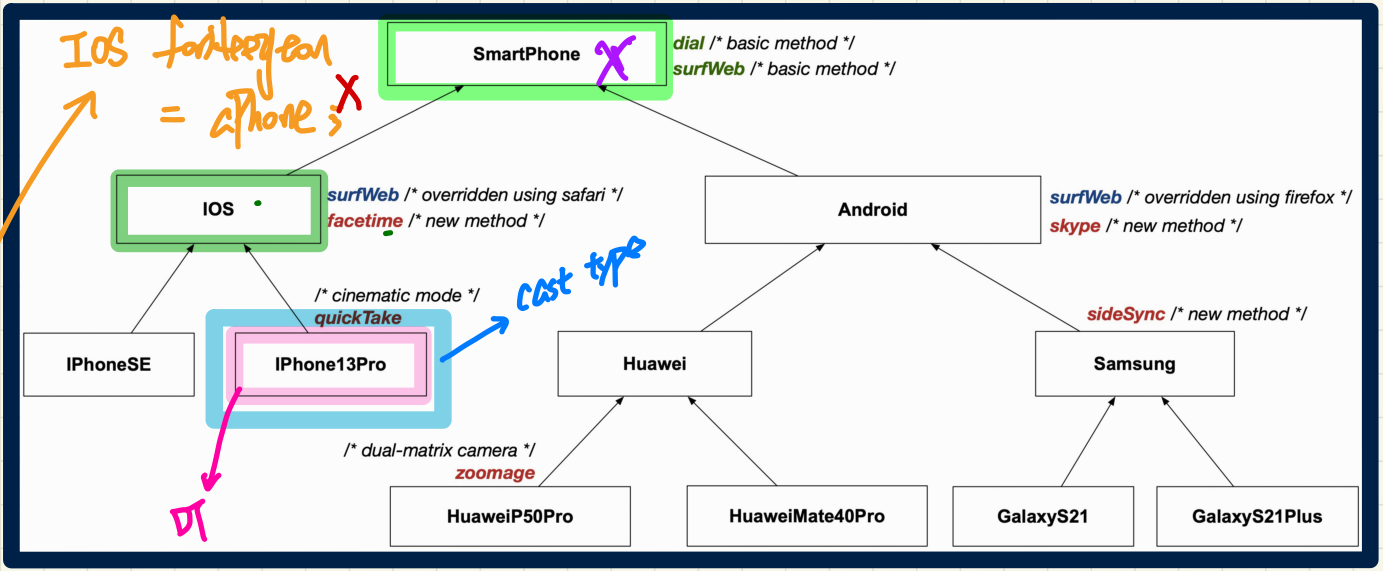
$$\underbrace{\texttt{ResidentStudent rs}}_{\text{ST: ResidentStudent}} \ \underbrace{=}_{\text{valid substitution}} \ \underbrace{(\texttt{ResidentStudent}) \ \underbrace{\texttt{jim}}_{\text{ST: Student}}}_{\text{an alias whose ST is ResidentStudent}};$$

ST of a new alias

rs points to wherever this new alias points to



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();
```

ST aPhone → IPBP ← IPhone13Pro

ST: IOS includes facetime as exp

Anonymous Cast: Use the cast result directly.

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

Exercise

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

type: IP13Pro

① ✓ ((IPhone13Pro) aPhone).facetime(); ✓

② ✗ (IPhone13Pro) aPhone.facetime(); ✗ ST of aPhone only look at STs.

→ restrict expectation. → polymorphism

Compilable Casts: Upwards vs. Downwards

DT irrelevant for deciding if a cast compiles w.r.t. a variable's ST.

```
Android myPhone = new GalaxyS21Plus();
```

result of upward cast → fewer exp.

```
② SmartPhone sp = (SmartPhone) myPhone;
```

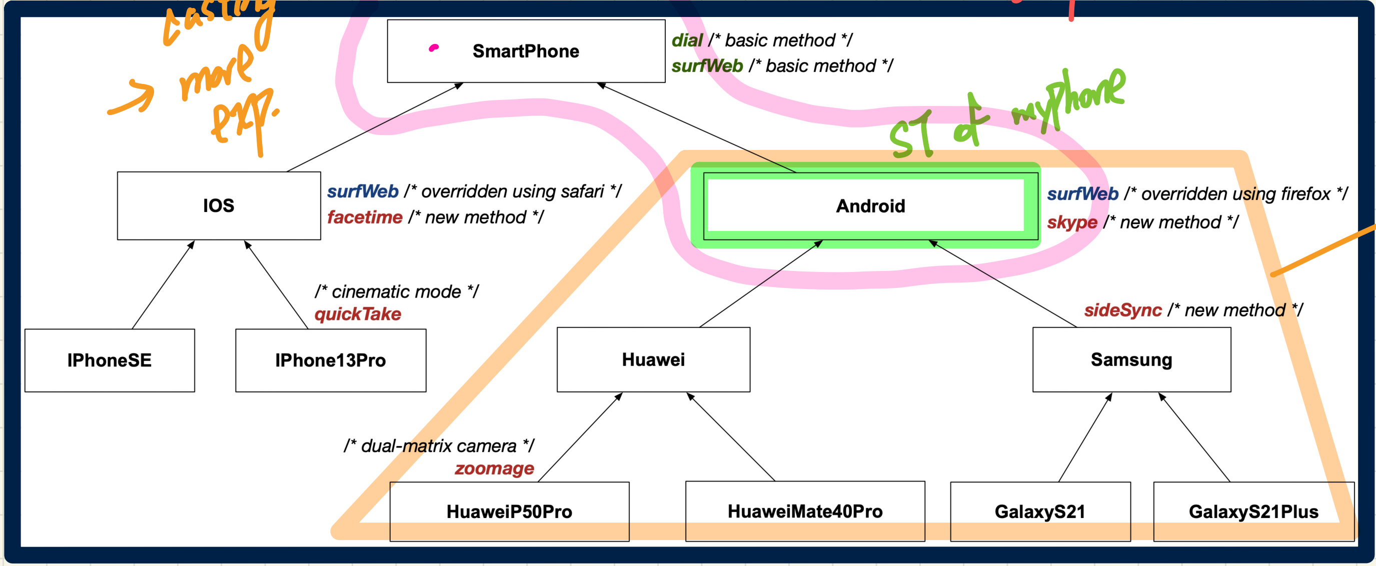
```
③ GalaxyS21Plus ga = (GalaxyS21Plus) myPhone;
```

result of downward casting → more exp.

ancestor classes.

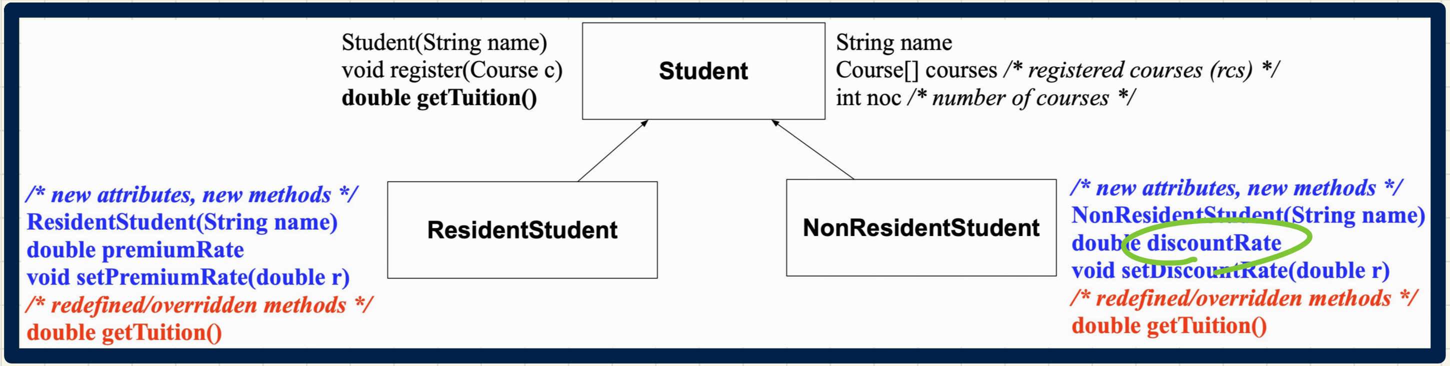
Expectations

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



polymorphism: descendants of myPhone's ST can be its DTs.

Compilable Type Cast May Fail at Runtime (1)


Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] courses /* registered courses (rcs) */
int noc /* number of courses */
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

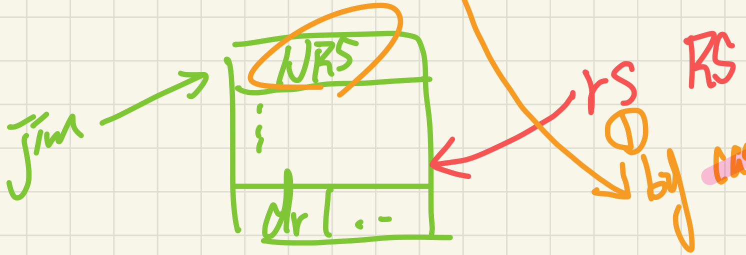
```
Student jim = new NonResidentStudent("J. Davis");
ResidentStudent rs = (ResidentStudent) jim;
rs.setPremiumRate(1.5);
```

downward cast

② the DT of jim (NRS) cannot fulfill expe of RS

ST: RS

① + ②
↳ ClassCastExp.



① by casting jim to an alias of ST RS, we intend to call methods within the exp. of RS

ST: C1
DT: C2

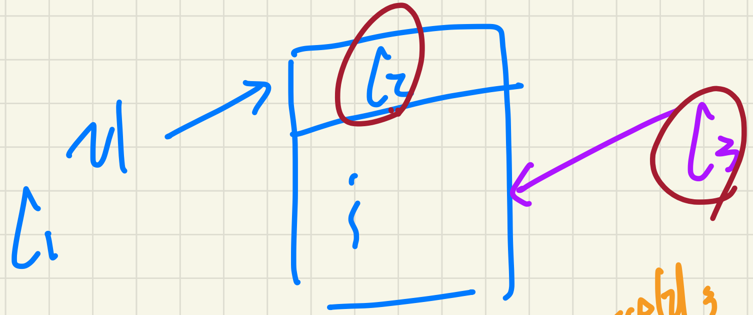
C1 v1 = new C2(..);

C3 v2 = $\boxed{(C3)\ v1}$;

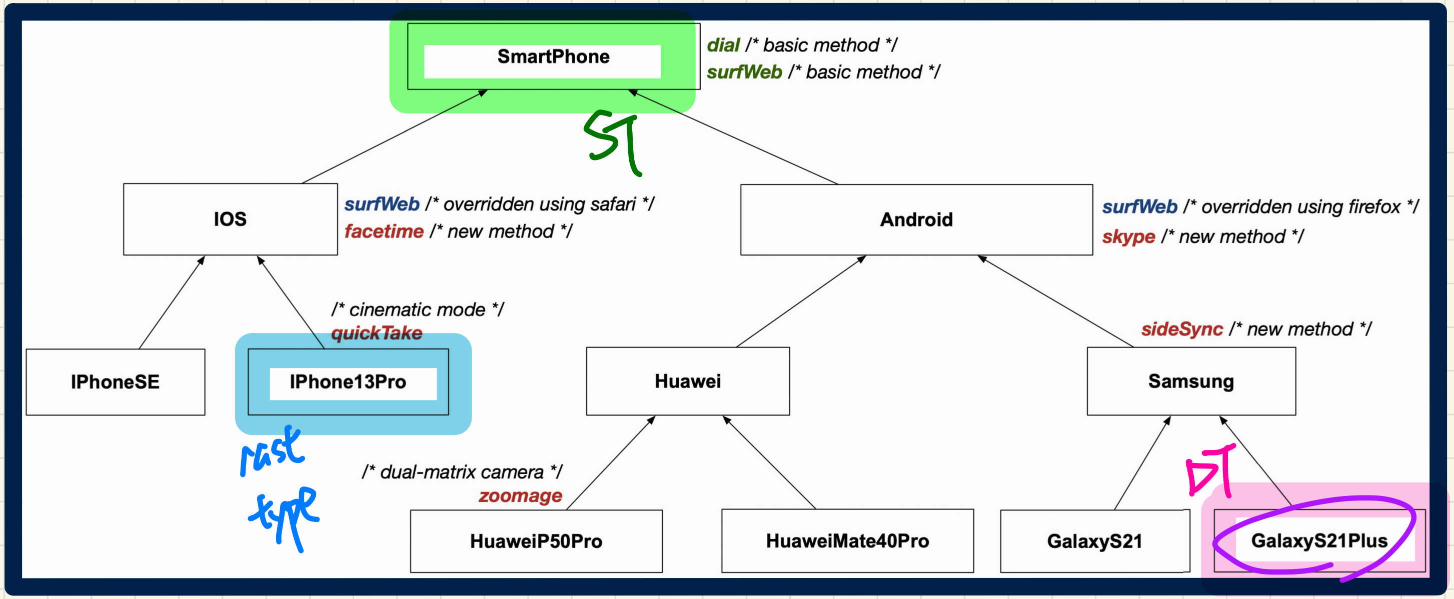
compiles if it's either upward or downward casting.

a ClassCastException occurs if DT of v1 (C2) cannot fulfill expectation of cast type (C3)

more precisely; when C2 is not a descendant of C3.



Compilable Type Cast May Fail at Runtime (2)



```
SmartPhone aPhone = new GalaxyS21Plus();
IPhone13Pro forHeeyeon = (IPhone13Pro) aPhone;
forHeeyeon.quickTake();
```

Every line compiles!

within exp. of ST of C.O. (forHeeyeon)

compiles ∵ downward cast

① DT of aPhone: GalaxyS21 Plus

② cast type has expectation of IPhone13Pro

③ CCE ∵ DT cannot fulfill exp. of cast type.