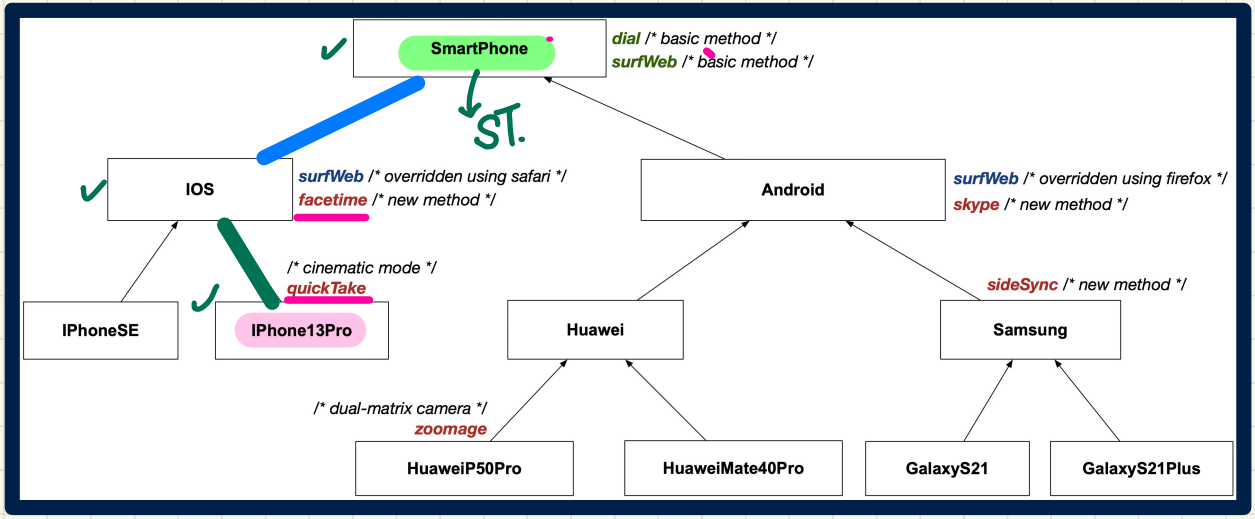
Lecture 5

Part L

Inheritance -
Static Types, Casts, Polymorphism

Static Types, Casts, Polymorphism (1)



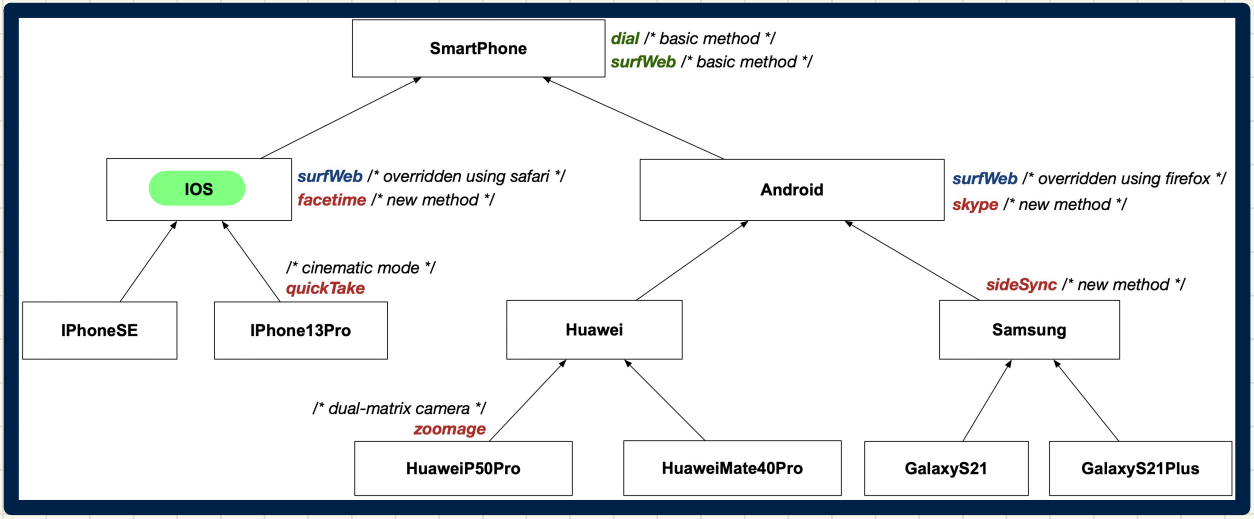
descendant IP13Pro can fulfill exp. of SmartPhone variable

```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

```
1  SmartPhone sp = new IPhone13Pro();   ✓
2  sp.dial();   ✓
3  sp.facetime();   ×
4  sp.quickTake();   ×
```

expectation determined only by ST.

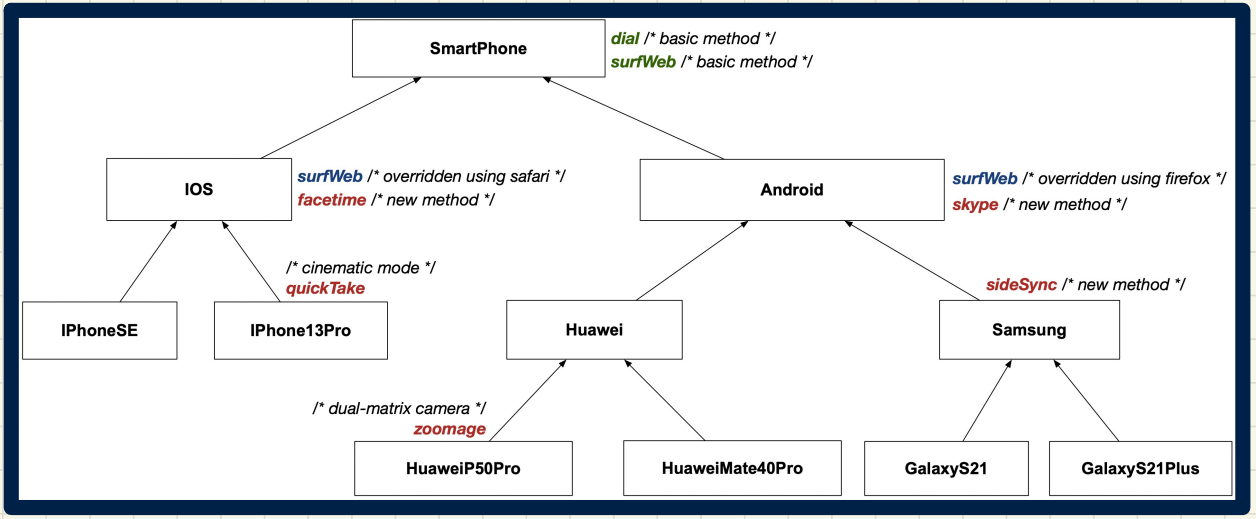
Static Types, Casts, Polymorphism (2)




```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

```
1  IOS ip = new IPhone13Pro();
2  ip.dial();
3  ip.facetime();
4  ip.quickTake();
```

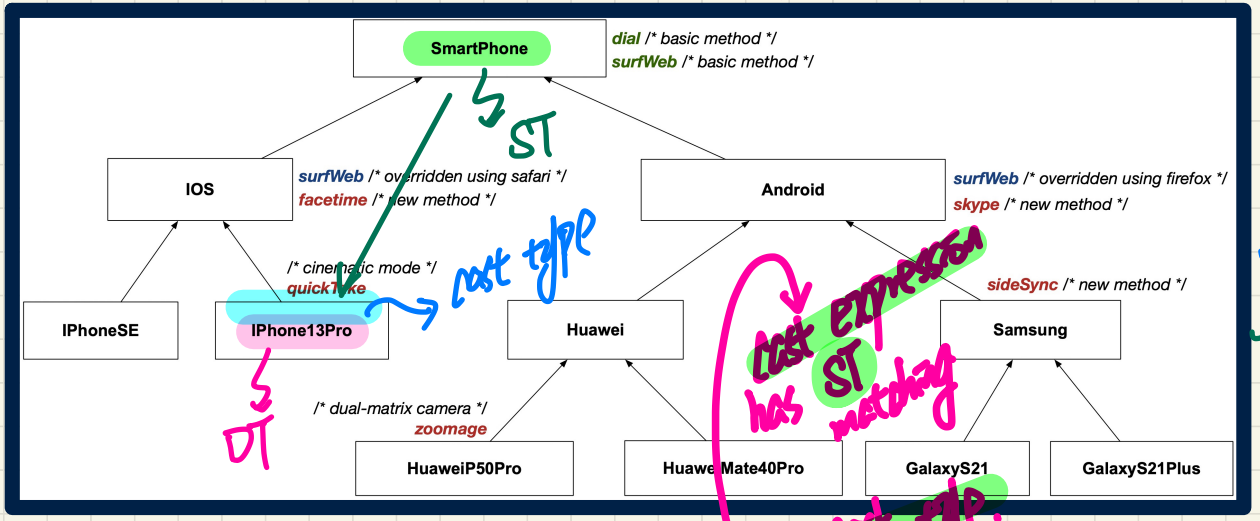
Static Types, Casts, Polymorphism (3)




```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

```
1  IPhone13Pro ip6sp = new IPhone13Pro();
2  ip6sp.dial();
3  ip6sp.facetime();
4  ip6sp.quickTake();
```

Static Types, Casts, Polymorphism (4)



1. Compilation ? w.r.t ST
Valid ? ; downward cast
2. ClassCastException?
Can DT fulfill cast type ? No.

sp → IP13Pro
SmartPhone

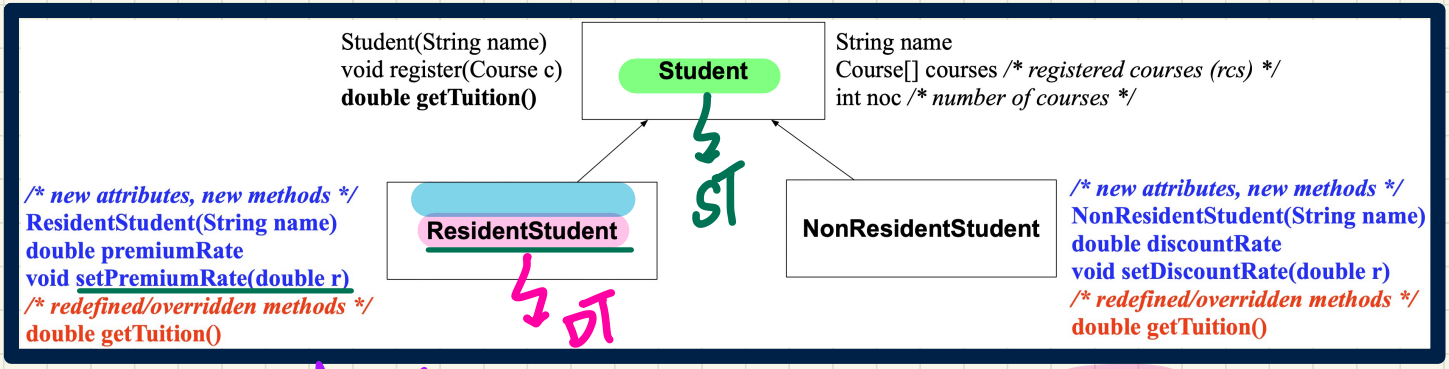
```
class SmartPhone {
  void dial() { ... }
}
class IOS extends SmartPhone {
  void facetime() { ... }
}
class IPhone13Pro extends IOS {
  void quickTake() { ... }
}
```

```
1  SmartPhone sp = new IPhone13Pro();          ✓
2  ((IPhone13Pro) sp).dial();                  ✓
3  ((IPhone13Pro) sp).facetime();              ✓
4  ((IPhone13Pro) sp).quickTake();             ✓
```

cast type

ref. var being cast

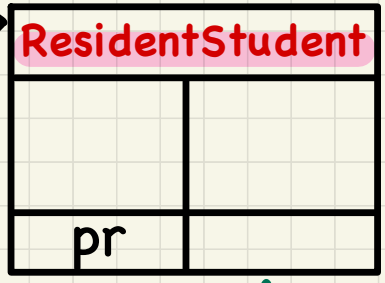
Static Types, Casts, Polymorphism (5)



True only if the DT of 's' can fulfill the expectation of RS.

(RS) s

Student s




```
Course eecs2030 = new Course("EECS2030", 500.0);
Student s = new ResidentStudent("Jim");
s.register(eecs2030);
if (s instanceof ResidentStudent) {
  ((ResidentStudent) s).setPremiumRate(1.75);
  System.out.println(((ResidentStudent) s).getTuition());
}
```

dynamic binding called.
⇒ version of RS

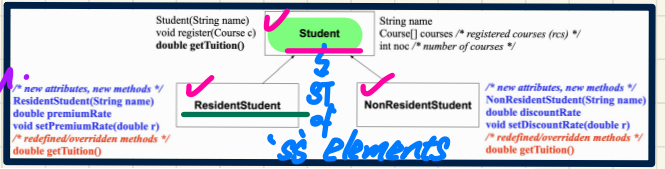
Anonymous most of ST (RS).

Lecture 5

Part M

Inheritance - Polymorphic Parameter Types

Polymorphic Parameters (1)



ST of param: ST of 'ss' elements

static type of elements of array 'ss'

```
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++; } }
```

parameter

Examples: Valid?

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

Student ⇒ DTs of elements can be descendants of Student

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

Valid: ST of rs is a descendant of ST of ss[c]

ST: Student

substitution valid?

ST: RS

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

sms.addRS(o)

C.O. of ST SMS

argument ST?

call by value: ST: RS ← rs = o

descendants of RS

ST?

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 s

ST: RS

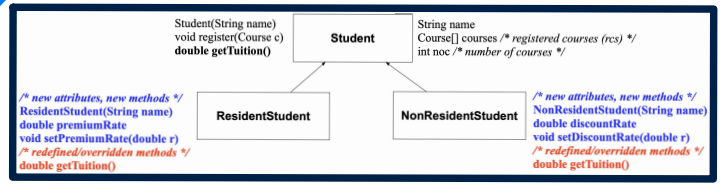
ST: Student

not valid

```
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);
```

call by value: s = nrs; → valid.

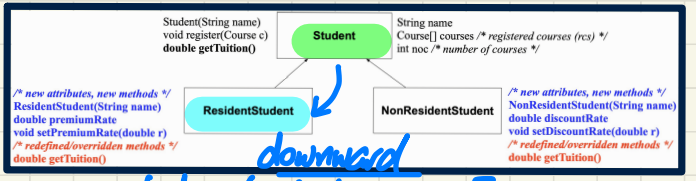
ST: Stud. ST: NRS



Casting Arguments

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

parameter



downward cast

sms.addRS((ResidentStudent) s) compiles? ① valid ② Runtime Exception

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

call by value: rs = s

ST: RS ST: Stu.

ClassCastException? YES.

only if DT of S is not a descendant of cast type

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

ClassCastException? YES.

DT NRS is not a descendant of cast type

Resid. Stud.

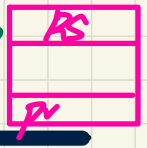
```
1 Student s = new ResidentStudent("Rachael");
2 /* s' ST: Student; s' DT: ResidentStudent */
3 StudentManagementSystem sms = new StudentManagementSystem();
4 sms.addRS(s); ×
```

No. DT RS can fulfil cast type.

ClassCastException?

RS

Student s

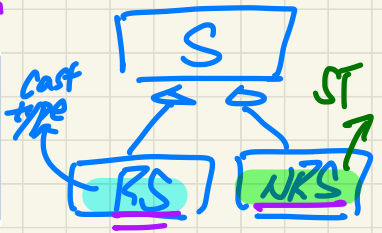


sms.addRS((ResidentStudent) nrs) compiles? No. RS is not cast type

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

ST: NRS

neither ancestor nor descendant of ST NRS.

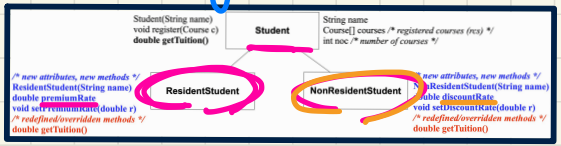


A Polymorphic Collection of Students

void addStudent(Student s){...} → call by value : s = rs

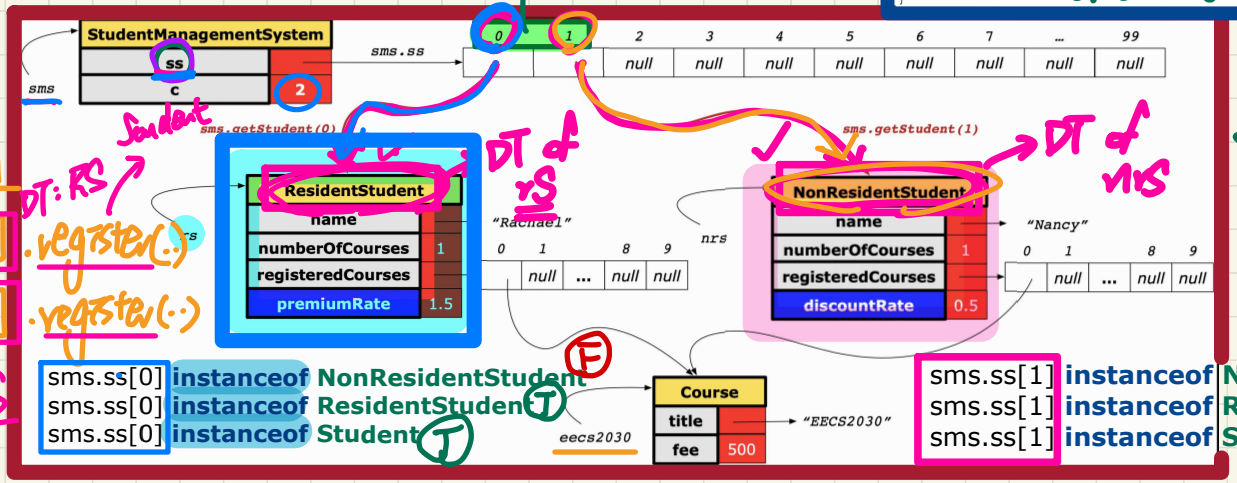
```
1  ResidentStudent rs = new ResidentStudent("Rachael");
2  rs.setPremiumRate(1.5);
3  NonResidentStudent nrs = new NonResidentStudent("Nancy");
4  nrs.setDiscountRate(0.5);
5  StudentManagementSystem sms = new StudentManagementSystem();
6  sms.addStudent(rs); /* polymorphism */
7  sms.addStudent(nrs);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

Iteration 1: sms.ss[0].getTuition()
Iteration 2: sms.ss[1].getTuition()
DT: RS
DT: NRS
ST: Student
ST: NRS
2




```
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);
    }
  }
}
```

Each element has ST Student.
RS version
ST
param
NRS version
2
compiler: Part of exp. of Student.
C.O. ST Student



Iteration
1. ss[0].register(..) DT: RS
2. ss[1].register(..)
DT: NRS
DT of rs
DT of nrs
T
F
T

Polymorphic collect.
ST of elements: Stu.
DT of de: descendants of ST

Lecture 5

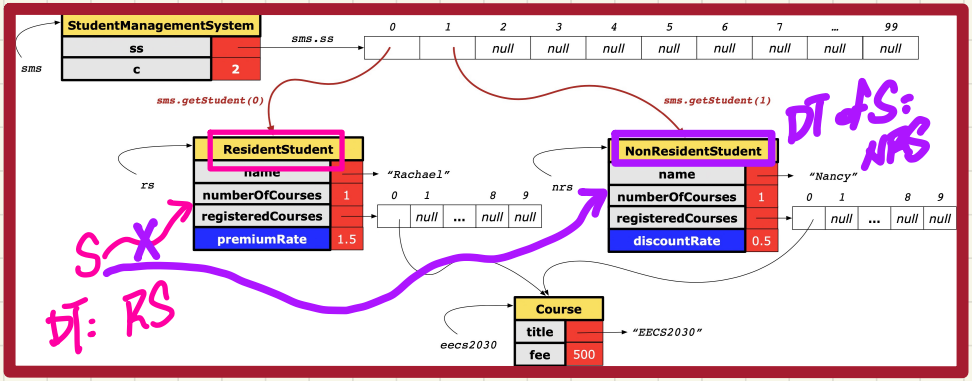
Part N

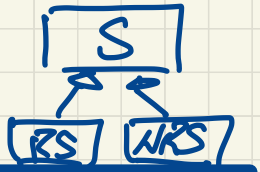
Inheritance -
Polymorphic Return Types

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("Invali
    }
    else {
      s = ss[i];
    }
    return s;
  }
}
```





valid? YES: RT can fulfill the exp. of ST of s

building polymorphic array

polymorphic collection

ST: RT of getStudent

DT: RS

ST: Student (RT)

ST: Student

ST: Student

valid?

① Can the ST of s fulfill the exp. of ST of getStudent's RV?

static type of return value from this accessor method

↳ dynamic type of the return value can be any descendant class of ST.

DT of s: NRS

S ×

DT: RS

Lecture 5

Part O

Inheritance - Overridden Methods and Dynamic Binding

Summary: Type Checking Rules

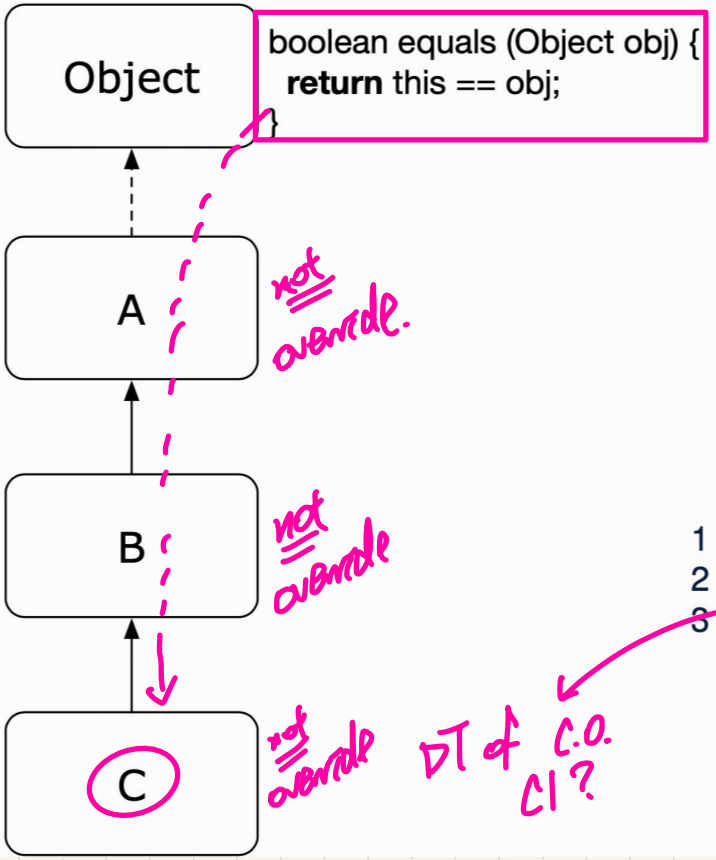
↳ compile time

Exercise: `z = x.m((C) y)` valid?

CODE	CONDITION TO BE TYPE CORRECT
`x = y`	Is `y`'s *ST* a **descendant** of `x`'s *ST*?
`x.m(y)`	Is method `m` defined in `x`'s *ST*? Is `y`'s *ST* a **descendant** of `m`'s parameter's *ST*?
`z = x.m(y)`	Is method `m` defined in `x`'s *ST*? Is `y`'s *ST* a **descendant** of `m`'s parameter's *ST*? Is *ST* of `m`'s return value a **descendant** of `z`'s *ST*?
`(C) y`	Is `C` an **ancestor** or a **descendant** of `y`'s *ST*?
`x = (C) y`	Is `C` an **ancestor** or a **descendant** of `y`'s *ST*? Is `C` a **descendant** of `x`'s *ST*?
`x.m((C) y)`	Is `C` an **ancestor** or a **descendant** of `y`'s *ST*? Is method `m` defined in `x`'s *ST*? Is `C` a **descendant** of `m`'s parameter's *ST*?

CCE at runtime if DT of y is not a descendant of C.

Overridden Methods and Dynamic Binding (1)

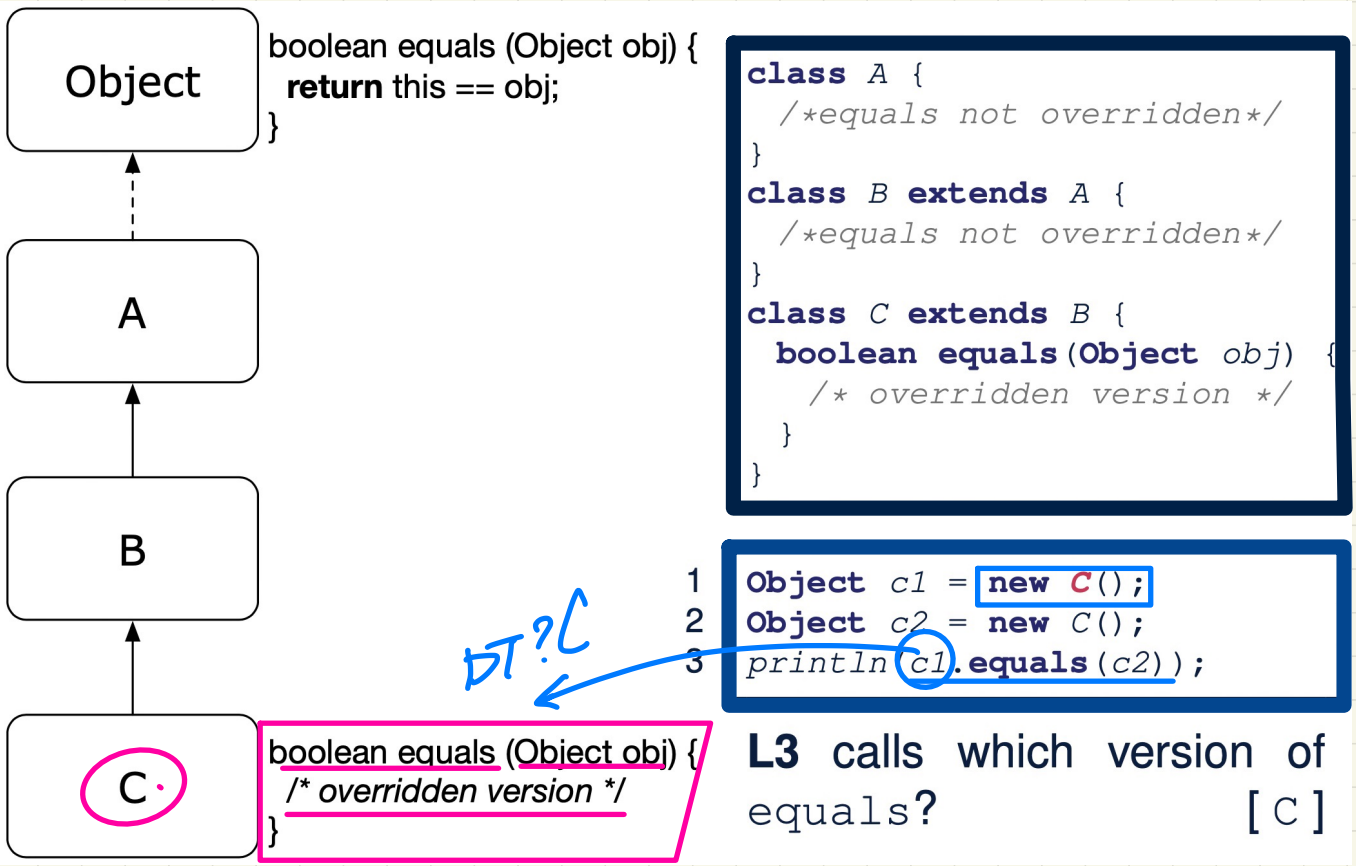



```
class A {
  /*equals not overridden*/
}
class B extends A {
  /*equals not overridden*/
}
class C extends B {
  /*equals not overridden*/
}
```

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

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

Overridden Methods and Dynamic Binding (2)



Overridden Methods and Dynamic Binding (3)