EECS2030 Advanced Object-Oriented Programming (Fall 2021)

Q&A - Lecture 5a

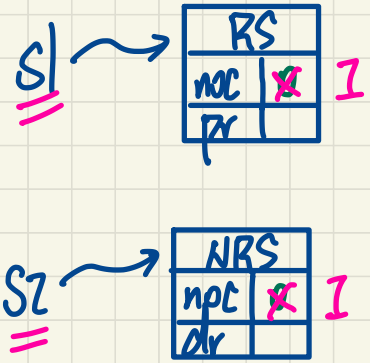
Wednesday, November 3

Announcement

1. type casts.
2. instanceof operator.

- Lecture W8 (released: Nov. 2)
- Programming Test 2 (due: Nov 4/Nov 5)
  - 1. Lab2 — exceptions
    Lab3 — equals, copy constructor
  - practice prog. test.

(Part D2; TimeStamp 16:07) If 2 different students register for the same course, wouldn't the attribute numberOfCourses be inaccurate, since the courses are the same but counted twice? If so, how can we avoid this?




```
class Student {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  Student (String name) {
    this.name = name;
    registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition() {
    double tuition = 0;
    for(int i = 0; i < numberOfCourses; i ++) {
      tuition += registeredCourses[i].fee;
    }
    return tuition; /* base amount only */
  }
}
```

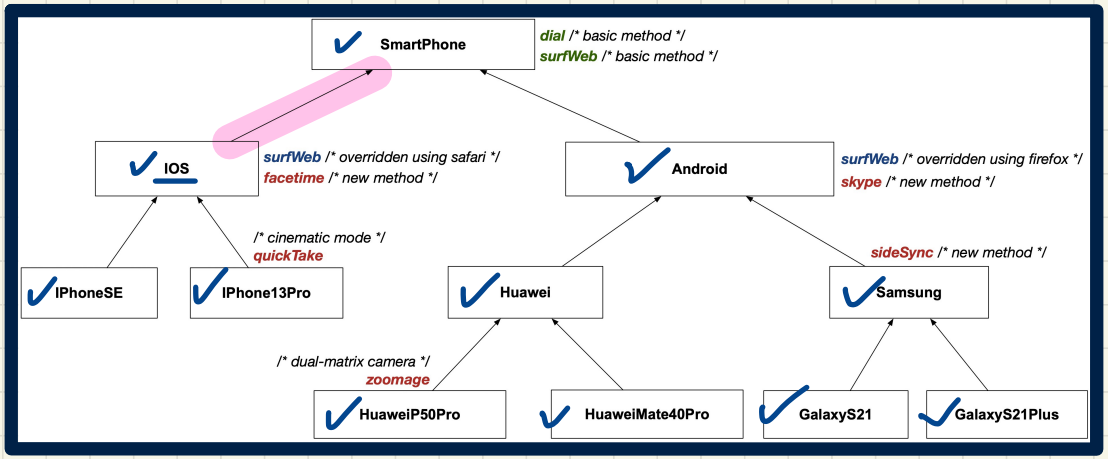
```
RS s1 = new RS("Jim");
NRS s2 = new NRS("Alan");
Course c = new Course("EECS2030", _);
s1.register(c);
s2.register(c);
print(s1.getTuition());
print(s2.getTuition());
```

```
class ResidentStudent extends Student {
  double premiumRate; /* there's a mutator meth
  ResidentStudent (String name) { super(name); }
  /* register method is inherited */
  double getTuition() {
    double base = super.getTuition();
    return base * premiumRate;
  }
}
```

```
class NonResidentStudent extends Student
  double discountRate; /* there's a mutator method
  NonResidentStudent (String name) { super(name); }
  /* register method is inherited */
  double getTuition() {
    double base = super.getTuition();
    return base * discountRate;
  }
}
```

EECS3311 .

When actually building something as complex as the smartphone diagram, assuming you've drawn it out first,
what would be the process of starting to code/implement everything?
Would you start at the top with SmartPhone and work your way down the chart? Or start at the buttom and work your way up?

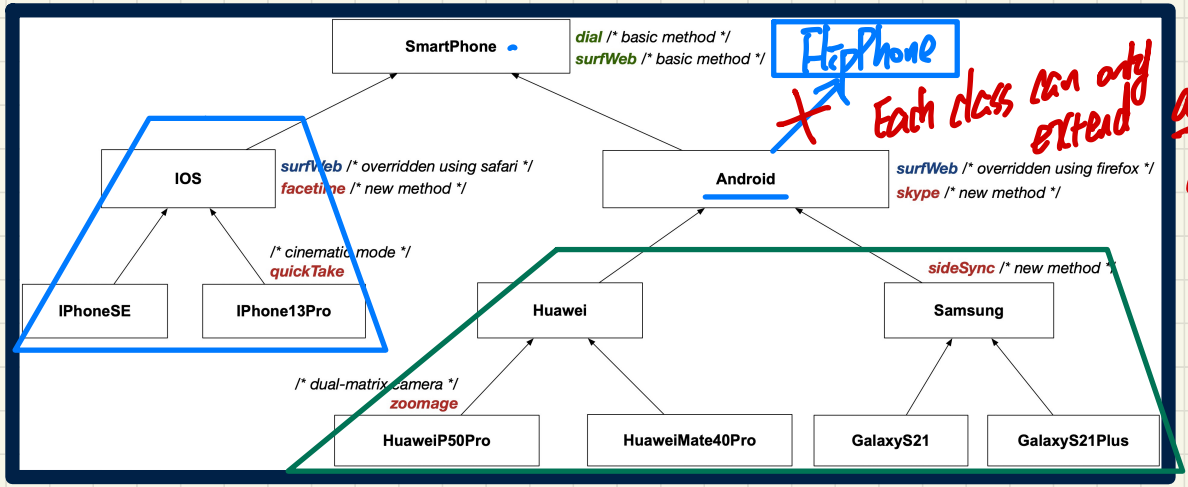


1. Create empty classes

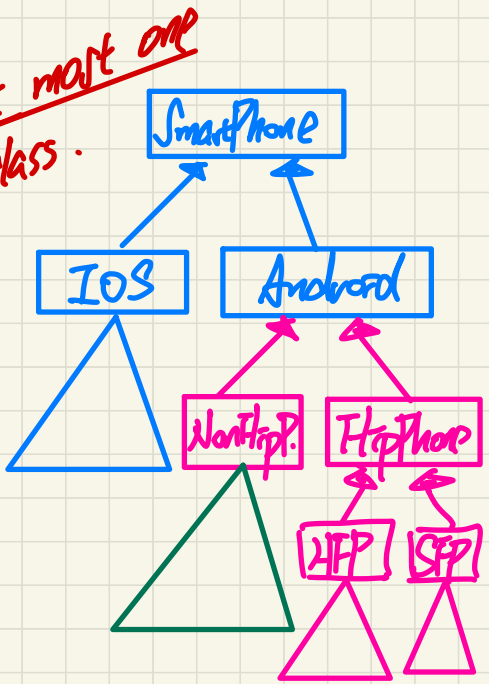
2. Implement the "extend" relations

e.g. public class IOS extends SmartPhone {
}

What if, for example there was another class flipPhone on the same level as SmartPhone.
And FlipPhone also runs android, would both smartPhone and flipPhone be ancestors to everything under Android? (Part G3; timestamp 2:00)



Each class can only extend at most one class.



- FlipPhone is still a SmartPhone.
- FlipPhone runs Android.
- FlipPhone manufactured by Samsung and/or Huawei?

As an extra example question to ensure my thinking is on the right track:
Wouldn't the getTuition method be considered as not single-choice principle because if you wanted to add something like tax to the price then you would have to change what it returns in both ResidentStudent and NonResidentStudent.
If the total price was set to be a variable then would it be single-choice because you can change whatever you need with that one variable?

≈ inflation rate

tax:

Case 1

→ Rates same for all students

Case 2

Rates different

base amount ←

```
class Student {
  String name;
  Course[] registeredCourses;
  int numberOfCourses;
  Student(String name) {
    this.name = name;
    registeredCourses = new Course[10];
  }
  void register(Course c) {
    registeredCourses[numberOfCourses] = c;
    numberOfCourses ++;
  }
  double getTuition() {
    double tuition = 0;
    for(int i = 0; i < numberOfCourses; i ++) {
      tuition += registeredCourses[i].fee;
    }
    return tuition; /* base amount only */
  }
}
```

* tax

```
class ResidentStudent extends Student {
  double premiumRate; /* there's a mutator meth
  ResidentStudent(String name) { super(name); }
  /* register method is inherited */
  double getTuition() {
    double base = super.getTuition();
    return base * premiumRate;
  }
}
```

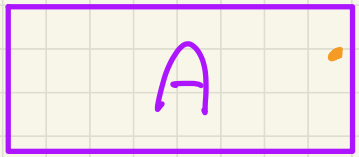
* tax1

```
class NonResidentStudent extends Student {
  double discountRate; /* there's a mutator method
  NonResidentStudent(String name) { super(name); }
  /* register method is inherited */
  double getTuition() {
    double base = super.getTuition();
    return base * discountRate;
  }
}
```

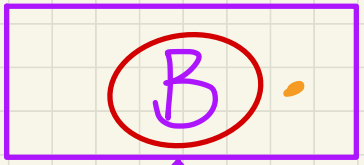
* tax2

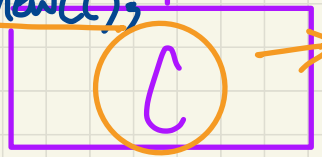
Overriding Grandparent Class Method

Case 1.



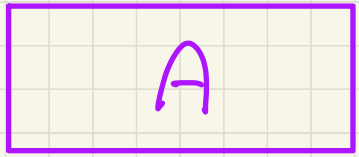

```
void m(){
    println("A.m");
}
```



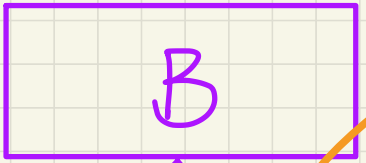



```
void m(){
    super.m();
    println("C.m");
}
```

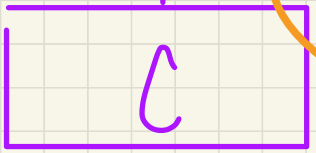
```
A obj1 = new B();
A obj2 = new C();
① obj1.m();
② obj2.m();
```




```
void m(){
    println("A.m");
}
```




```
void m(){
    println("B.m");
}
```




```
void m(){
    super.m();
    println("C.m");
}
```