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; /* 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++; } }
```

```
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);
```

Casting Arguments

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

ST: Student RS = S Stud.

sms.addRS((RS) s); DT: Student → CCE.

sms.addRS((ResidentStudent) s); Compiles?

RS temp = (RS) s; sms.addRS(temp); ST: St.

ClassCastException?

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

pv

sms.addRS((RS) s); DT: NRS → CCE

ClassCastException?

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

pv

sms.addRS((RS) s); DT: RS → No CCE

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

ST: NRS

✓ sms.addRS((ResidentStudent) nrs); No Compiles?

A Polymorphic Collection of Students

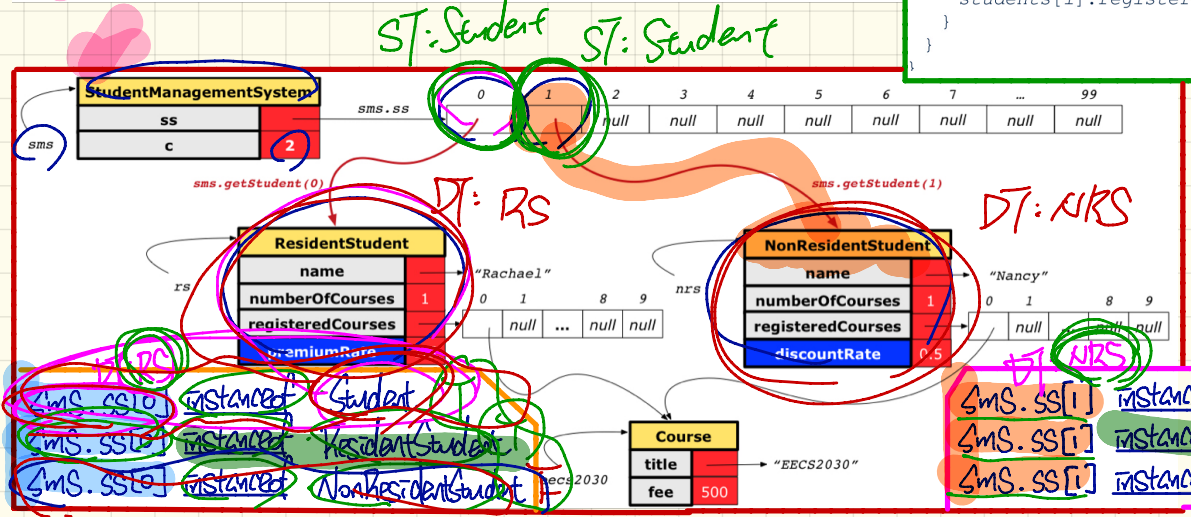
```
1  ResidentStudent rs = new ResidentStudent("Rachael");
2  rs.setPremiumRate(1.5);
3  NonResidentStudent nrs = new NonResidentStudent("Nancy");
4  nrs.setDiscountRate(0.5);
5  StudentManagementSystem sms = new StudentManagementSystem();
6  sms.addStudent(rs); /* polymorphism */
7  sms.addStudent(nrs); /* 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 }
```

if (At runtime : T
sms.ss[0] instanceof NRS) {
NRS nrs = (NRS) sms.ss[0];
}

```
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)
  }
 }
}
```

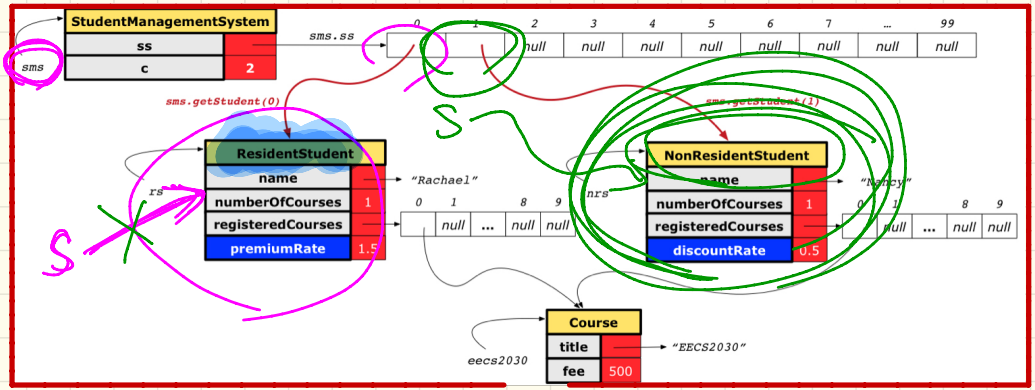


Polymorphic Return Values

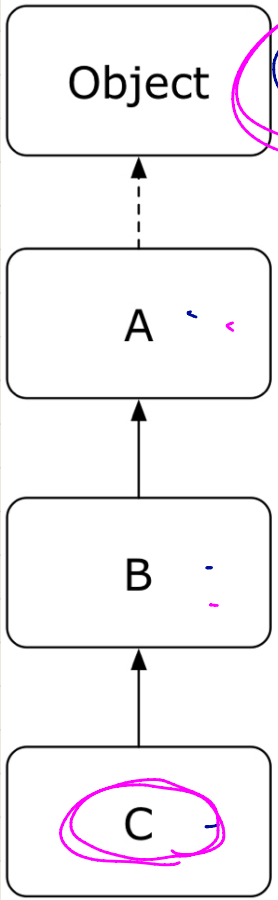
return type (static)

```
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); ×
```

```
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;
  } }
```



Overridden Method & Dynamic Binding (1)




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

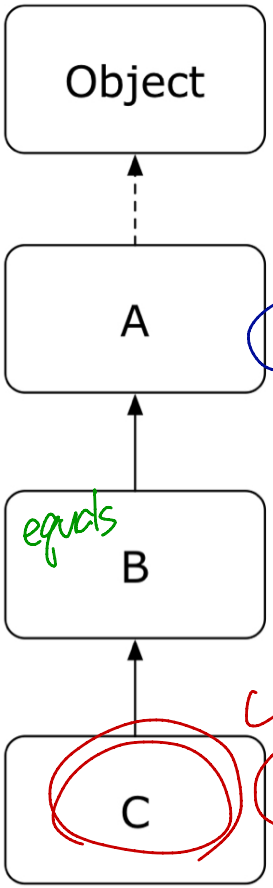
```
1  Object c1 = new C();
2  Object c2 = new C();
3  println(c1.equals(c2));
```

Q1: compile? ✓

Q2: version?

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

Overridden Method & Dynamic Binding (2)




```
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]

```
Point p1 = new Point(3, 4);
println(p1);
println(p1.toString());
```

DT: C

```
o2.equals(o3);
o3.equals(o2);
```

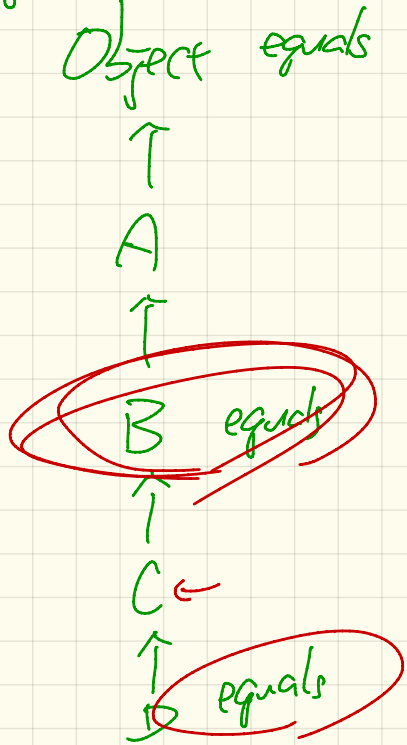
DT: D

Object equals

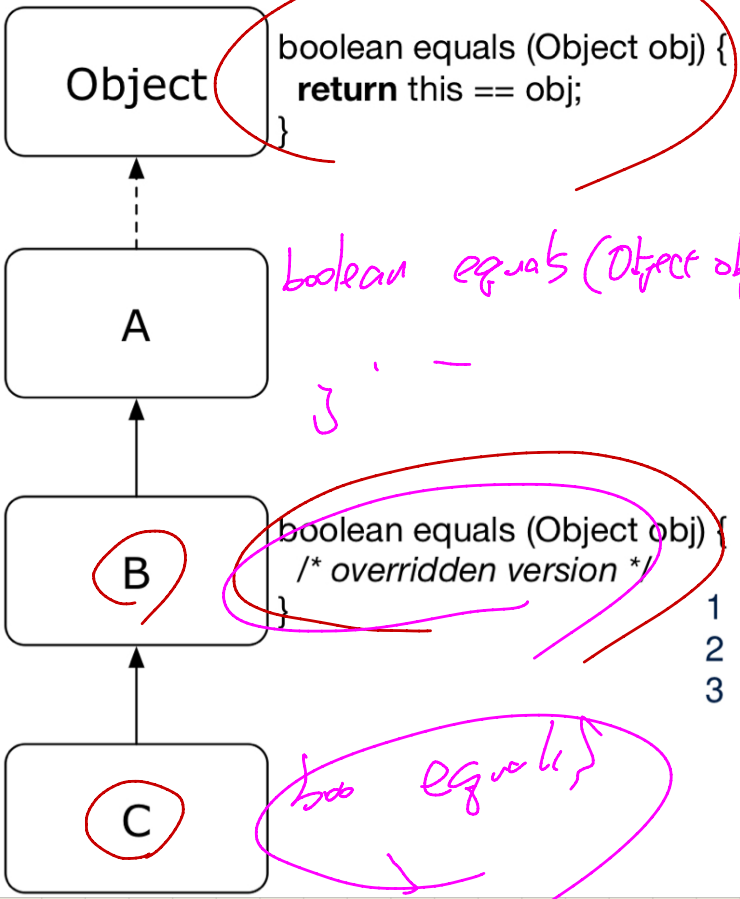
```
Object o1 = new B();
Object o2 = new C();
Object o3 = new D();
o1.equals(o2);
o2.equals(o1);
```

DT: B

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]