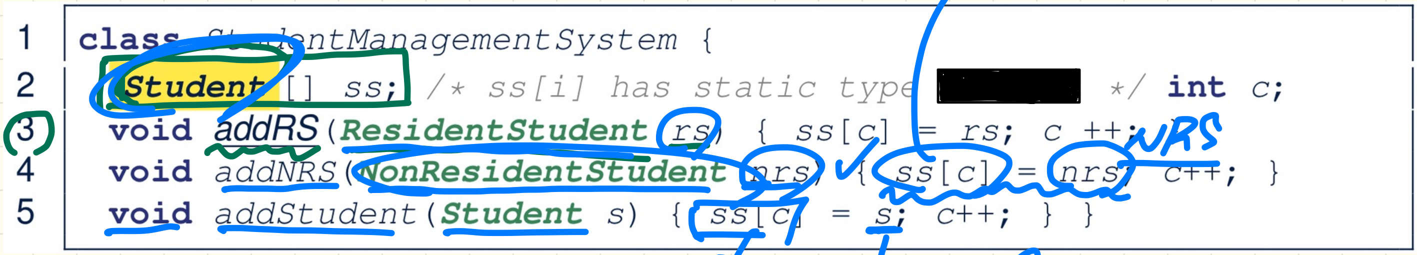
# LECTURE 20

FRIDAY NOVEMBER 15

# Polymorphic Arguments (1)



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

Student

NRS

Student

ST: S.

Q. Static type of ss[0], ss[1], ..., ss[ss.length - 1]?

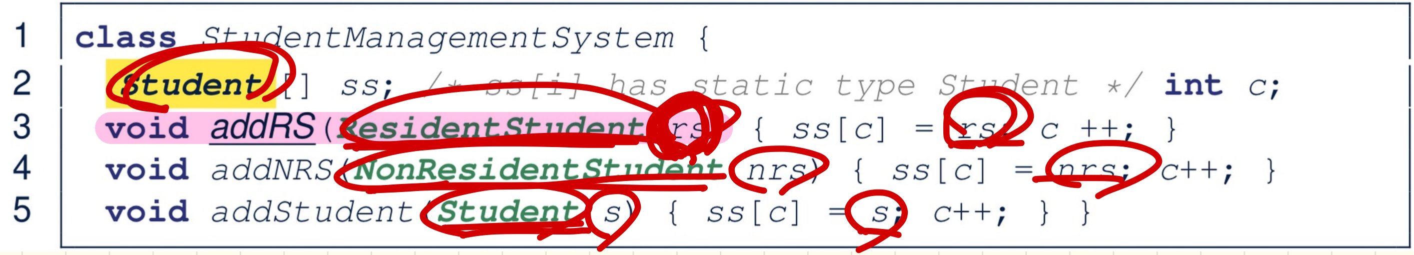
Student

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

✓

ss[c] = rs ;

Student ST: RS



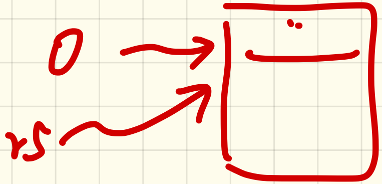
```
class StudentManagementSystem {
  Student[] ss; /* ss[i] has static type Student */ int c;
  void addRS(ResidentStudent rs) { ss[c] = rs; c ++; }
  void addNRS(NonResidentStudent nrs) { ss[c] = nrs; c++; }
  void addStudent(Student s) { ss[c] = s; c++; } }
```

SMS sms = new SMS();

:

sms.addRS(o);

rs = o;



# Polymorphic Arguments (2)

parameter: ST RS

rs = s1.

```
class StudentManagementSystem {
  Student [] ss; /* ss[i] has static type Student */ int c;
  void addRS(ResidentStudent rs) { ss[c] = rs; c ++; }
  void addNRS(NonResidentStudent nrs) { ss[c] = nrs; c++; }
  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);    ✓
```

argument: ST Student

not compile ∵ ST of s1 (argument) not a descendant of ST of rs (param).

ST: descendants of ST of param of addStud.

# Casting Arguments

addRS(RS rs)

sms.addRS((ResidentStudent) s) compiles?

valid down cast with ST: RS

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

1

(ResidentStudent) S ✓

ClassCastException?

YES ∵ DT Student cannot fulfill RS exp.

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

2

(RS) S ✓

can't fulfil

DT

ClassCastException?

YES.

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

3

(RS) S ✓

can fulfil

ClassCastException?

NO

---

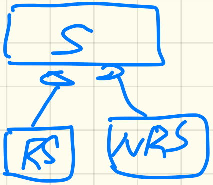
(RS) nrs

sms.addRS((ResidentStudent) nrs) compiles?

ST: NRS

NO.

```
NonResidentStudent nrs = new NonResidentStudent();
/* ST: NonResidentStudent; DT: NonResidentStudent */
StudentManagementSystem sms = new StudentManagementSystem();
sms.addRS(nrs);
```



```
class SMS {
    void addRS(RS rs) { - - }
}
```

```
Student s = new - -

sms.addRS((RS) s)
```

valid down cast with ST RS

whether DT of s can fulfill expec. of RS.

should've done

```
if (s instanceof RS) {
    sms.addRS((RS) s);
}
```

# A Polymorphic Collection of Students (1)

```
ResidentStudent rs = new ResidentStudent("Rachael");
rs.setPremiumRate(1.5);
NonResidentStudent nrs = new NonResidentStudent("Nancy");
nrs.setDiscountRate(0.5);
StudentManagementSystem sms = new StudentManagementSystem();
sms.addStudent( rs ); /* polymorphism */
sms.addStudent( nrs ); /* polymorphism */
Course eecs2030 = new Course("EECS2030", 500.0);
sms.registerAll(eecs2030);
for(int i = 0; i < sms.numberOfStudents; i ++) {
  /* Dynamic Binding:
   * Right version of getTuition will be called */
  System.out.println(sms.students[i]. getTuition() );
}
```

DT: NRS

DT: RS ← sms. ss[0]. getTuition();

call version getT in RS

sms. ss[1]. getTuition();

nrs

~~rs~~

void addStudent(S ~~x~~) {

ss[c] = s;

} c++;

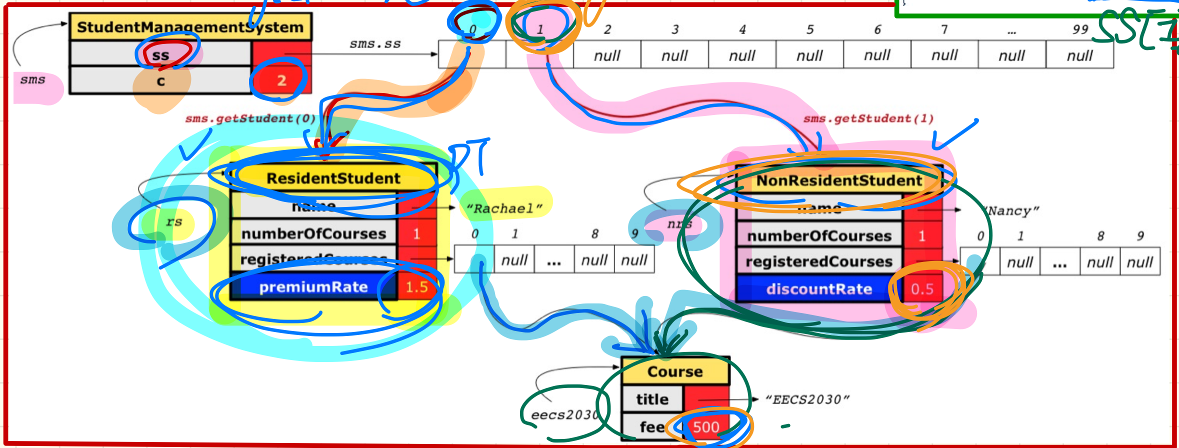
```
class StudentManagementSystem {
  Student[] students;
  int numOfStudents;

  void addStudent(Student s) {
    students[numOfStudents] = s;
    numOfStudents ++;
  }

  void registerAll (Course c) {
    for(int i = 0; i < numberOfStudents; i ++) {
      students[i].register(c)
    }
  }
}
```

ss

ss

ss[0]. register(c);

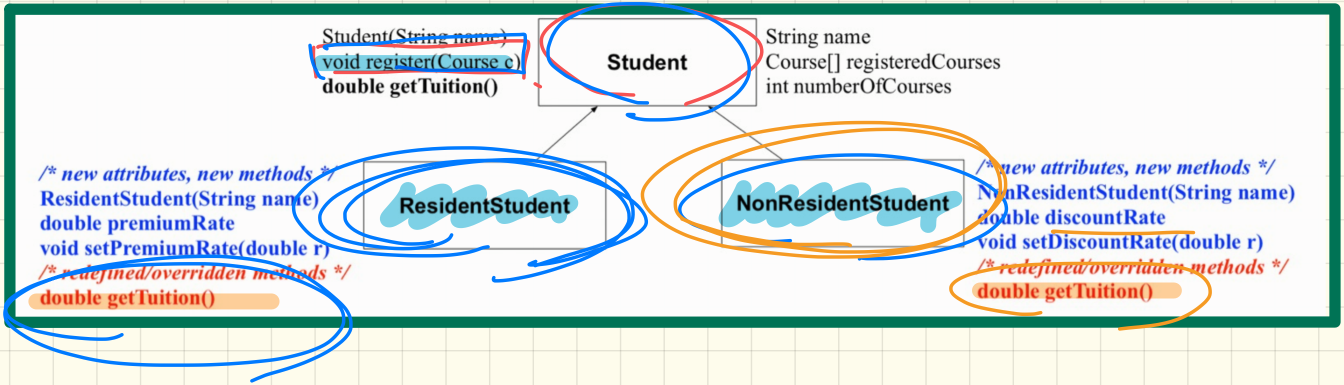
ss[1]. register(c);



ss[0] = rs;

ss[1] = nrs;

# Reference: Hierarchy of Students



Student(String name)
void register(Course c)
**double getTuition()**

String name
Course[] registeredCourses
int numberOfCourses

*/* new attributes, new methods */*
**ResidentStudent(String name)**
**double premiumRate**
**void setPremiumRate(double r)**
*/* redefined/overridden methods */*
**double getTuition()**

*/* new attributes, new methods */*
**NonResidentStudent(String name)**
**double discountRate**
**void setDiscountRate(double r)**
*/* redefined/overridden methods */*
**double getTuition()**

# A Polymorphic Collection of Students (2)

```
ResidentStudent rs = new ResidentStudent("Rachael");
rs.setPremiumRate(1.5);
NonResidentStudent nrs = new NonResidentStudent("Nancy");
nrs.setDiscountRate(0.5);
StudentManagementSystem sms = new StudentManagementSystem();
sms.addStudent( rs ); /* polymorphism */
sms.addStudent( nrs ); /* polymorphism */
Course eecs2030 = new Course("EECS2030", 500.0);
sms.registerAll(eecs2030);
for(int i = 0; i < sms.numberOfStudents; i ++) {
  /* Dynamic Binding:
   * Right version of getTuition will be called */
  System.out.println(sms.students[i]. getTuition() );
}
```

ST: Student

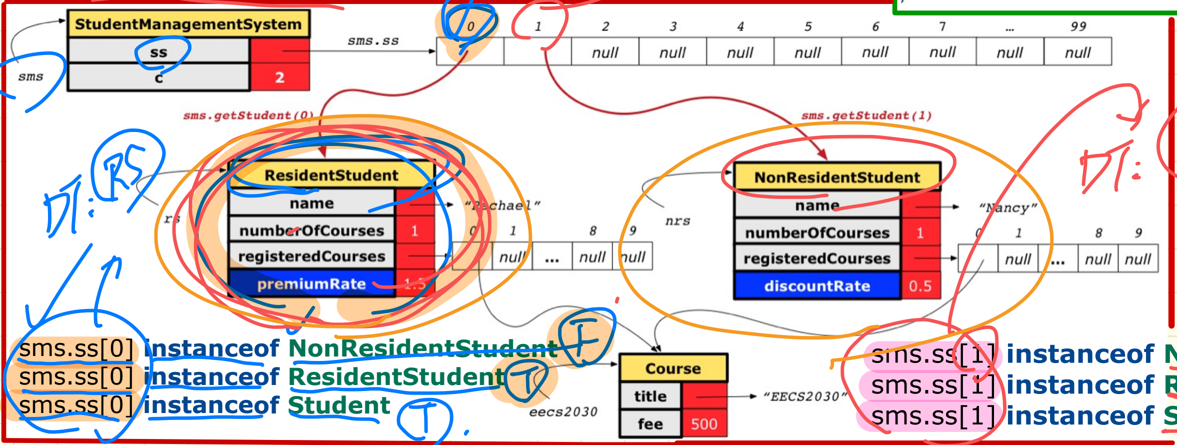
✓ sms . ss[0] . register( eecs2030);

~~sms. ss[0]. setPr(1.8)~~

↳ ST: Student

~~DT: RS~~

```
class StudentManagementSystem {
  Student[] students;
  int numOfStudents;

  void addStudent(Student s) {
    students[numOfStudents] = s;
    numOfStudents ++;
  }

  void registerAll (Course c) {
    for(int i = 0; i < numberOfStudents; i ++) {
      students[i].register(c)
    }
  }
}
```



sms.ss[0] instanceof NonResidentStudent F

sms.ss[0] instanceof ResidentStudent T

sms.ss[0] instanceof Student T

sms.ss[1] instanceof NonResidentStudent T

sms.ss[1] instanceof ResidentStudent F

sms.ss[1] instanceof Student T

```
class SMS {
  Student ss[] . . -
}
```

```
SMS.ss[0].setPr(. . -)
```

ST: Student

not declared (not part of expectation)

# Polymorphic Return Values

```
class StudentManagementSystem {
  Student[] ss; int c;
  void addStudent(Student s) { ss[c] = s; c++; }
  Student getStudent(int i) {
   Student s = null;
   if(i < 0 || i >= c) {
     throw new IllegalArgumentException("Invalid
   }
   else {
     s = ss[i];
   }
   return s;
  } }
```

```
Course eecs2030 = new Course("EECS2030", 500);
ResidentStudent rs = new ResidentStudent("Rachael");
rs.setPremiumRate(1.5); rs.register(eecs2030);
NonResidentStudent nrs = new NonResidentStudent("Nancy");
nrs.setDiscountRate(0.5); nrs.register(eecs2030);
StudentManagementSystem sms = new StudentManagementSystem();
sms.addStudent(rs); sms.addStudent(nrs);
Student s = sms.getStudent(0) ; /* dynamic type of s? */

             static return type: Student
print(s instanceof Student && s instanceof ResidentStudent);/*true*/
print(s instanceof NonResidentStudent); /* false */
print(s.getTuition());/*Version in ResidentStudent called:750*/
ResidentStudent rs2 = sms.getStudent(0);  ×
s =    sms.getStudent(1)  ;  /* dynamic type of s? */

      static return type: Student
print(s instanceof Student && s instanceof NonResidentStudent);/*true*/
print(s instanceof ResidentStudent);  /* false */
print( s.getTuition() );/*Version in NonResidentStudent called:250*/
NonResidentStudent nrs2 = sms.getStudent(1); ×
```

DT: RS

Student s = sms.getStudent(0);

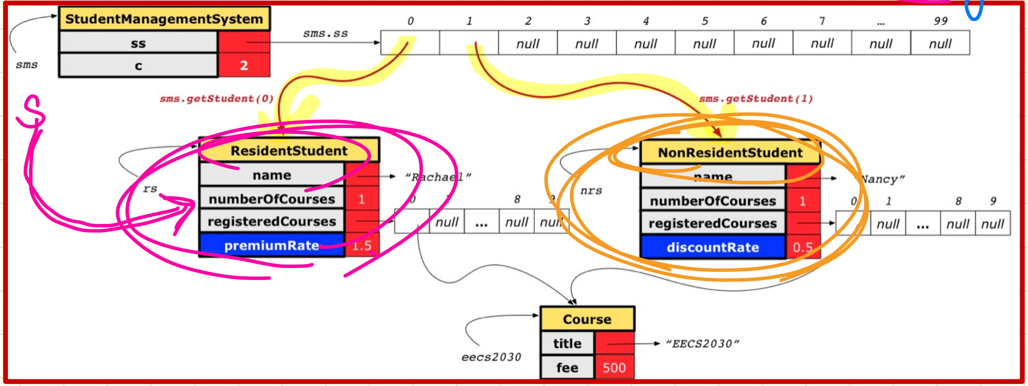
ST: Student

T

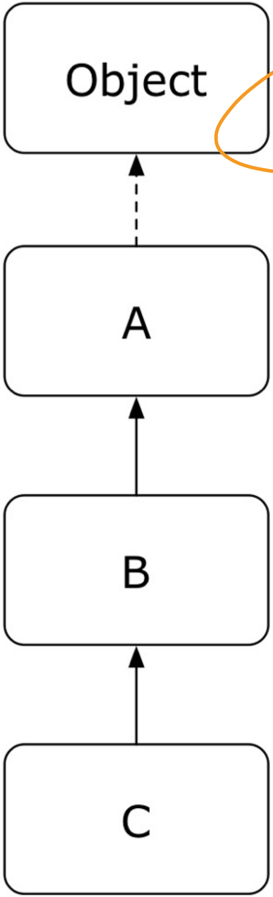
sms.getStudent(0) instanceof ~~ResidentStudent~~

NRS

F



# Overridden Methods and Dynamic Binding (1)



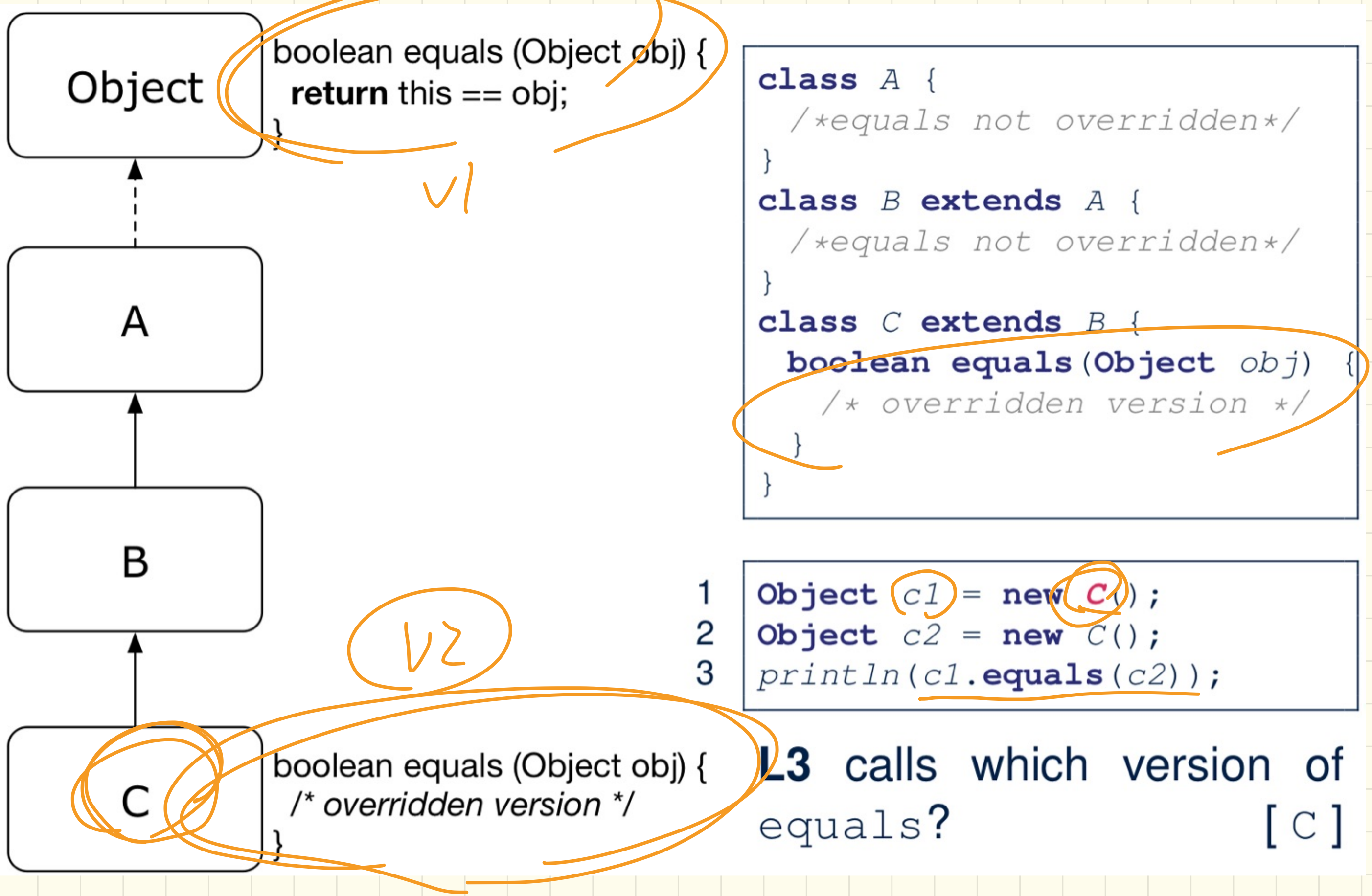
```
boolean equals (Object obj) {
  return this == obj;
}
```

```
class A {
  /*equals not overridden*/
}
class B extends A {
  /*equals not overridden*/
}
class C extends B {
  /*equals not overridden*/
}
```

```
1  Object c1 = new C();
2  Object c2 = new C();
3  println(c1.equals(c2));
```

**L3** calls which version of `equals`? [Object]

# Overridden Methods and Dynamic Binding (2)



```
boolean equals (Object obj) {
  return this == obj;
}
```

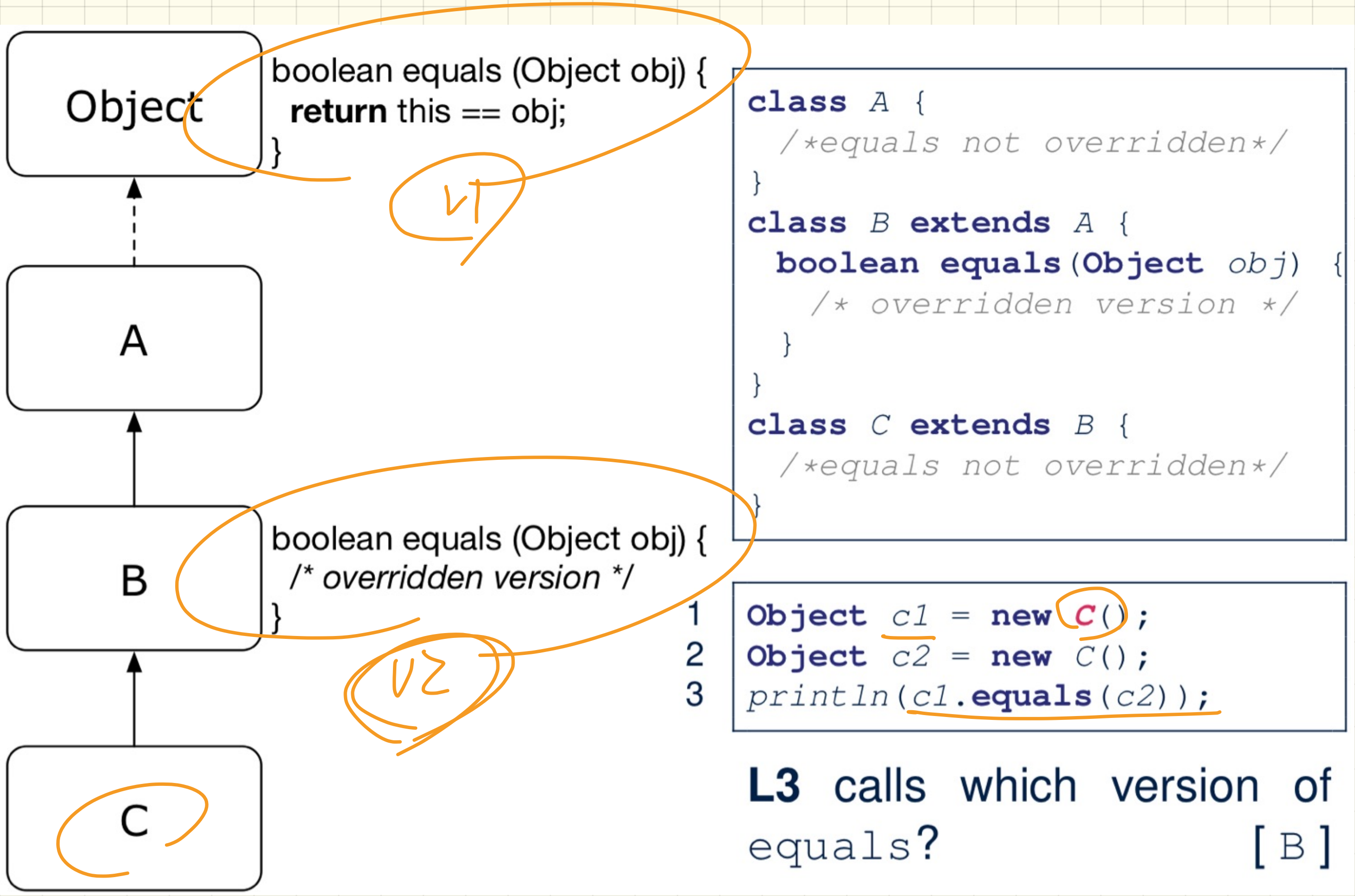
```
boolean equals (Object obj) {
  /* overridden version */
}
```

```
class A {
  /*equals not overridden*/
}
class B extends A {
  /*equals not overridden*/
}
class C extends B {
  boolean equals(Object obj) {
    /* overridden version */
  }
}
```

```
1  Object c1 = new C();
2  Object c2 = new C();
3  println(c1.equals(c2));
```

L3 calls which version of equals? [ C ]

# Overridden Methods and Dynamic Binding (3)



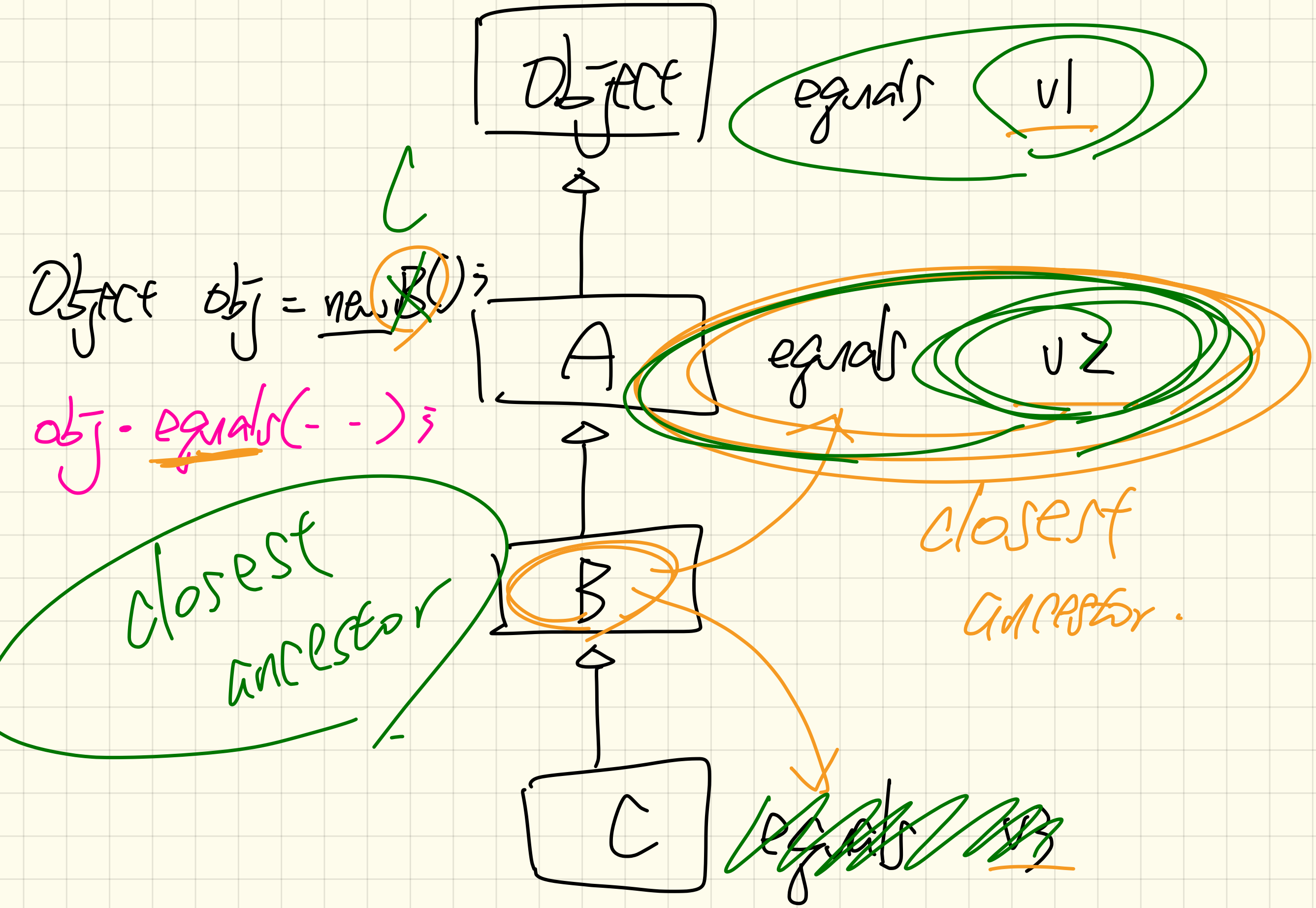
```
boolean equals (Object obj) {
  return this == obj;
}
```

```
boolean equals (Object obj) {
  /* overridden version */
}
```

```
class A {
  /*equals not overridden*/
}
class B extends A {
  boolean equals(Object obj) {
    /* overridden version */
  }
}
class C extends B {
  /*equals not overridden*/
}
```

```
1  Object c1 = new C();
2  Object c2 = new C();
3  println(c1.equals(c2));
```

**L3** calls which version of equals? [ B ]



C

Object obj = new ~~B~~();

obj.equals(- -);