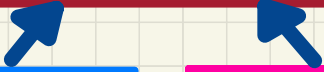


Student Classes (with inheritance)

```
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 */
    }
}
```



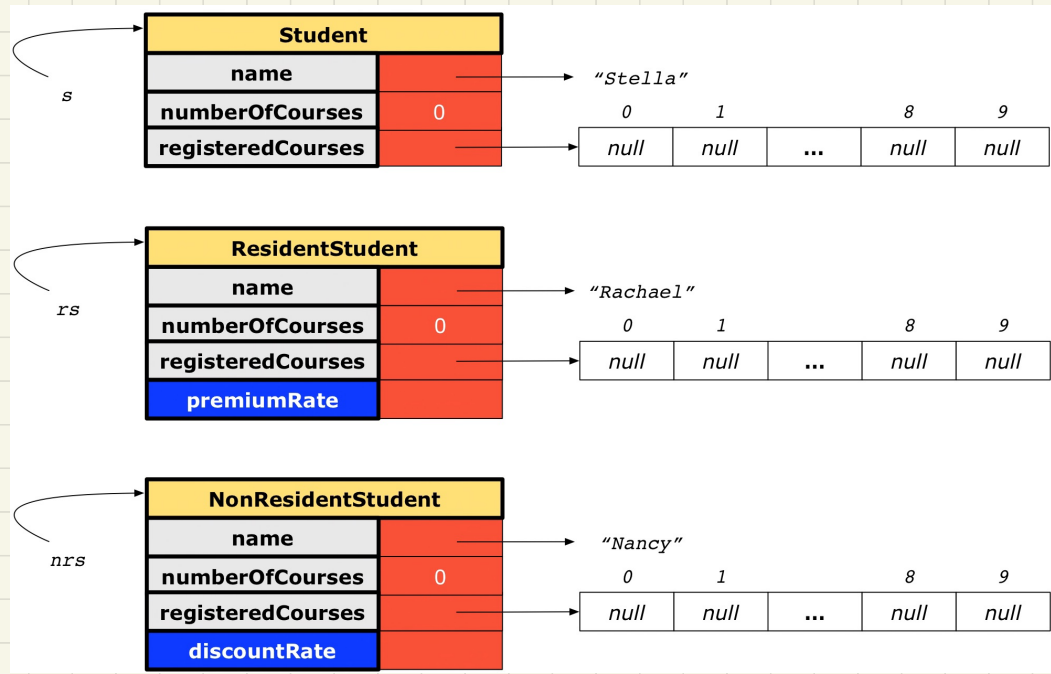
Two blue arrows point from the bottom of the Student class box to the top of the ResidentStudent and NonResidentStudent class boxes, indicating inheritance.

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

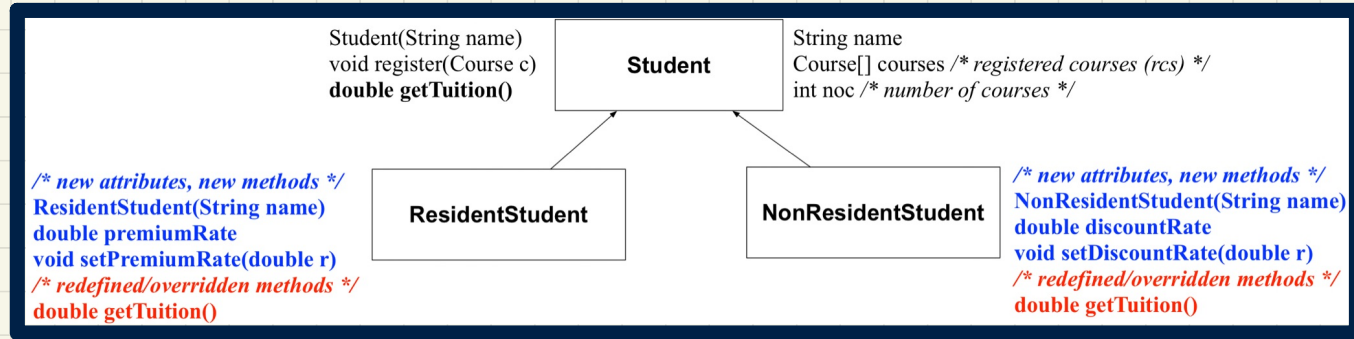
```
class NonResidentStudent extends Student {
    double discountRate; /* there's a mutator method */
    NonResidentStudent (String name) { super(name); }
    /* register method is inherited */
    double getTuition() {
        double base = super.getTuition();
        return base * discountRate;
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

Visualizing **Parent** and **Child** Objects

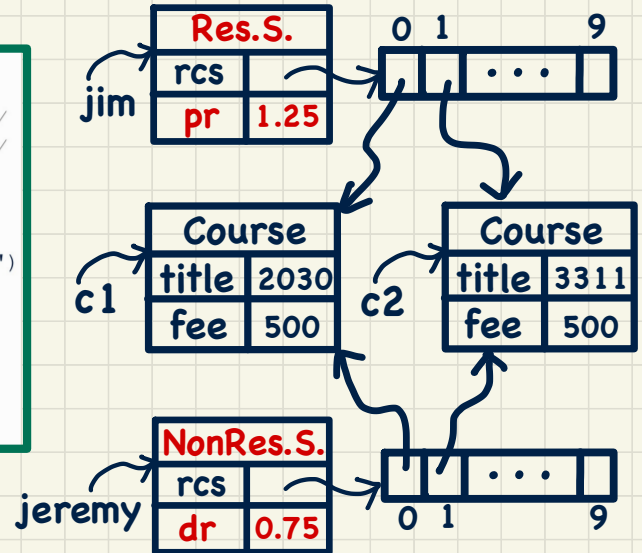
```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
```



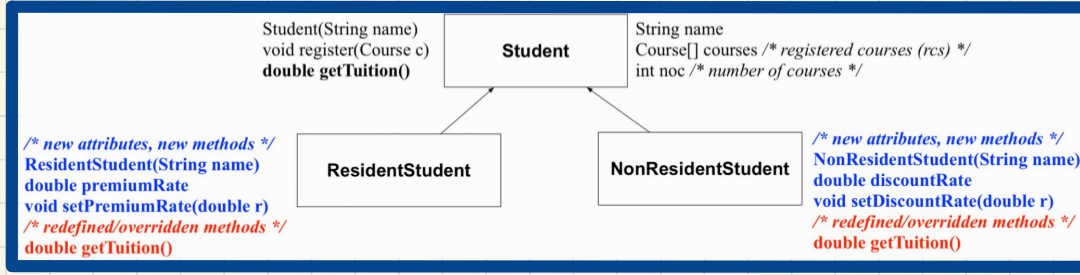
Testing Student Classes (with inheritance)



```
public class StudentTester {
    public static void main(String[] args) {
        Course c1 = new Course("EECS2030", 500.00); /* title and fee */
        Course c2 = new Course("EECS3311", 500.00); /* title and fee */
        ResidentStudent jim = new ResidentStudent("J. Davis");
        jim.setPremiumRate(1.25);
        jim.register(c1); jim.register(c2);
        NonResidentStudent jeremy = new NonResidentStudent("J. Gibbons");
        jeremy.setDiscountRate(0.75);
        jeremy.register(c1); jeremy.register(c2);
        System.out.println("Jim pays " + jim.getTuition());
        System.out.println("Jeremy pays " + jeremy.getTuition());
    }
}
```



Recall: Visualizing Parent and Child Objects

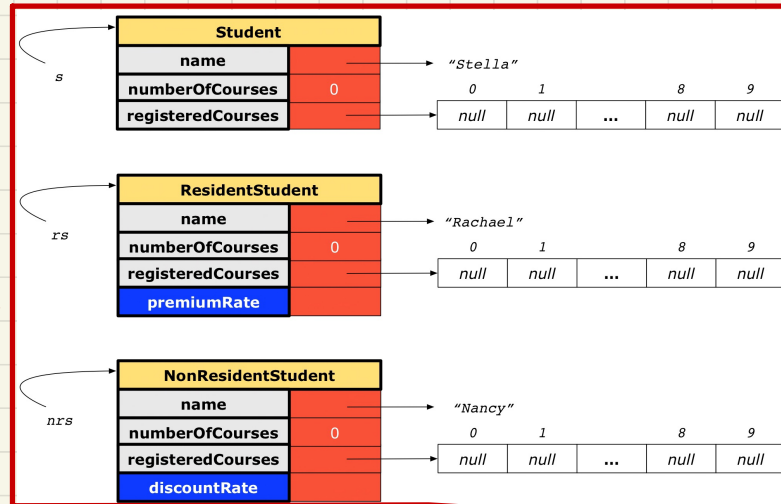


Inheritance Hierarchy

```
Student s = new Student("Stella");  
ResidentStudent rs = new ResidentStudent("Rachael");  
NonResidentStudent nrs = new NonResidentStudent("Nancy");
```

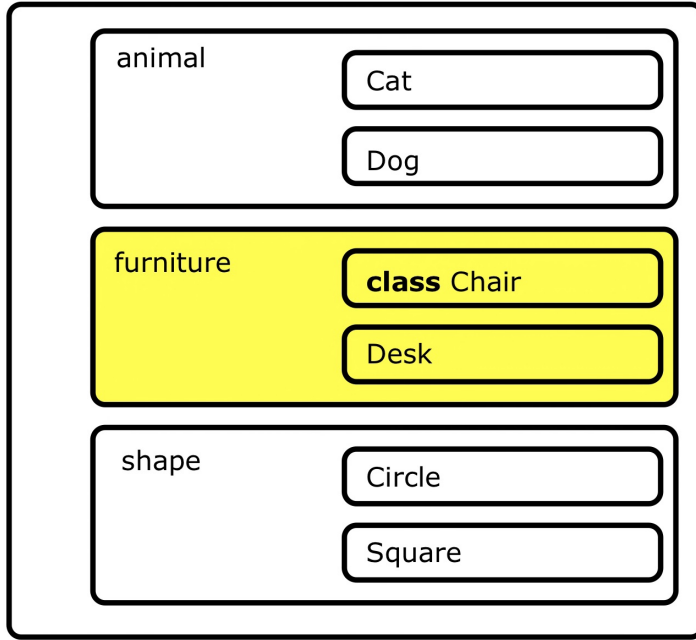
Declaring Static Types

Runtime Object Structure

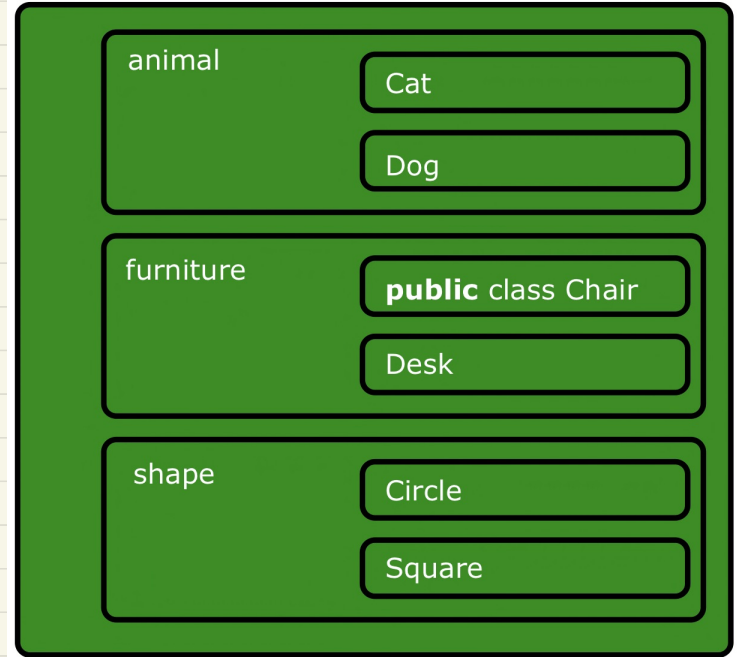


Visibility: Classes

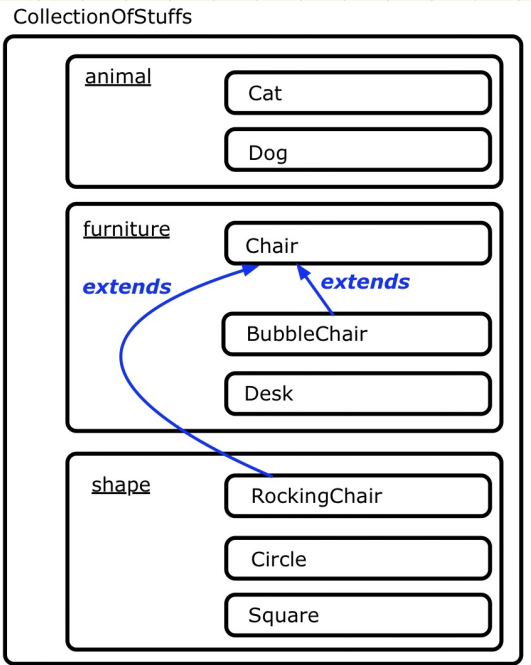
CollectionOfStuffs



CollectionOfStuffs



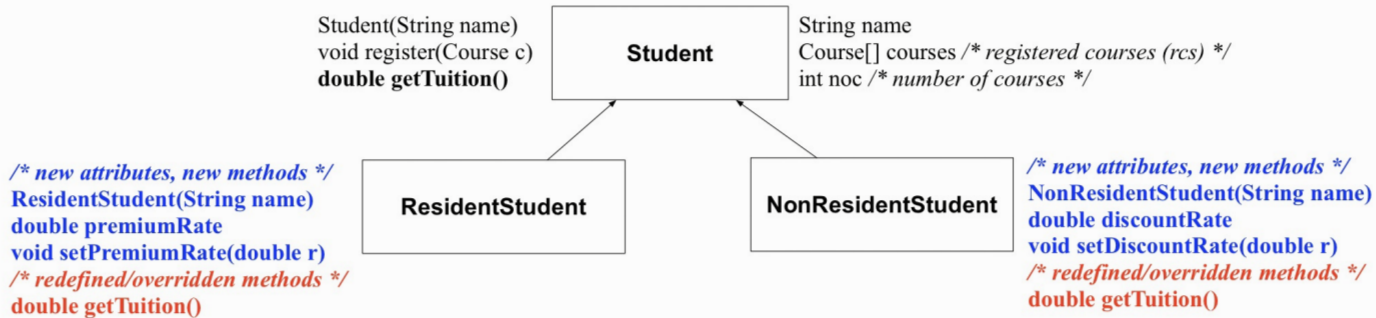
Visibility: Attributes and Methods



```
public class Chair {
    private int w;
    int x;
    protected int y;
    public int z;
}
```

	CLASS	PACKAGE	SUBCLASS (same pkg)	SUBCLASS (different pkg)	NON-SUBCLASS (across Project)
public	Green	Green	Green	Green	Green
protected	Green	Green	Green	Green	Red
no modifier	Green	Green	Green	Red	Red
private	Green	Red	Red	Red	Red

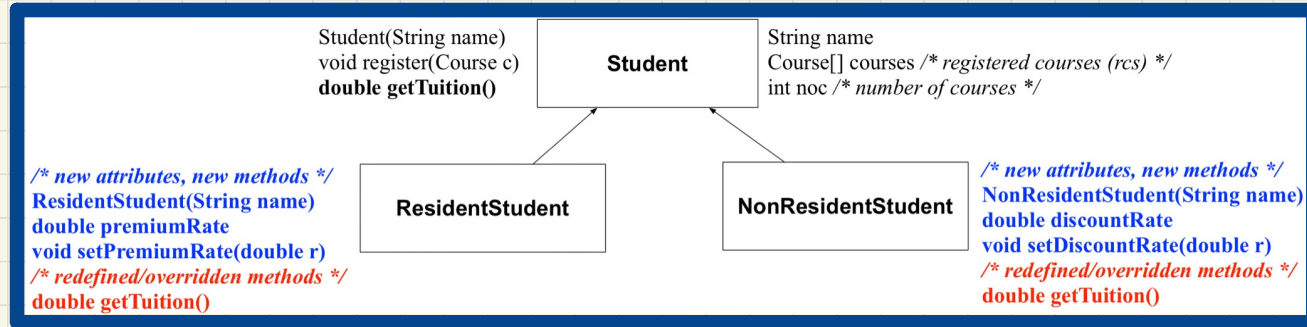
Student Classes (with inheritance): Expectations



```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
```

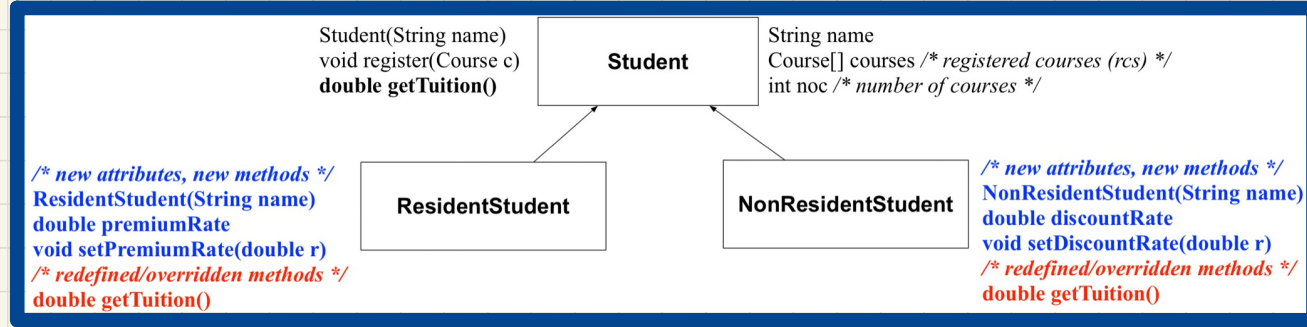
[illegible]

Intuition: Polymorphism

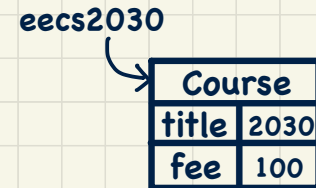
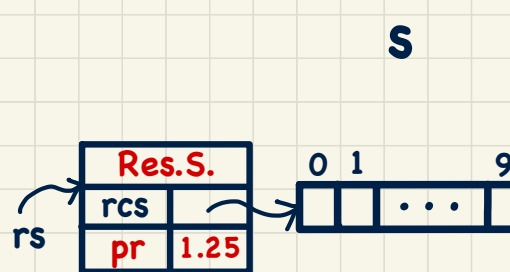


```
1 Student s = new Student("Stella");
2 ResidentStudent rs = new ResidentStudent("Rachael");
3 rs.setPremiumRate(1.25);
4 s = rs; /* Is this valid? */
5 rs = s; /* Is this valid? */
```

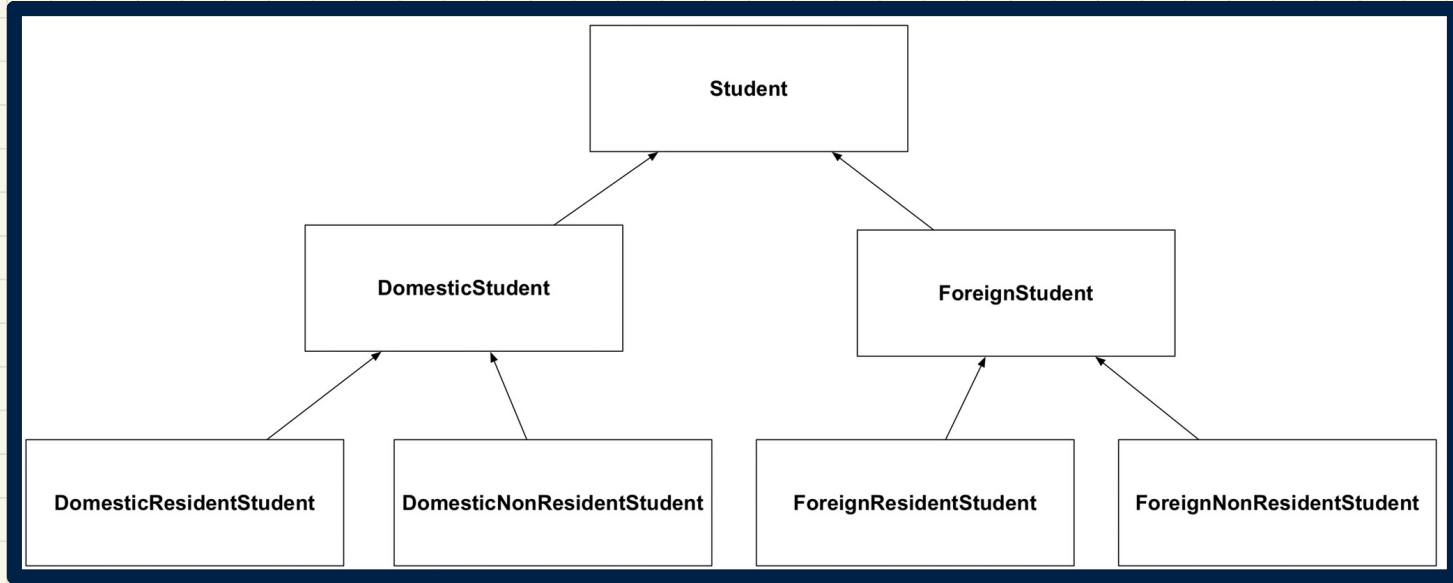
Intuition: Dynamic Binding



```
1 Course eecs2030 = new Course("EECS2030", 100.0);
2 Student s;
3 ResidentStudent rs = new ResidentStudent("Rachael");
4 NonResidentStudent nrs = new NonResidentStudent("Nancy");
5 rs.setPremiumRate(1.25); rs.register(eecs2030);
6 nrs.setDiscountRate(0.75); nrs.register(eecs2030);
7 s = rs; System.out.println(s.getTuition());
8 s = nrs; System.out.println(s.getTuition());
```



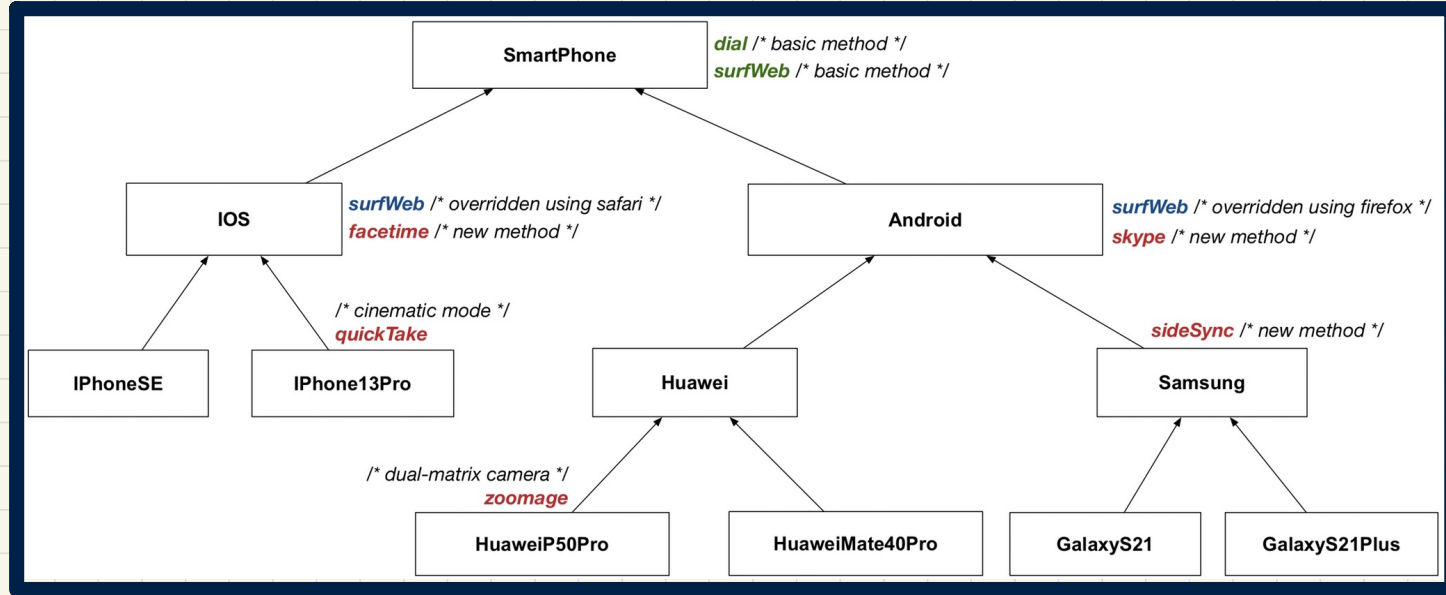
Multi-Level Inheritance Hierarchy: Students



Reflections:

- For [Design 1](#), how many encodings to check for each method?
- For [Design 2](#), how many arrays to store for SMS?
- For [Design 3](#), where are common attributes/methods stored?

Multi-Level Inheritance Hierarchy: Smartphones



Reflections:

- For Design 1, how many encodings to check for each method?
- For Design 2, how many arrays to store for SMS?
- For Design 3, where are common attributes/methods stored?