Lecture 21 - Nov 23

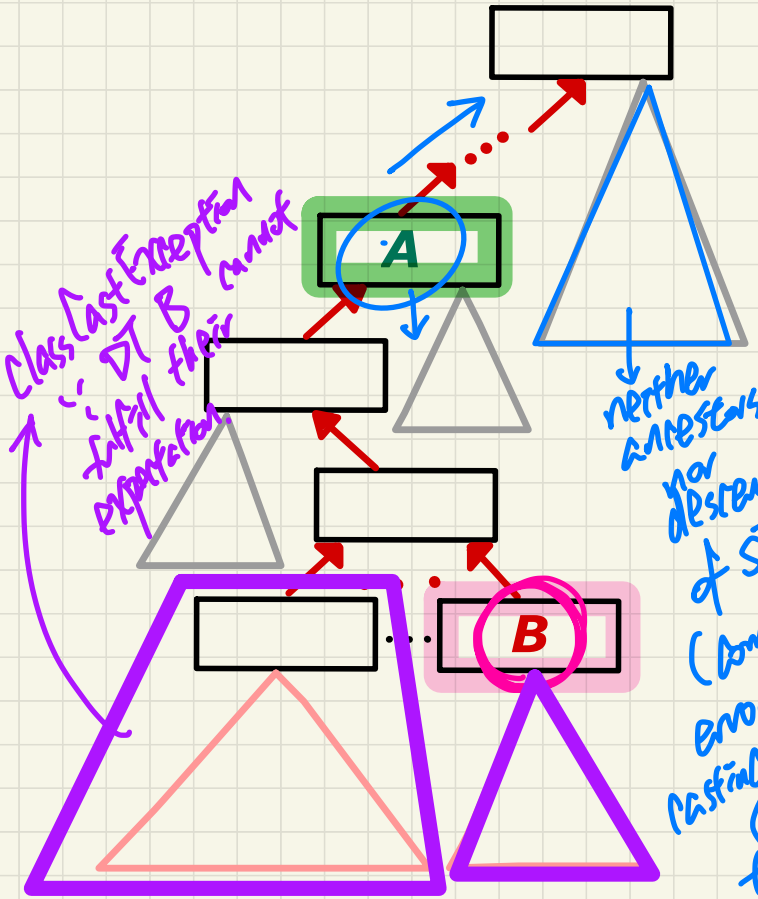
Inheritance

Polymorphic Method Parameters
Polymorphic Method Return Types

Announcements

- Lab4 due soon!
- ProgTest2 results to be released on Thursday
- WT3 and ProgTest3 approaching...

The instanceof Operator




```
1  A obj = new B();
2  if (obj instanceof ??) {
3     ?? obj2 = (??) obj;
   }
```

True if obj's DT can fulfill exp.

thp of ?? ↓ cast type.

cast

- L1 compiles if B can fulfill expectations of A.

- L3:
  - Compiles if Up or Down cast w.r.t. A.
  - ClassCastException if B cannot fulfill expectations on ??.

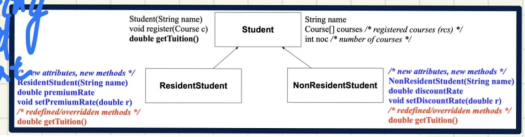
runtime DT!

- L2:
  - Evaluates to true if B can fulfill expectations on ??.

From last lecture...

Polymorphic Parameters (1)

① It has an associated type. Hierarchy
② Each element in array has declared type Student




```
1  class StudentManagementSystem {
2    Student [] ss; /* ss[i] has static type [redacted] */ 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]?

Student

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

valid ⇔ ST of RHS descendant of ST of LHS.

ST: Student ↳ ST: RS

call by value: rs = 0 = para. arg.

Q. Under what circumstances can the following method call be valid/compilable?

sms.addRS(o)

valid: ST of arg. o should be a descendant of para. rs

what should be the type of o?

Polymorphic Parameters (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++; } }
```

call by value:

RS=S1

S1=RS

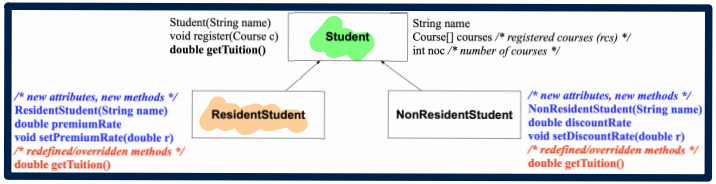
S=S1

```
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); ✓
```

addRS(RS rs)

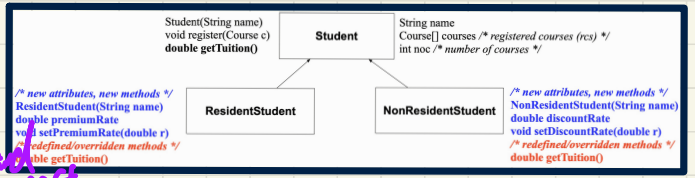
addS(S s)

the higher the ST of parameter is, the more types of arguments it can accept



Casting Arguments

Student s;




```
void addRS(ResidentStudent rs)
```

sms.addRS((ResidentStudent) s) compiles?

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

ClassCastException?

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

ClassCastException?

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

ClassCastException?

sms.addRS((ResidentStudent) nrs) compiles?

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

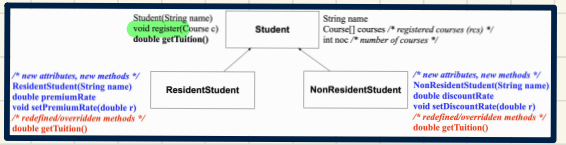
A Polymorphic Collection of Students

```
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 }
```

parameter type: Student, accepting arguments of its descendant classes

dynamic binding

0, 1



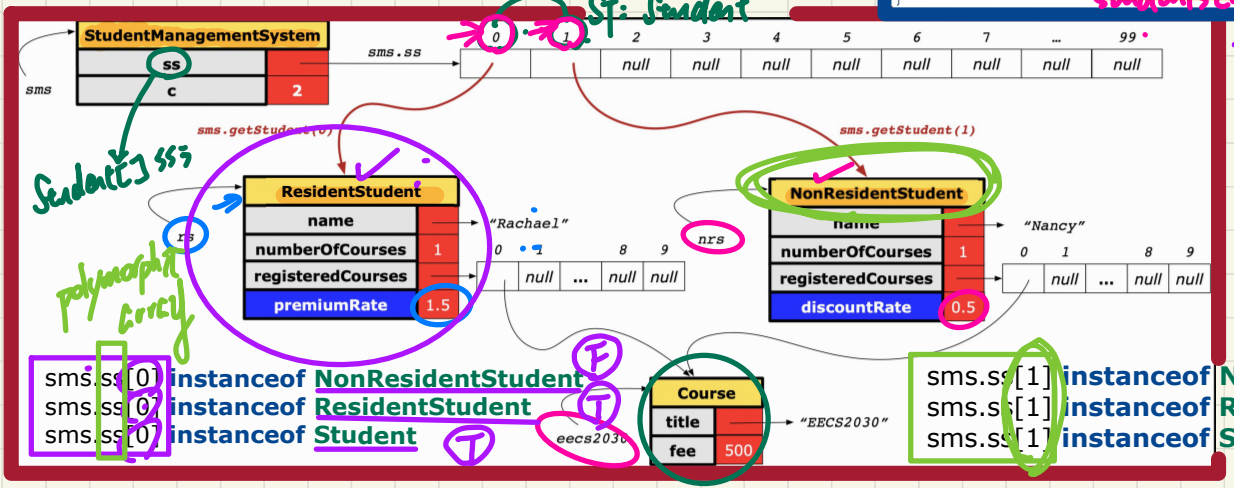

```
class StudentManagementSystem {
  Student[] students;
  int numOfStudents;

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

what if: students[i].getTuition()?

dynamic binding: which version of getTuition method is invoked?

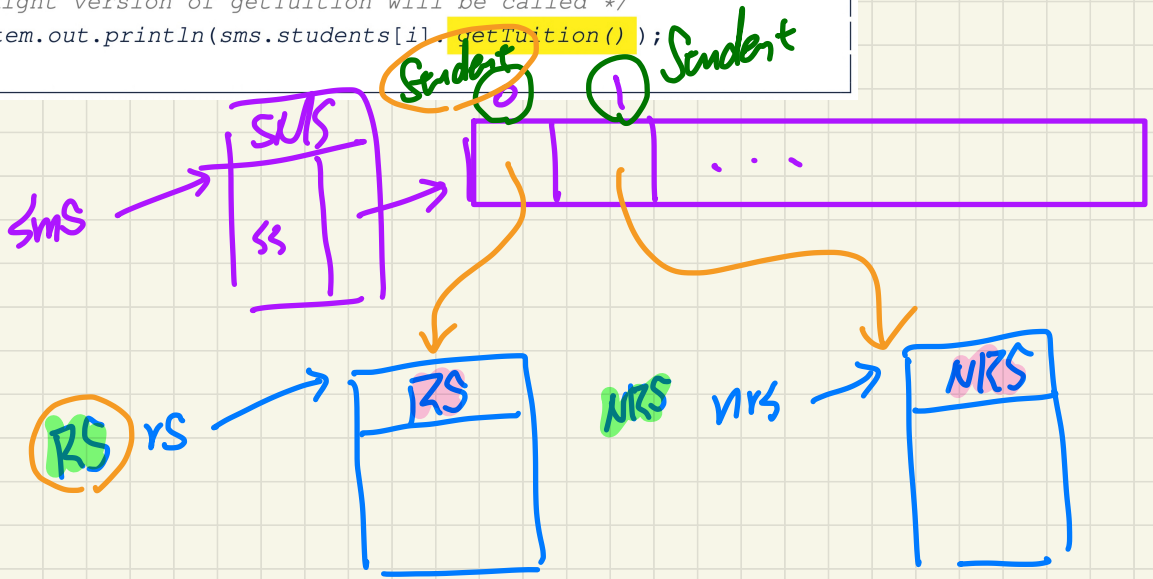
0

students[0].register(c);
students[1].register(c);



Polymorphic array:
- same ST for each elem.
- diff. DTs for elements

```java
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());
}
```

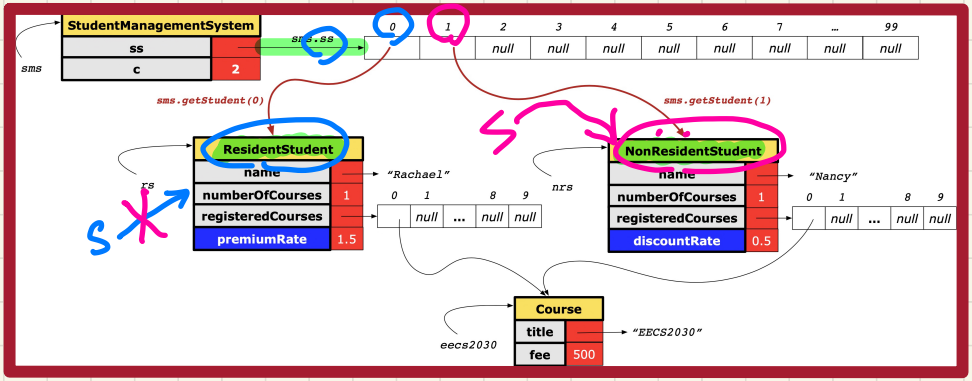


Polymorphic Return Types

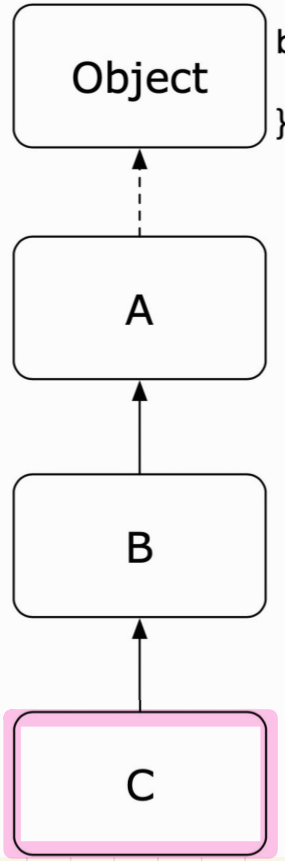
```
Course eecs2030 = new Course("EECS2030", 500);
ResidentStudent rs = new ResidentStudent("Rachael");
rs.setPremiumRate(1.5); rs.register(eecs2030);
NonResidentStudent nrs = new NonResidentStudent("Nancy");
nrs.setDiscountRate(0.5); nrs.register(eecs2030);
StudentManagementSystem sms = new StudentManagementSystem();
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); //tru
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);//
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;
  } }
```

ss is a polymorphic array
① ss[i] has ST : Student
② ss[i] has DT a descendant of ST
ss[0]
ss[1]



Overridden Methods and Dynamic Binding (1)



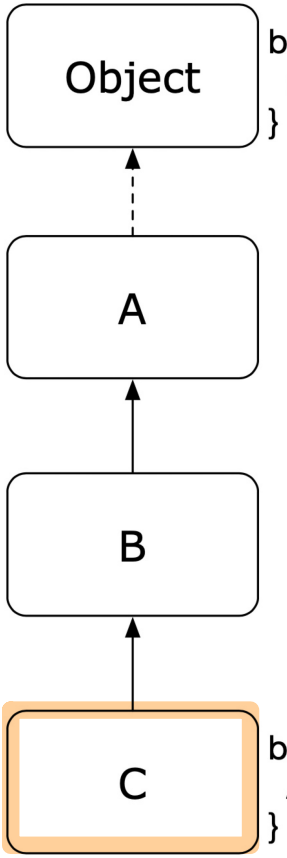

```
boolean equals (Object obj) {
  return this == obj;
}
```

```
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;
}
```

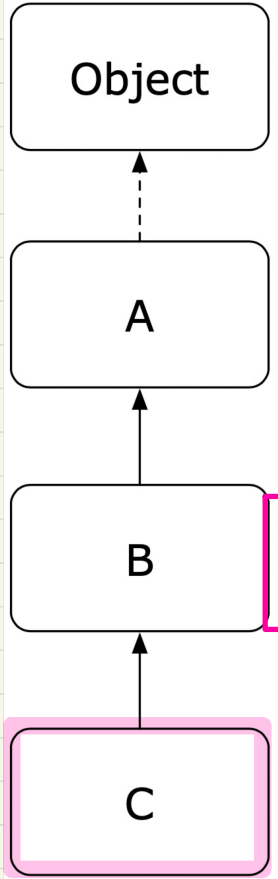
```
boolean equals (Object obj) {
  /* overridden version */
}
```

```
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) {...}

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

overridden version of closest ancestor.

```
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]