

Wednesday Nov. 14
Lecture 19

Polymorphic Arguments (1)

```
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++; } }
```

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

Q. In addRS: does $ss[c] = rs$ compile?

Polymorphic Arguments (2)

```
1 class StudentManagementSystem {  
2     Student[] ss; // 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++; } }
```

```
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); ● → X  
sms.addRS(rs); ●  
sms.addRS(nrs); ●  
sms.addStudent(s1); ● ✓  
sms.addStudent(s2); ●  
sms.addStudent(s3); ●  
sms.addStudent(rs); ●  
sms.addStudent(nrs); ●
```

$S \neq s1$
 S $s2$

$\frac{RS}{RS} = \frac{S3}{St}$

Casting Arguments

`sms.addRS((ResidentStudent)s);`

```

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

Annotations: `RS temp = (RS)s;`, `sms.addRS(temp);`, `sms.addRS((RS)s);`.
 Diagrams: `ST: Student`, `RS = S`, `ST: Student`.

Compiles?
 ClassCastException?
`DT: Student` → CCE

```

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

Annotations: `sms.addRS((CRS)s);`.
 Diagrams: `DT: NRS`.

ClassCastException?
`DT: NRS` → CCE

```

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

Annotations: `sms.addRS((CRS)s);`.
 Diagrams: `DT: RS`.

ClassCastException?
`DT: RS` → No CCE

```

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

`sms.addRS((ResidentStudent)nrs);`
 Compiles?

`ST: NRS`
 No
 Compiles?

A Polymorphic Collection of Students

if (at runtime: F Instance of NRS) {
 sms.ss[0] instance of NRS }
 sms.ss[1] instance of Student
 sms.ss[2] instance of ResidentStudent
 sms.ss[3] instance of ResidentStudent
 sms.ss[4] instance of ResidentStudent
 sms.ss[5] instance of ResidentStudent
 sms.ss[6] instance of ResidentStudent
 sms.ss[7] instance of ResidentStudent
 sms.ss[8] instance of ResidentStudent
 sms.ss[9] instance of ResidentStudent
 sms.ss[10] instance of ResidentStudent
 sms.ss[11] instance of ResidentStudent
 sms.ss[12] instance of ResidentStudent
 sms.ss[13] instance of ResidentStudent
 sms.ss[14] instance of ResidentStudent
 sms.ss[15] instance of ResidentStudent
 sms.ss[16] instance of ResidentStudent
 sms.ss[17] instance of ResidentStudent
 sms.ss[18] instance of ResidentStudent
 sms.ss[19] instance of ResidentStudent
 sms.ss[20] instance of ResidentStudent
 sms.ss[21] instance of ResidentStudent
 sms.ss[22] instance of ResidentStudent
 sms.ss[23] instance of ResidentStudent
 sms.ss[24] instance of ResidentStudent
 sms.ss[25] instance of ResidentStudent
 sms.ss[26] instance of ResidentStudent
 sms.ss[27] instance of ResidentStudent
 sms.ss[28] instance of ResidentStudent
 sms.ss[29] instance of ResidentStudent
 sms.ss[30] instance of ResidentStudent
 sms.ss[31] instance of ResidentStudent
 sms.ss[32] instance of ResidentStudent
 sms.ss[33] instance of ResidentStudent
 sms.ss[34] instance of ResidentStudent
 sms.ss[35] instance of ResidentStudent
 sms.ss[36] instance of ResidentStudent
 sms.ss[37] instance of ResidentStudent
 sms.ss[38] instance of ResidentStudent
 sms.ss[39] instance of ResidentStudent
 sms.ss[40] instance of ResidentStudent
 sms.ss[41] instance of ResidentStudent
 sms.ss[42] instance of ResidentStudent
 sms.ss[43] instance of ResidentStudent
 sms.ss[44] instance of ResidentStudent
 sms.ss[45] instance of ResidentStudent
 sms.ss[46] instance of ResidentStudent
 sms.ss[47] instance of ResidentStudent
 sms.ss[48] instance of ResidentStudent
 sms.ss[49] instance of ResidentStudent
 sms.ss[50] instance of ResidentStudent
 sms.ss[51] instance of ResidentStudent
 sms.ss[52] instance of ResidentStudent
 sms.ss[53] instance of ResidentStudent
 sms.ss[54] instance of ResidentStudent
 sms.ss[55] instance of ResidentStudent
 sms.ss[56] instance of ResidentStudent
 sms.ss[57] instance of ResidentStudent
 sms.ss[58] instance of ResidentStudent
 sms.ss[59] instance of ResidentStudent
 sms.ss[60] instance of ResidentStudent
 sms.ss[61] instance of ResidentStudent
 sms.ss[62] instance of ResidentStudent
 sms.ss[63] instance of ResidentStudent
 sms.ss[64] instance of ResidentStudent
 sms.ss[65] instance of ResidentStudent
 sms.ss[66] instance of ResidentStudent
 sms.ss[67] instance of ResidentStudent
 sms.ss[68] instance of ResidentStudent
 sms.ss[69] instance of ResidentStudent
 sms.ss[70] instance of ResidentStudent
 sms.ss[71] instance of ResidentStudent
 sms.ss[72] instance of ResidentStudent
 sms.ss[73] instance of ResidentStudent
 sms.ss[74] instance of ResidentStudent
 sms.ss[75] instance of ResidentStudent
 sms.ss[76] instance of ResidentStudent
 sms.ss[77] instance of ResidentStudent
 sms.ss[78] instance of ResidentStudent
 sms.ss[79] instance of ResidentStudent
 sms.ss[80] instance of ResidentStudent
 sms.ss[81] instance of ResidentStudent
 sms.ss[82] instance of ResidentStudent
 sms.ss[83] instance of ResidentStudent
 sms.ss[84] instance of ResidentStudent
 sms.ss[85] instance of ResidentStudent
 sms.ss[86] instance of ResidentStudent
 sms.ss[87] instance of ResidentStudent
 sms.ss[88] instance of ResidentStudent
 sms.ss[89] instance of ResidentStudent
 sms.ss[90] instance of ResidentStudent
 sms.ss[91] instance of ResidentStudent
 sms.ss[92] instance of ResidentStudent
 sms.ss[93] instance of ResidentStudent
 sms.ss[94] instance of ResidentStudent
 sms.ss[95] instance of ResidentStudent
 sms.ss[96] instance of ResidentStudent
 sms.ss[97] instance of ResidentStudent
 sms.ss[98] instance of ResidentStudent
 sms.ss[99] instance of ResidentStudent

```

1 ResidentStudent rs = new ResidentStudent("Rachael");
2 rs.setPremiumRate(1.5);
3 NonResidentStudent nrs = new NonResidentStudent("Nancy");
4 nrs.setDiscountRate(0.5);
5 StudentManagementSystem sms = new StudentManagementSystem();
6 sms.addStudent(rs); /* polymorphism */
7 sms.addStudent(nrs); /* polymorphism */
8 Course eecs2030 = new Course("EECS2030", 500.0);
9 sms.registerAll(eecs2030);
10 for(int i = 0; i < sms.numberOfStudents; i++) {
11     /* Dynamic Binding:
12     * Right version of getTuition will be called */
13     System.out.println(sms.students[i].getTuition());
14 }
    
```

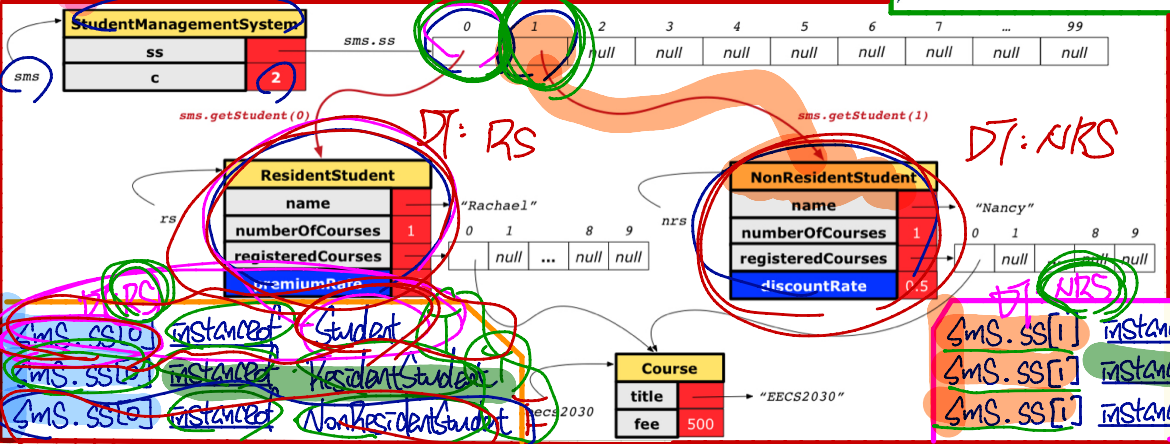
```

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)
        }
    }
}
    
```

ST: Student ST: Student



Handwritten notes at the bottom right:

- Arrows pointing from `sms.ss[0]` to "instance of Student".
- Arrows pointing from `sms.ss[1]` to "instance of ResidentStudent".
- Arrows pointing from `sms.ss[i]` to "instance of NonResidentStudent".
- Handwritten "NRS" and "Student" labels.

Polymorphic Return Values

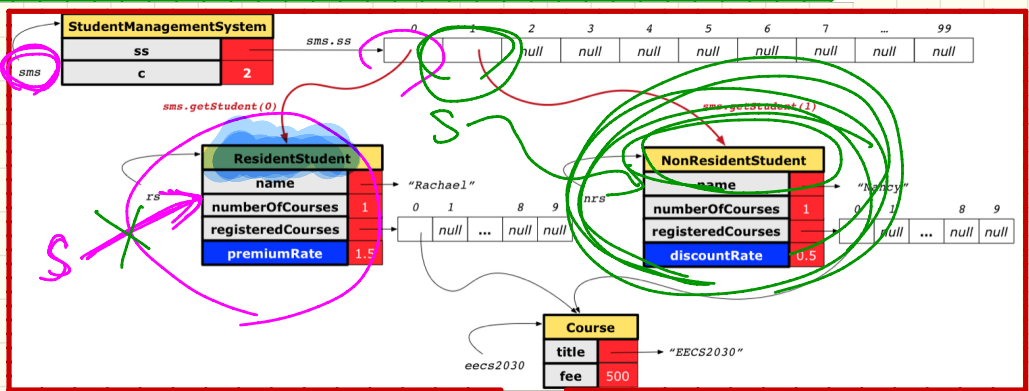
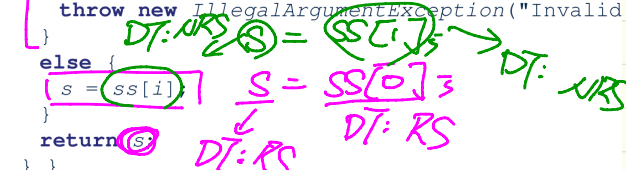
return type (static)

```

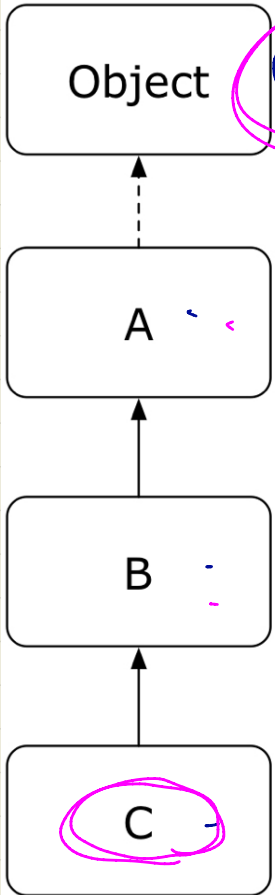
class StudentManagementSystem {
    Student[] ss; int c;
    void addStudent(Student s) { ss[c] = s; c++; }
    Student getStudent(int i) {
        Student s = null;
        if(i < 0 || i >= c) {
            throw new IllegalArgumentException("Invalid");
        }
        else {
            s = ss[i];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



Overridden Method & 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*/  
}
```

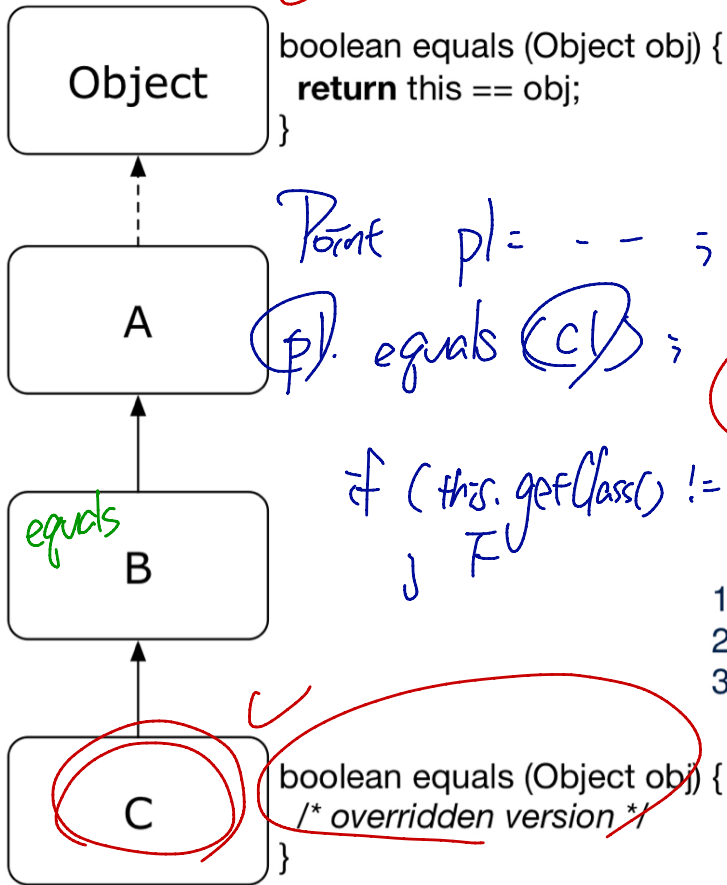
```
1 Object c1 = new C();  
2 Object c2 = new C();  
3 println(c1.equals(c2));
```

Q1: compile?
Q2: version?

L3 calls which version of equals? [Object]

Overridden Method & Dynamic Binding (2)

A



```
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]

Point p1 = new Point (3, 4);

println (p1);

DT: C
02.equals(03);
03.equals(02);

println (p1.toString());

Object equals

↑
A

↑

B equals

↑

C ←

↑

D equals

Object o1 = new B();

Object o2 = new C();

Object o3 = new D();

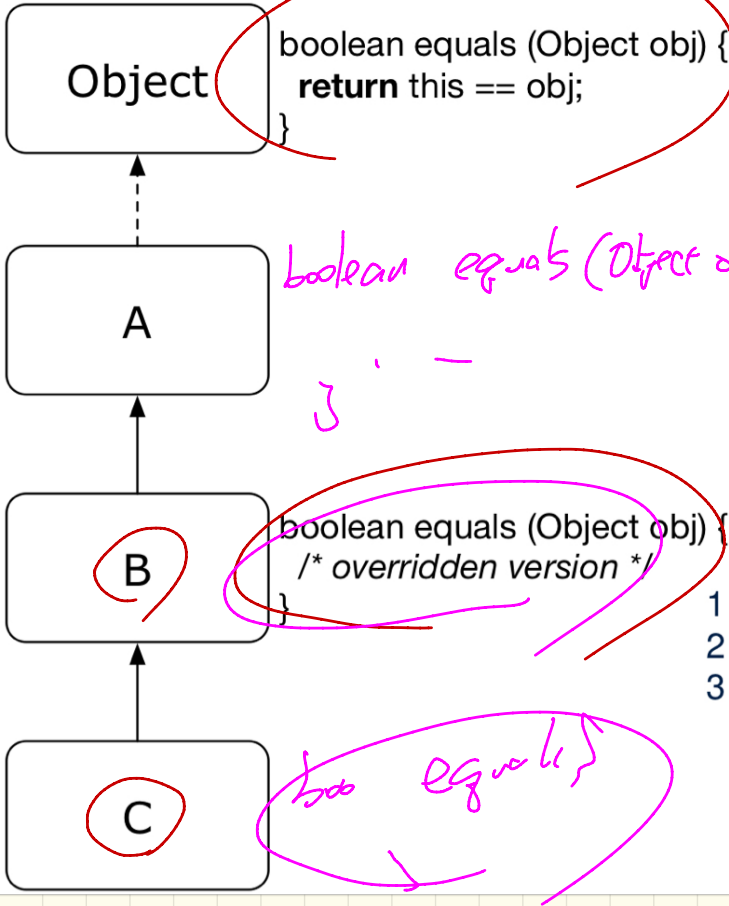
o1.equals(o2);

DT: B

o2.equals(o1);

DT: C

Overridden Method & Dynamic Binding (3)



```
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]