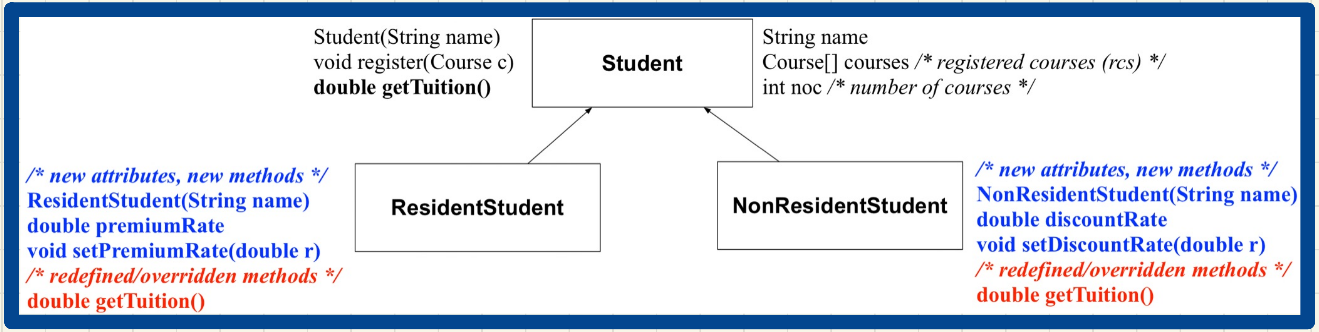
Intuition: Dynamic Binding

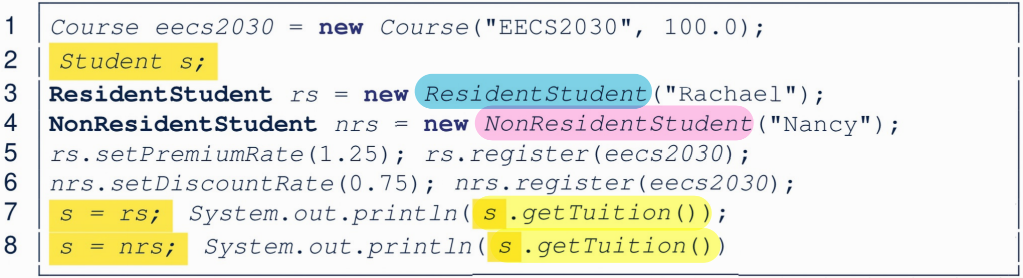


Student(String name)
void register(Course c)
double getTuition()

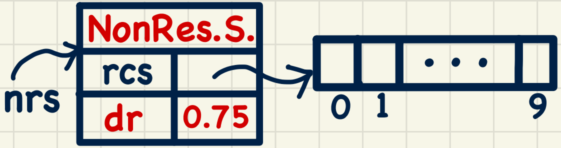
String name
Course[] courses /* *registered courses (rcs)* */
int noc /* *number of courses* */

/ new attributes, new methods */*
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/ redefined/overridden methods */*
double getTuition()

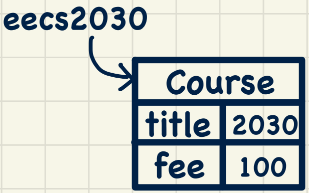
/ new attributes, new methods */*
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/ redefined/overridden methods */*
double getTuition()

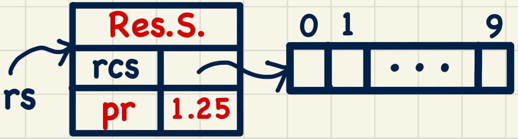


```
Course eecs2030 = new Course("EECS2030", 100.0);
Student s;
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
rs.setPremiumRate(1.25); rs.register(eecs2030);
nrs.setDiscountRate(0.75); nrs.register(eecs2030);
s = rs; System.out.println( s .getTuition());
s = nrs; System.out.println( s .getTuition())
```



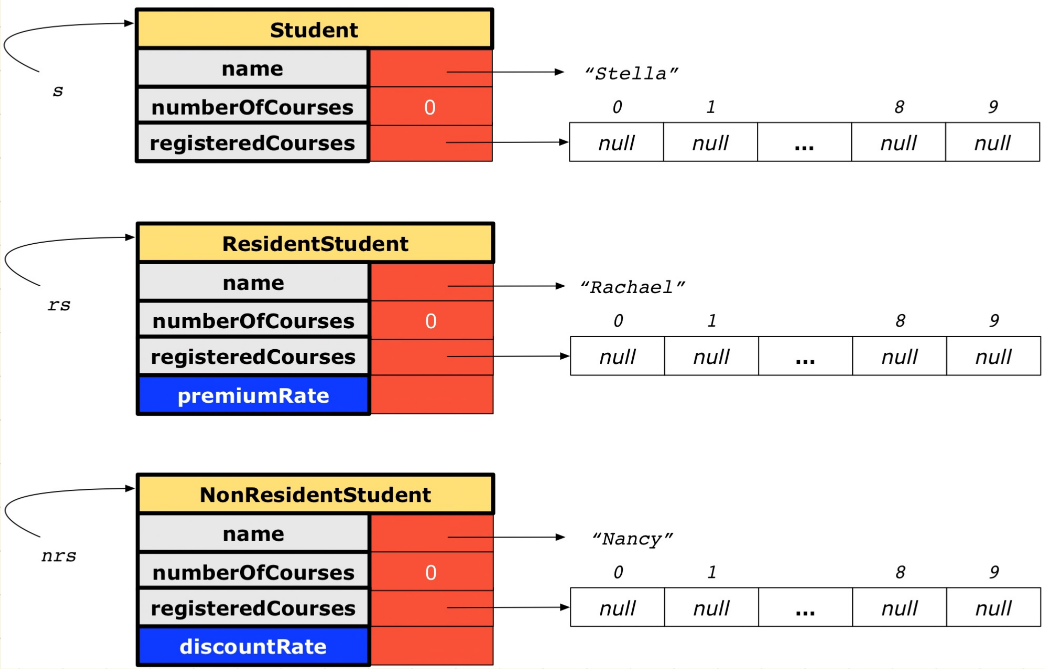
s



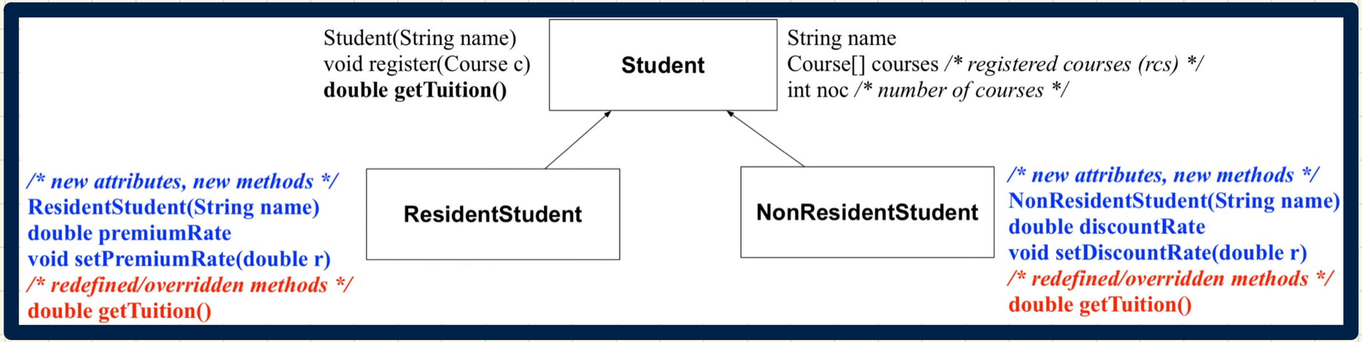


Visualizing Parent and Child Objects

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



Testing Student Classes (with inheritance)



Student(String name)
void register(Course c)
double getTuition()

Student

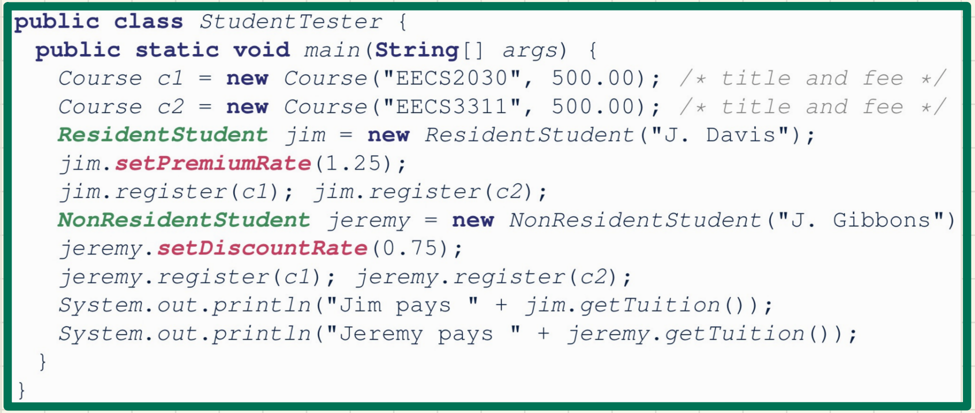
String name
Course[] courses /* *registered courses (rcs)* */
int noc /* *number of courses* */

/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()

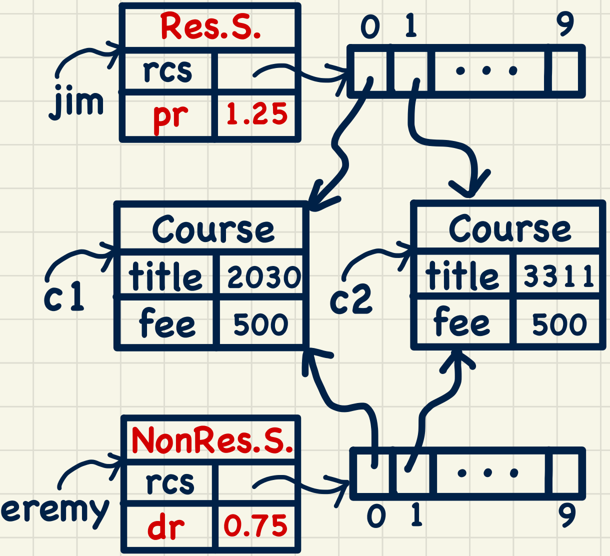
ResidentStudent

NonResidentStudent

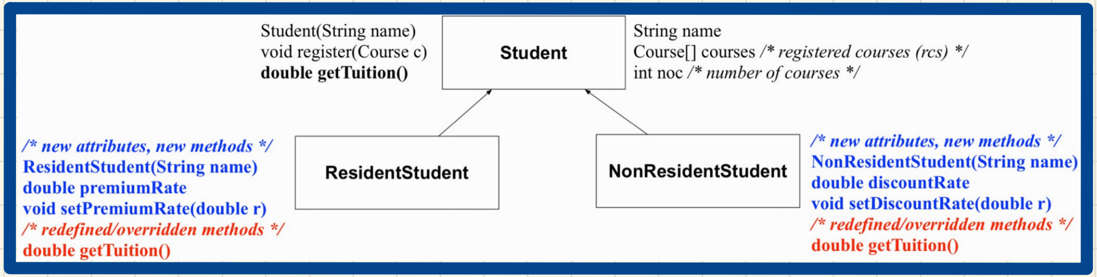
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()



```
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



Student(String name)
void register(Course c)
double getTuition()

Student

String name
Course[] courses /* *registered courses (rcs)* */
int noc /* *number of courses* */

/ new attributes, new methods */*
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/ redefined/overridden methods */*
double getTuition()

ResidentStudent

NonResidentStudent

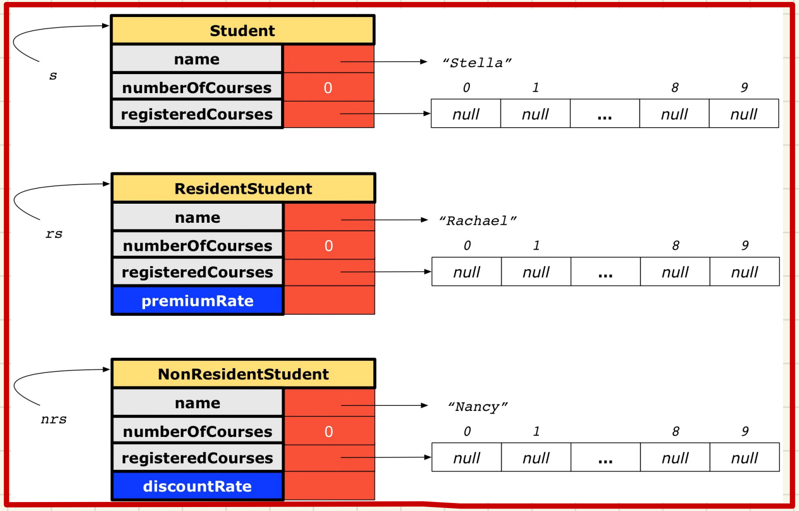
/ new attributes, new methods */*
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/ redefined/overridden methods */*
double getTuition()

Inheritance
Hirarchy

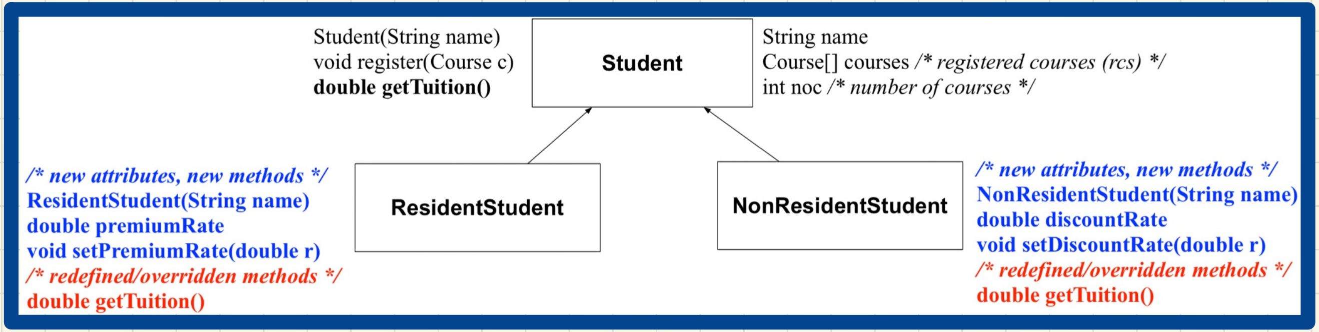
```
Student s = new Student("Stella");
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
```

Declaring
Static Types

Runtime
Object
Structure



Intuition: Dynamic Binding



Student(String name)
void register(Course c)
double getTuition()

Student

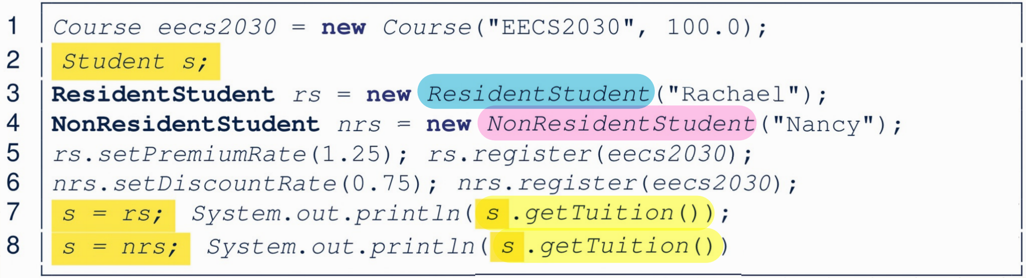
String name
Course[] courses */* registered courses (rcs) */*
int noc */* number of courses */*

/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()

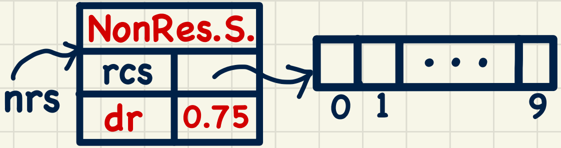
ResidentStudent

NonResidentStudent

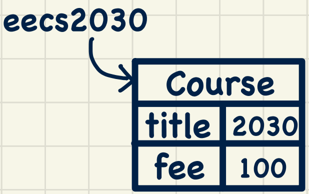
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

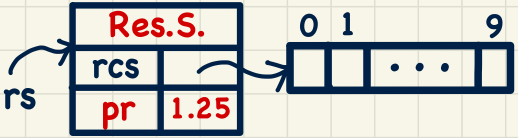


```
Course eecs2030 = new Course("EECS2030", 100.0);
Student s;
ResidentStudent rs = new ResidentStudent("Rachael");
NonResidentStudent nrs = new NonResidentStudent("Nancy");
rs.setPremiumRate(1.25); rs.register(eecs2030);
nrs.setDiscountRate(0.75); nrs.register(eecs2030);
s = rs; System.out.println( s .getTuition());
s = nrs; System.out.println( s .getTuition())
```

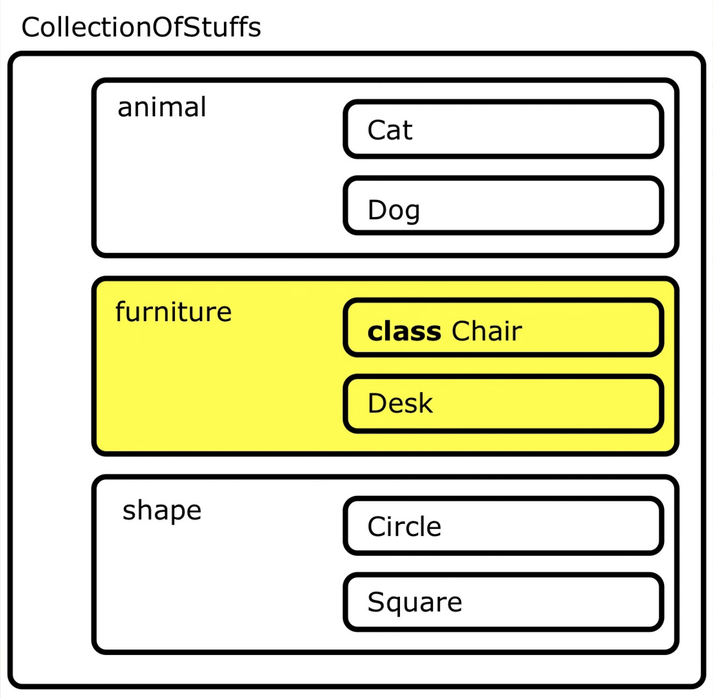


s





Visibility: Classes



CollectionOfStuffs

animal

Cat

Dog

furniture

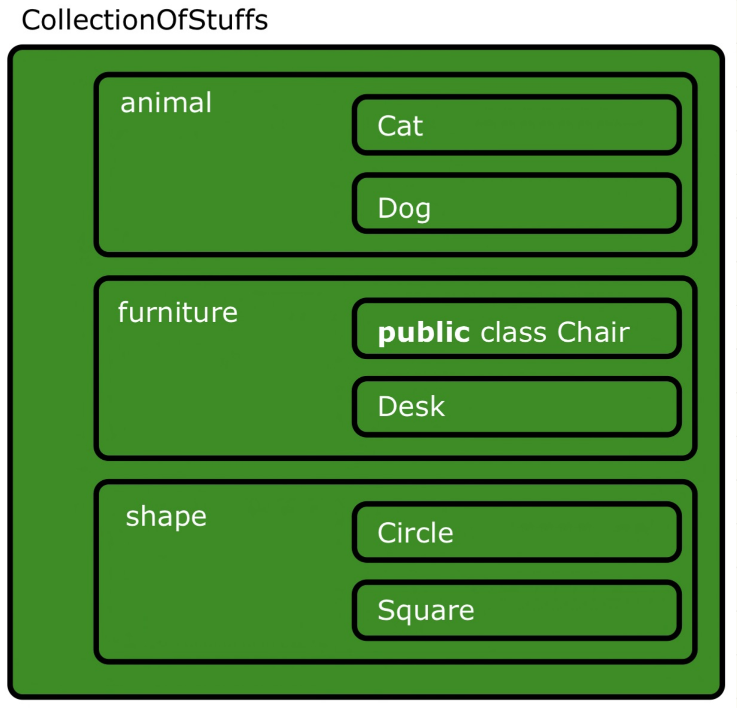
class Chair

Desk

shape

Circle

Square



CollectionOfStuffs

animal

Cat

Dog

furniture

public class Chair

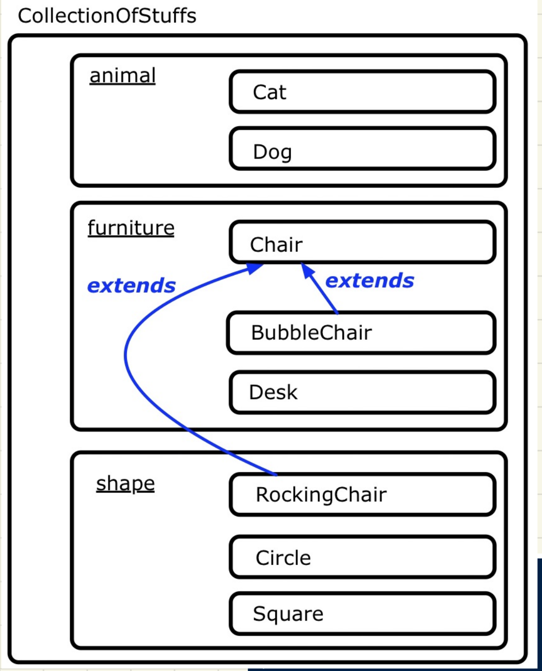
Desk

shape

Circle

Square

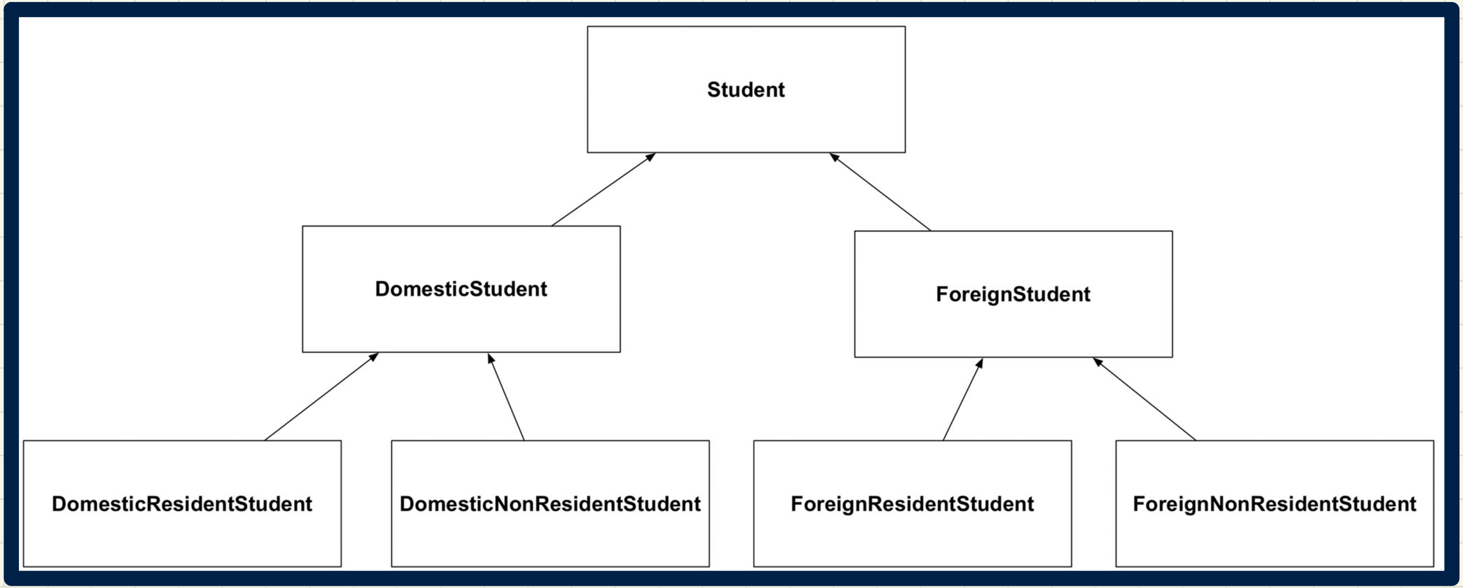
Visibility: Attributes and Methods



```
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	🟩	🟩	🟩	🟩	🟩
protected	🟩	🟩	🟩	🟩	🟥
no modifier	🟩	🟩	🟩	🟥	🟥
private	🟩	🟥	🟥	🟥	🟥

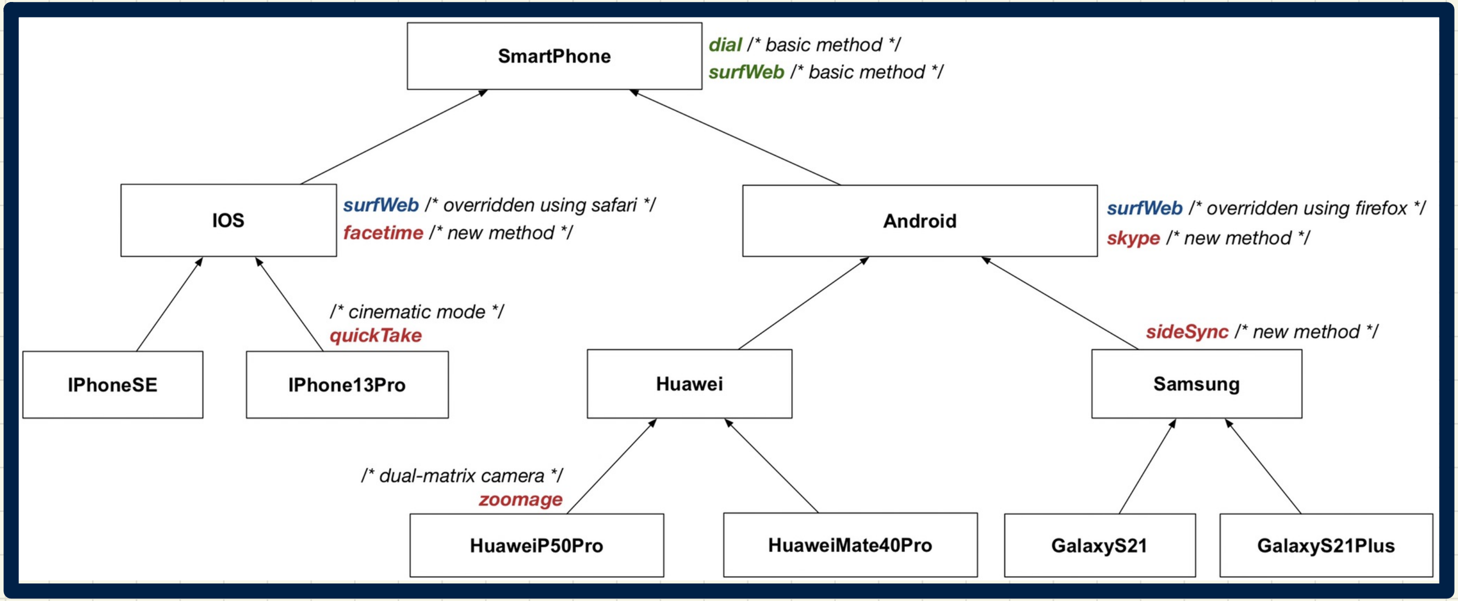
Multi-Level Inheritance Hierarchy: Students



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?

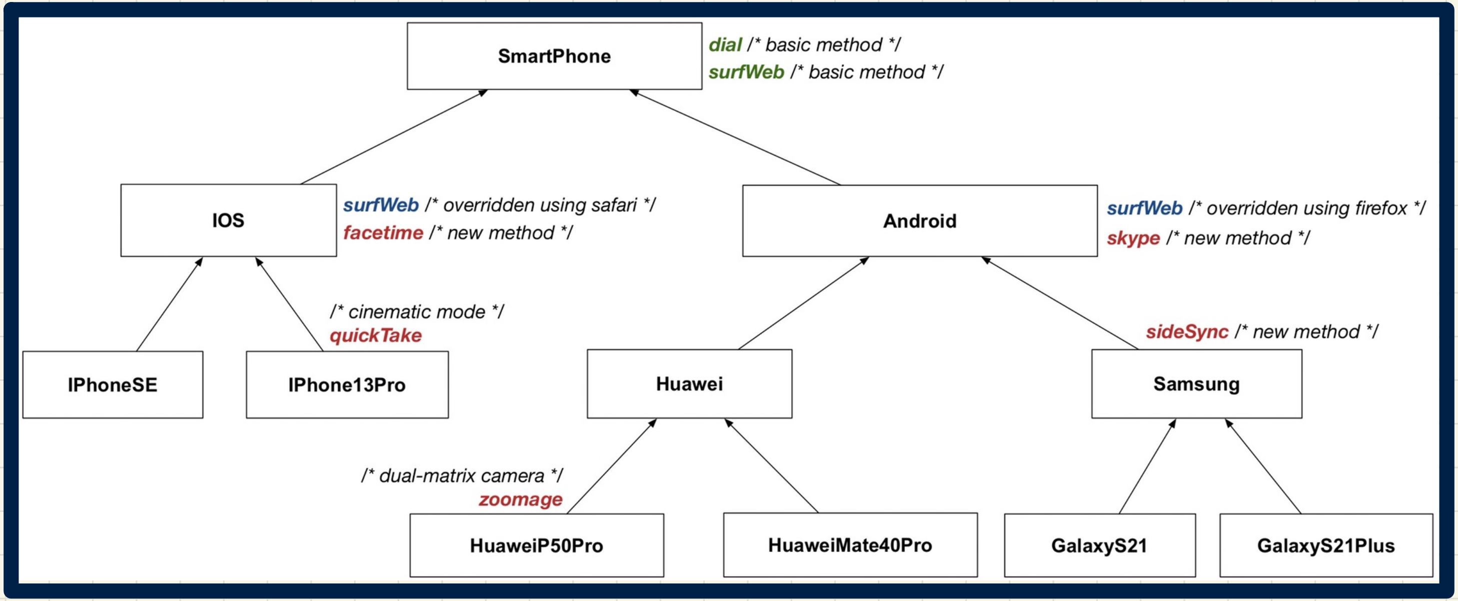
Multi-Level Inheritance Hierarchy: Smartphones



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?

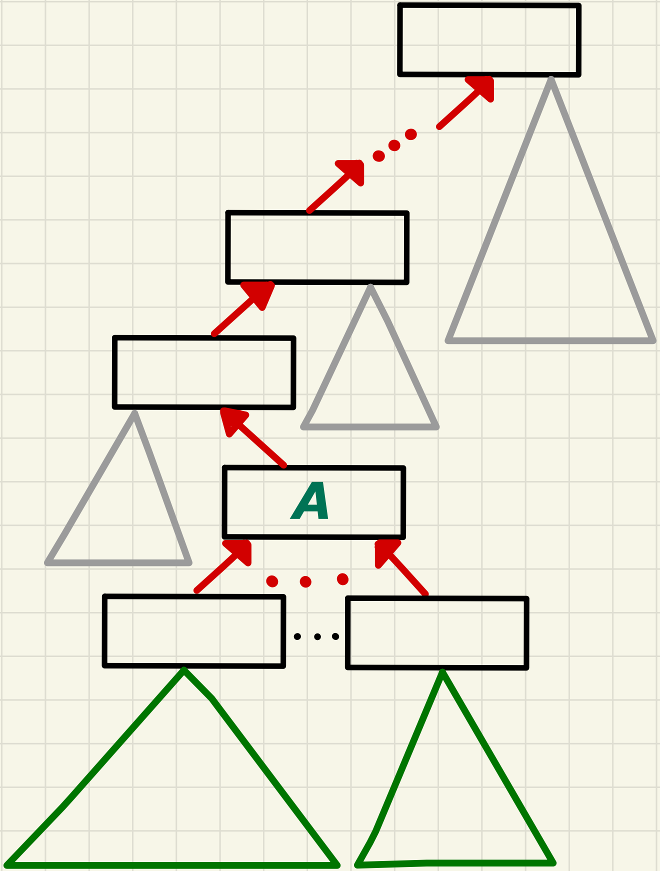
Multi-Level Inheritance Hierarchy: Smartphones



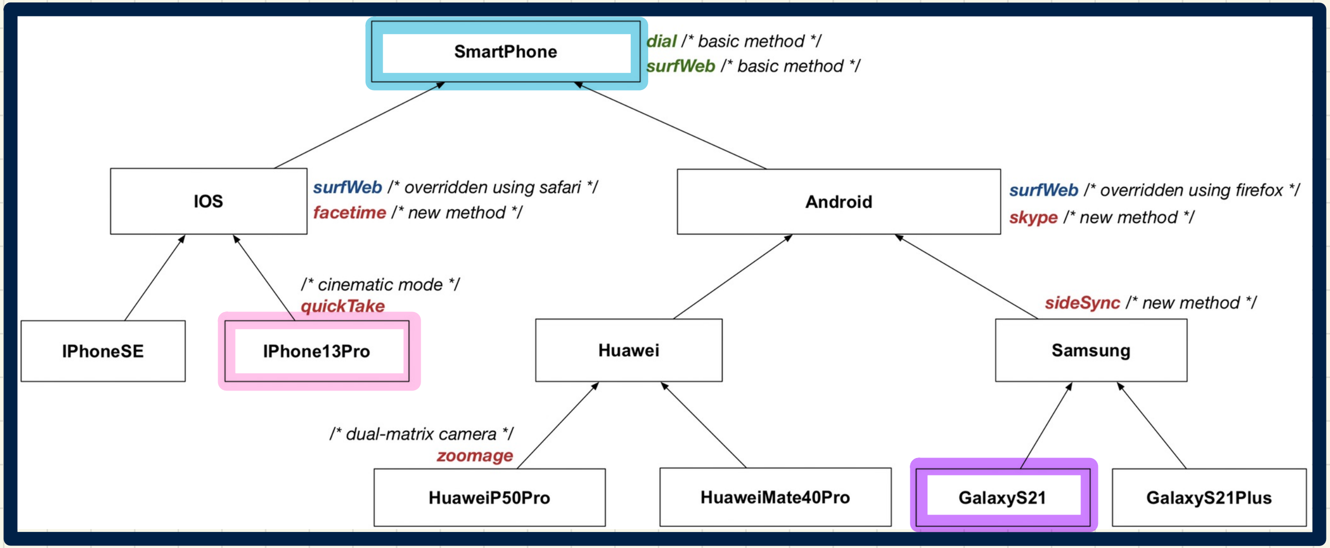
Exercise Compare the ranges of expectations of:

+ IPhone13Pro
+ HuaweiP50Pro
+ GalaxyS21Plus

Inheritance Forms a Type Hierarchy

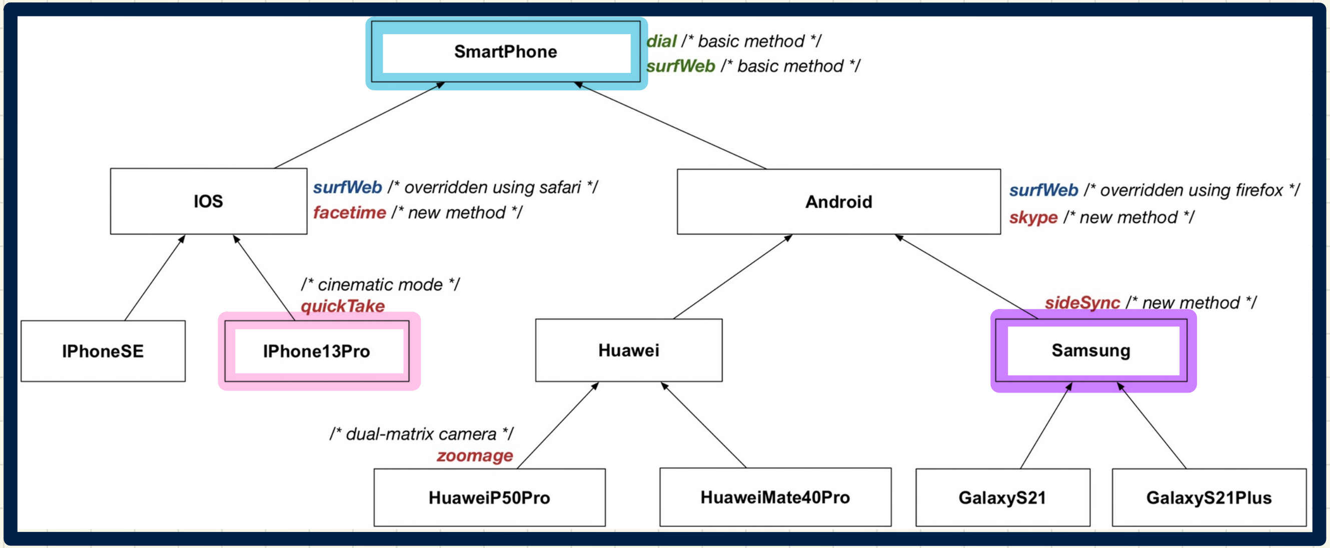


Inheritance Accumulates Code for Reuse



	ancestors	expectations	descendants
🟪			
▨			
🟦			

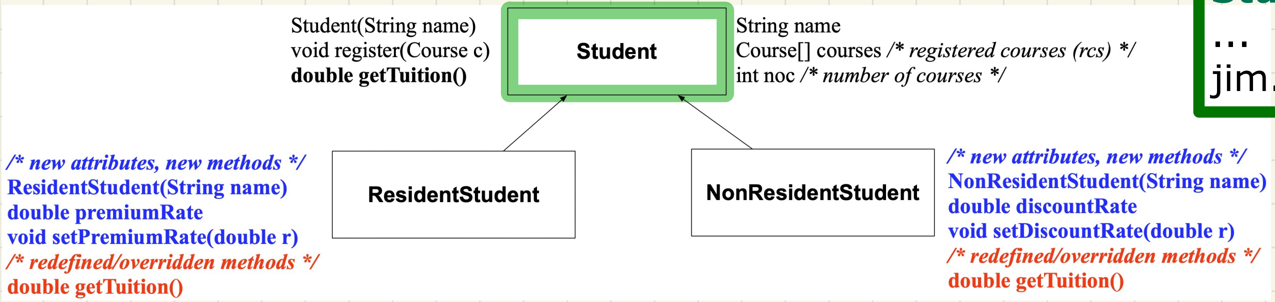
Inheritance Accumulates Code for Reuse



```
SmartPhone sp1;          sp1 = ?;
IPhone13Pro sp2;         sp2 = ?;
Samsung sp3;             sp3 = ?;
```

Static Types determine Expectations

Inheritance Herarchy: Students



Student

Student(String name)
void register(Course c)
double getTuition()

String name
Course[] courses /* *registered courses (rcs)* */
int noc /* *number of courses* */

ResidentStudent

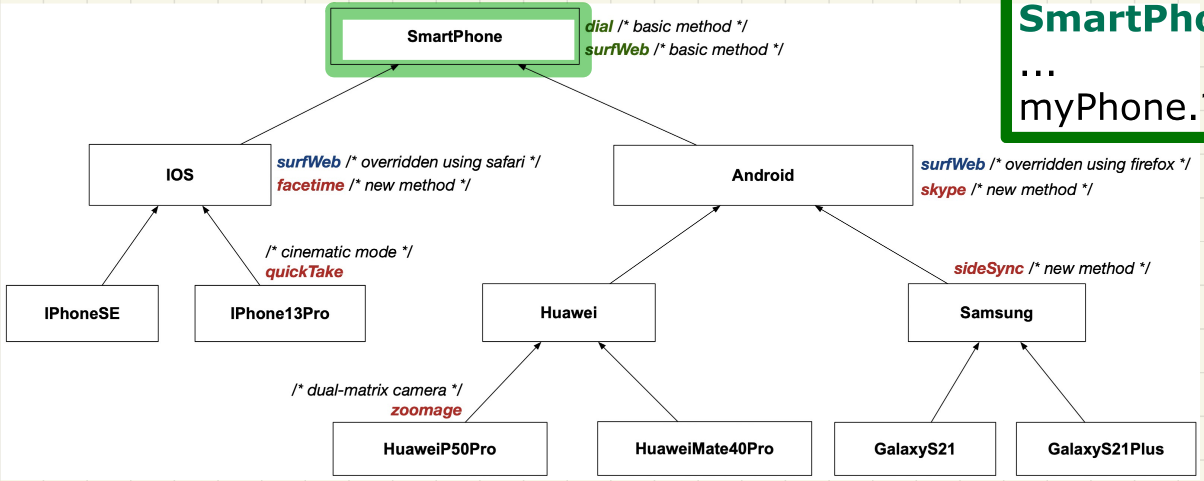
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()

NonResidentStudent

/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

Declare:
Student jim;
...
jim.??

Inheritance Herarchy: Smart Phones



SmartPhone

dial */* basic method */*
surfWeb */* basic method */*

IOS

surfWeb */* overridden using safari */*
facetime */* new method */*

Android

surfWeb */* overridden using firefox */*
skype */* new method */*

IPhoneSE

IPhone13Pro

/ cinematic mode */*
quickTake

Huawei

Samsung

sideSync */* new method */*

HuaweiP50Pro

/ dual-matrix camera */*
zoomage

HuaweiMate40Pro

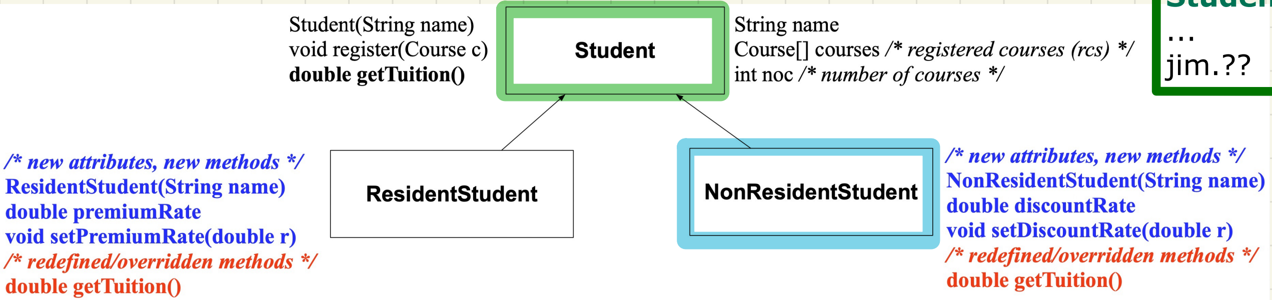
GalaxyS21

GalaxyS21Plus

Declare:
SmartPhone myPhone;
...
myPhone.??

Static Types determine Expectations

Inheritance Herarchy: Students



Student(String name)
void register(Course c)
double getTuition()

Student

String name
Course[] courses /* *registered courses (rcs)* */
int noc /* *number of courses* */

ResidentStudent

/ new attributes, new methods */*
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/ redefined/overridden methods */*
double getTuition()

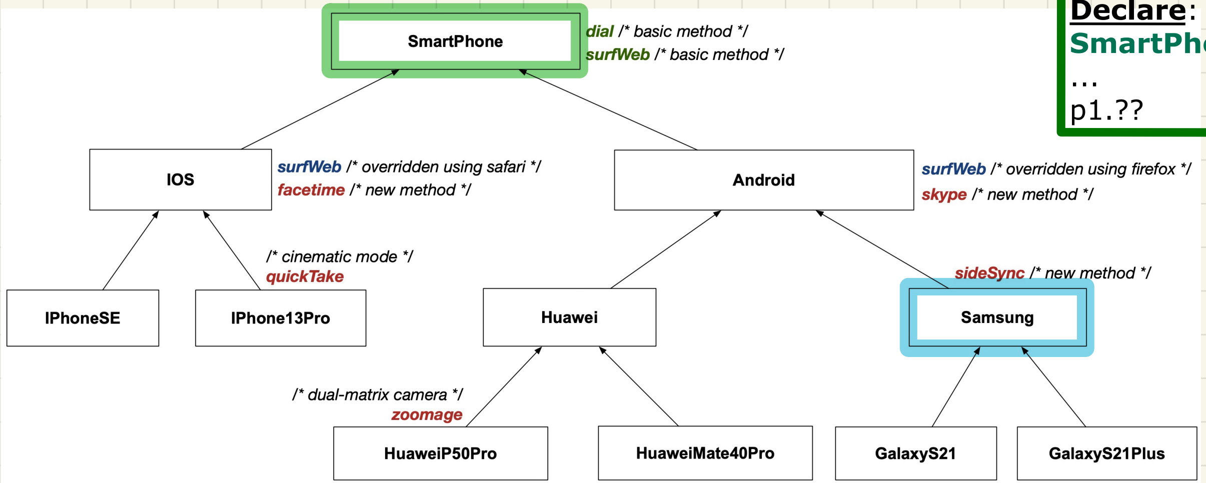
NonResidentStudent

/ new attributes, new methods */*
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/ redefined/overridden methods */*
double getTuition()

Declare:
Student jim;
...
jim.??

Declare:
NRS alan;
...
alan.??

Inheritance Herarchy: Smart Phones



SmartPhone

dial / basic method */*
surfWeb / basic method */*

IOS

surfWeb / overridden using safari */*
facetime / new method */*

Android

surfWeb / overridden using firefox */*
skype / new method */*

IPhoneSE

/ cinematic mode */*
quickTake

IPhone13Pro

Huawei

sideSync / new method */*

Samsung

/ dual-matrix camera */*
zoomage

HuaweiP50Pro

HuaweiMate40Pro

GalaxyS21

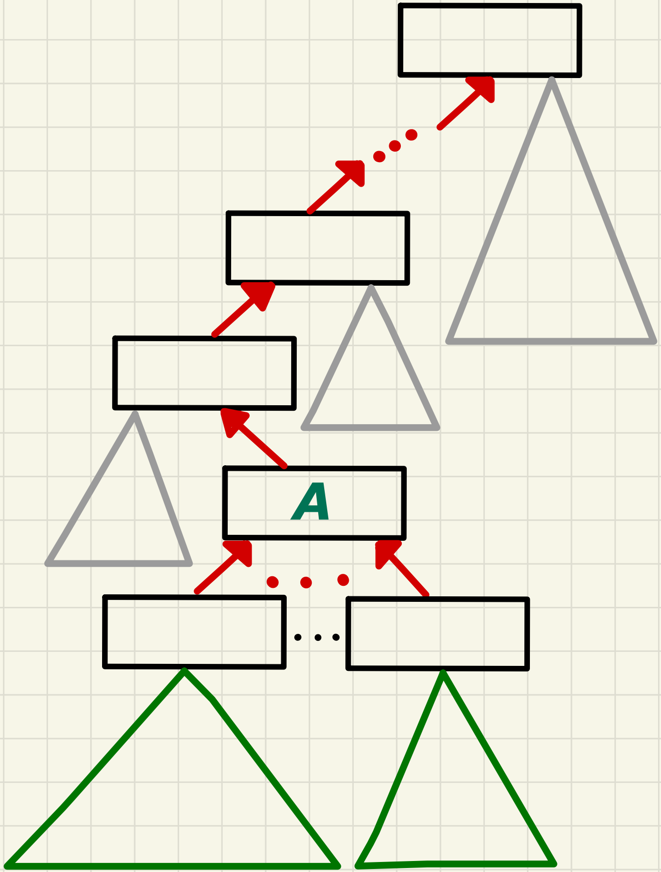
GalaxyS21Plus

Declare:
SmartPhone p1;
...
p1.??

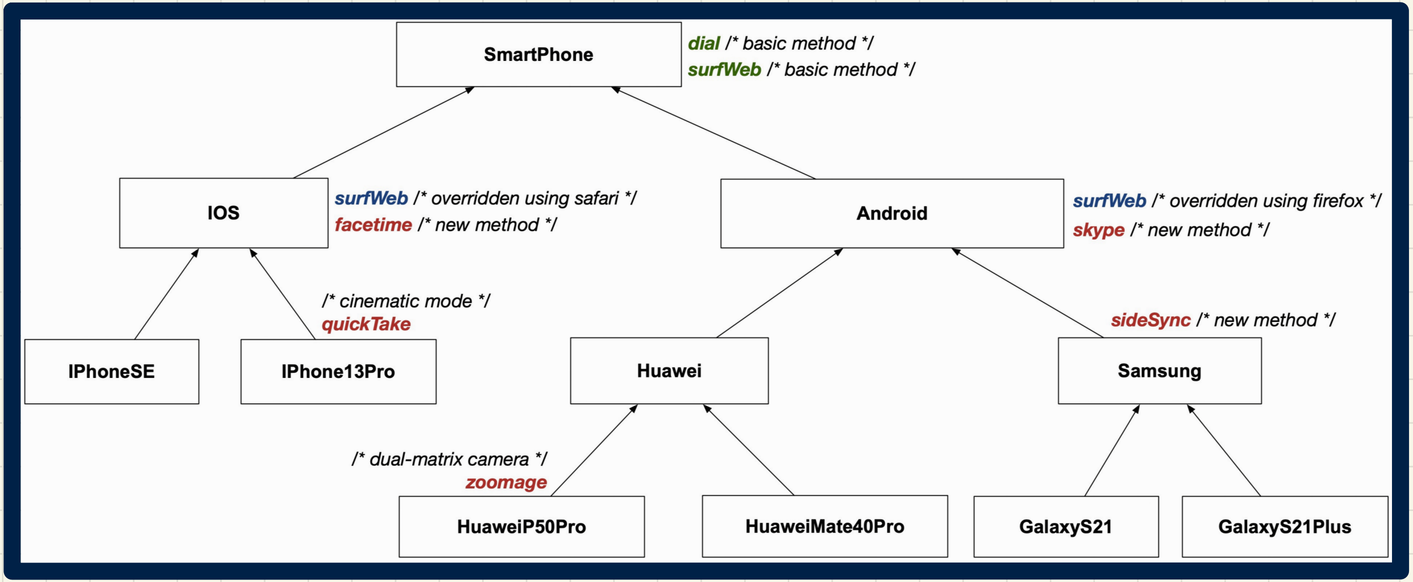
Declare:
Samsung p2;
...
p2.??

Rules of Substitutions

```
A oa = …;
? ob = …;
oa = ob;
```



Rules of Substitutions (1)



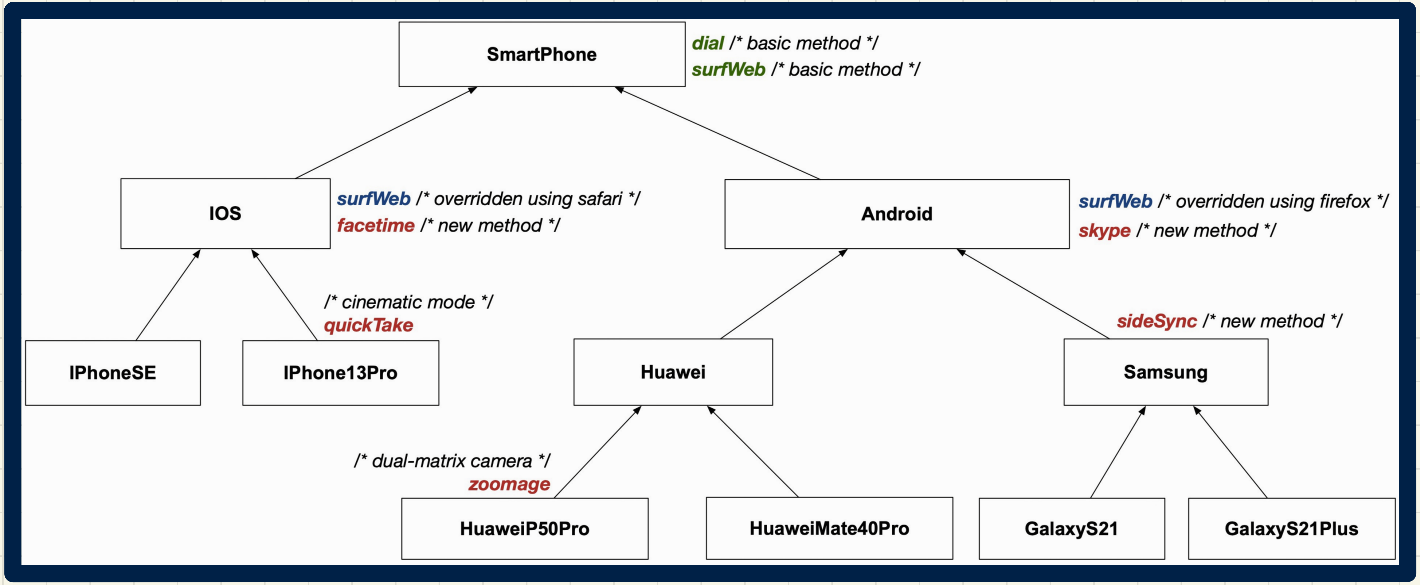
Declarations:

```
IOS sp1;
IPhoneSE sp2;
IPhone13Pro sp3;
```

Substitutions:

```
sp1 = sp2;
sp1 = sp3;
```

Rules of Substitutions (2)



SmartPhone

dial /* basic method */

surfWeb /* basic method */

IOS

surfWeb /* overridden using safari */

facetime /* new method */

Android

surfWeb /* overridden using firefox */

skype /* new method */

IPhoneSE

IPhone13Pro

/* cinematic mode */

quickTake

Huawei

Samsung

sideSync /* new method */

HuaweiP50Pro

/* dual-matrix camera */

zoomage

HuaweiMate40Pro

GalaxyS21

GalaxyS21Plus

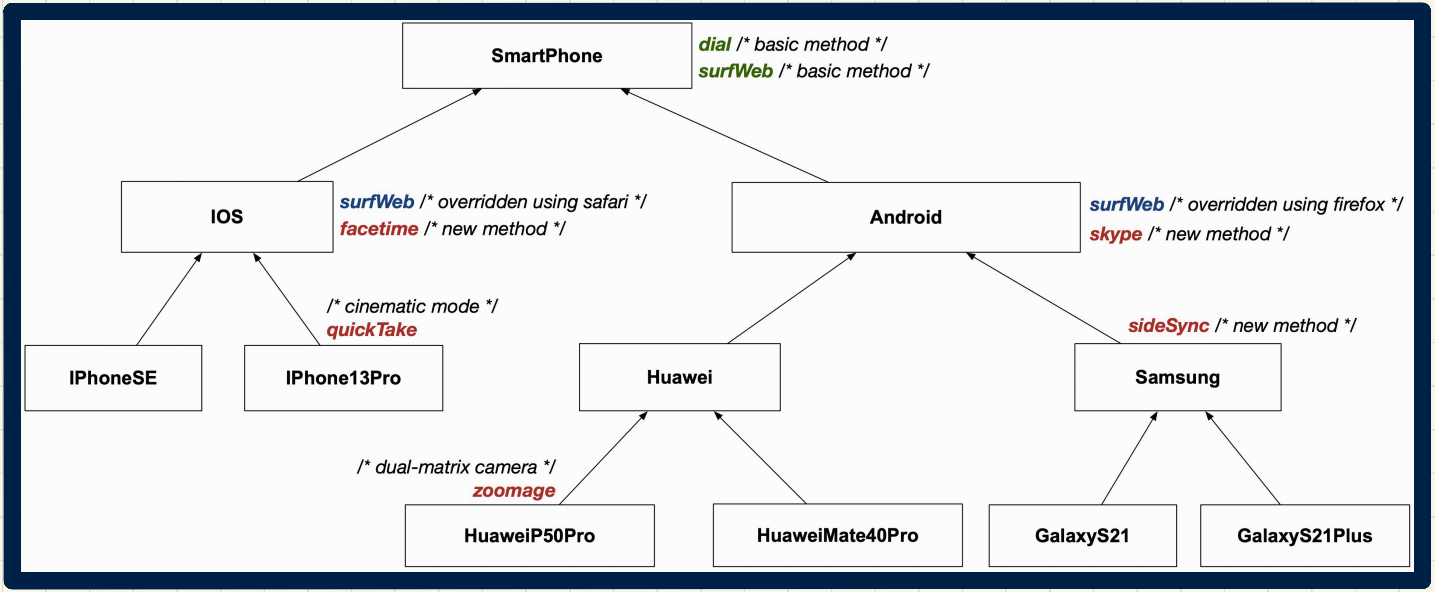
Declarations:

```
IOS sp1;
SmartPhone sp2;
```

Substitutions:

```
sp1 = sp2;
```

Rules of Substitutions (3)



Declarations:

```
IOS sp1;
HuaweiP50Pro sp2;
```

Substitutions:

```
sp1 = sp2;
```

Visualization: Static Type vs. Dynamic Type

Declaration:

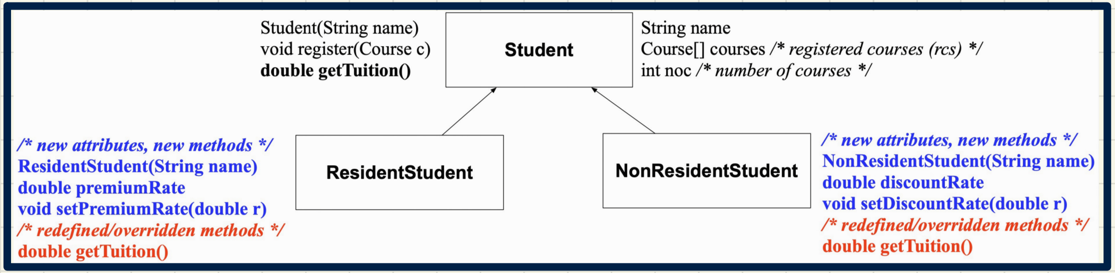
```
Student s;
```

Substitution:

```
s = new ResidentStudent("Rachael");
```

Static Type: Expectation

Dynamic Type: Accumulation of Code



Student(String name)
void register(Course c)
double getTuition()

Student

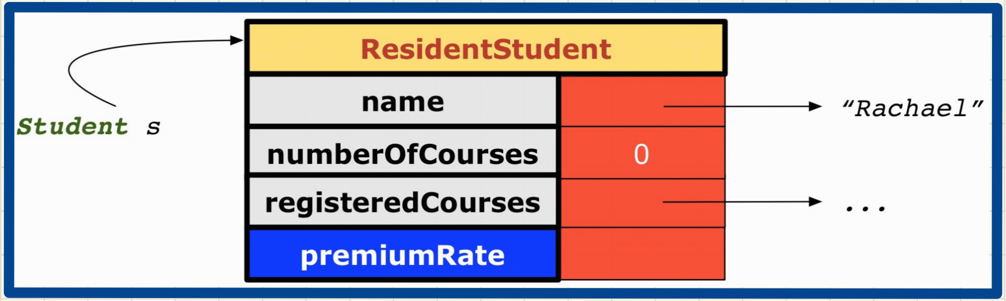
String name
Course[] courses /* *registered courses (rcs)* */
int noc /* *number of courses* */

/ new attributes, new methods */*
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/ redefined/overridden methods */*
double getTuition()

ResidentStudent

NonResidentStudent

/ new attributes, new methods */*
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/ redefined/overridden methods */*
double getTuition()



Student s

ResidentStudent

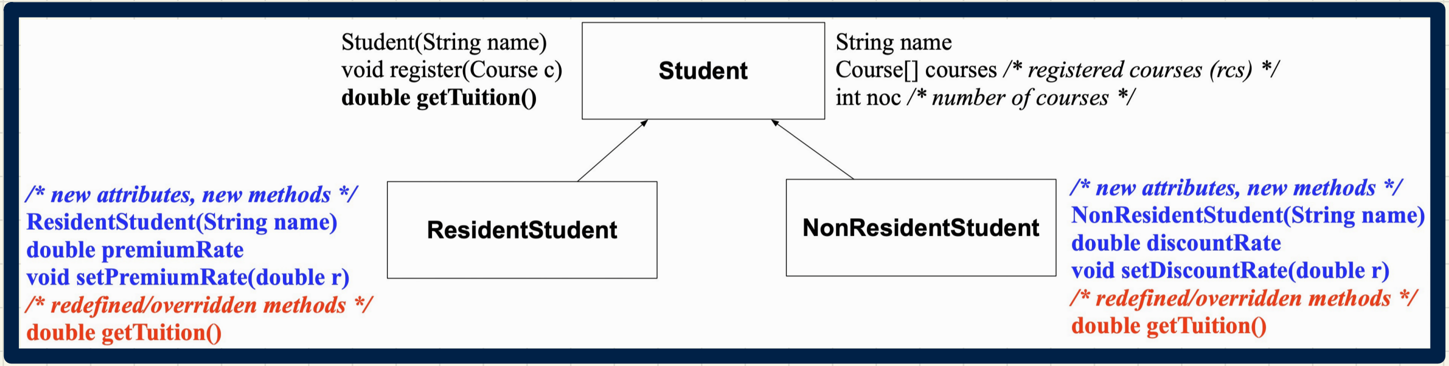
name → "Rachael"

numberOfCourses 0

registeredCourses → ...

premiumRate

Change of Dynamic Type (1.1)



Student(String name)
void register(Course c)
double getTuition()

Student

String name
Course[] courses /* *registered courses (rcs)* */
int noc /* *number of courses* */

/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()

ResidentStudent

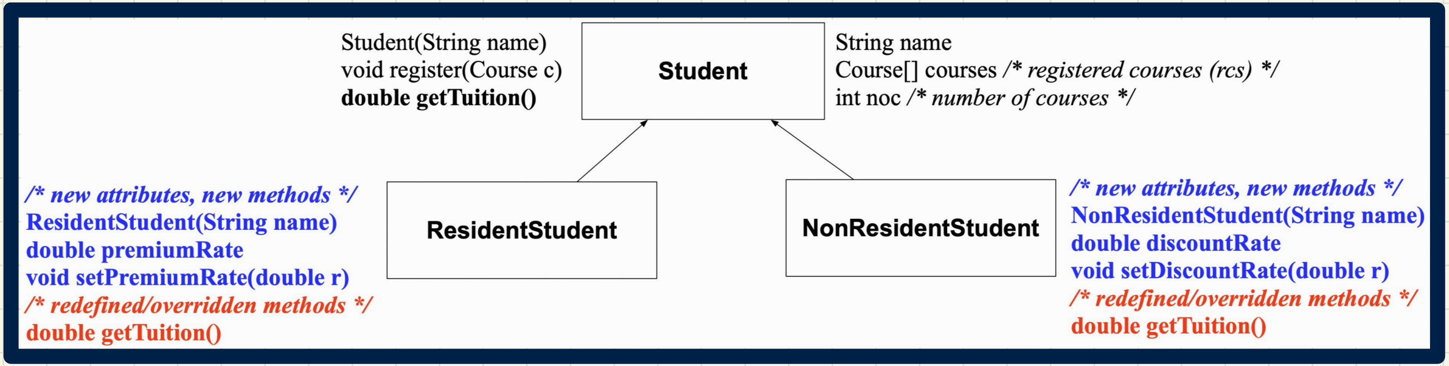
NonResidentStudent

/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

Example 1:

```
Student jim = new ResidentStudent(...);
jim = new NonResidentStudent(...);
```

Change of Dynamic Type (1.2)



Student(String name)
void register(Course c)
double getTuition()

String name
Course[] courses /* *registered courses (rcs)* */
int noc /* *number of courses* */

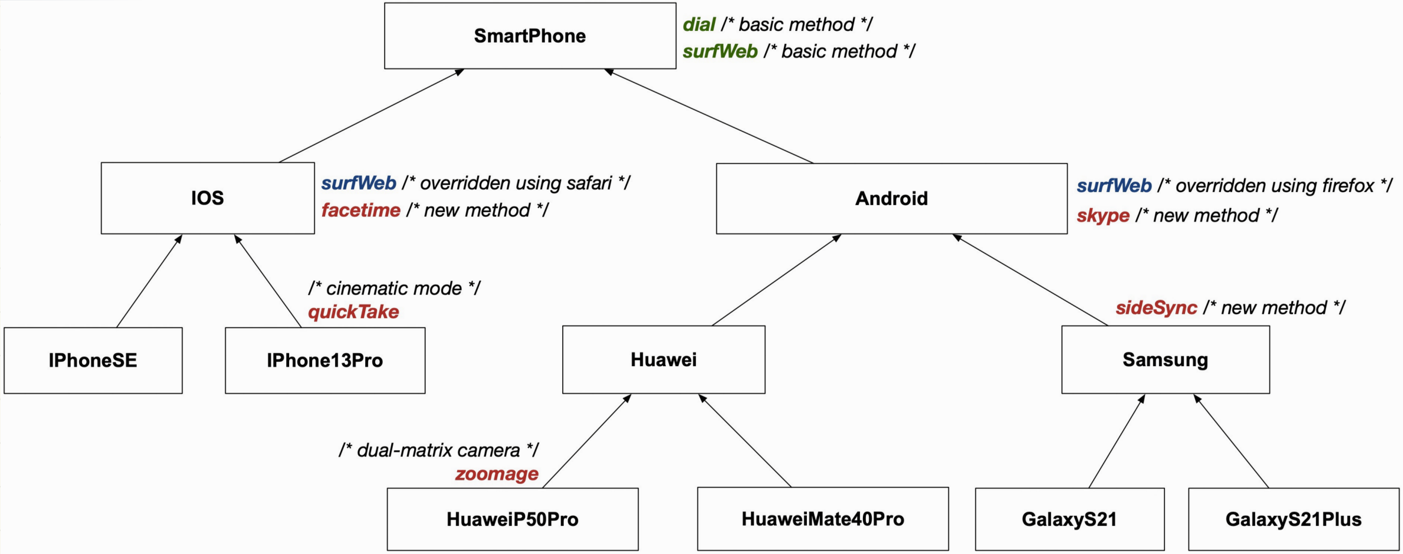
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()

/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

Example 2:

ResidentStudent jeremy = **new** **Student**(...);

Change of Dynamic Type: Exercise (1)



SmartPhone

dial /* basic method */
surfWeb /* basic method */

IOS

surfWeb /* overridden using safari */
facetime /* new method */

Android

surfWeb /* overridden using firefox */
skype /* new method */

IPhoneSE

IPhone13Pro

/* cinematic mode */
quickTake

Huawei

Samsung

sideSync /* new method */

HuaweiP50Pro

/* dual-matrix camera */
zoomage

HuaweiMate40Pro

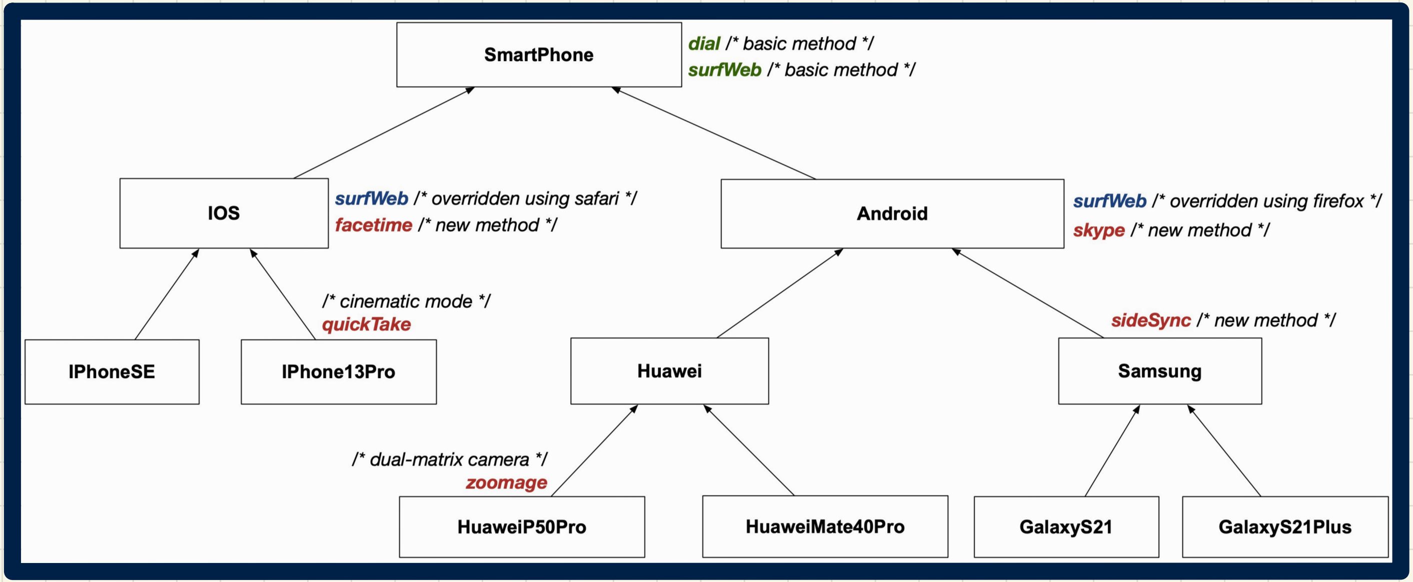
GalaxyS21

GalaxyS21Plus

Exercise 1:

```
Android myPhone = new HuaweiP50Pro(...);
myPhone = new GalaxyS21(...);
```

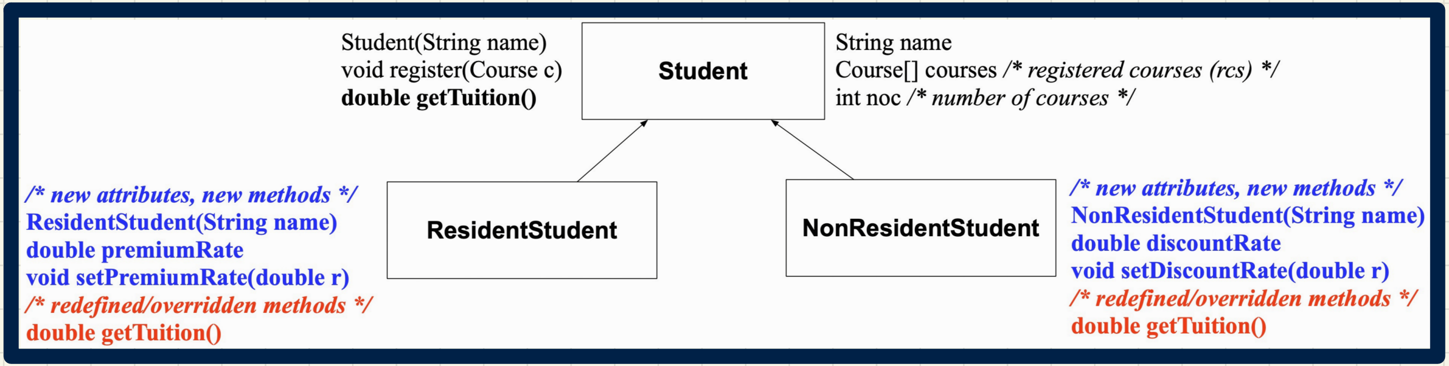
Change of Dynamic Type: Exercise (2)



Exercise 2:

```
IOS myPhone = new HuaweiP50Pro(...);
myPhone = new GalaxyS21(...);
```

Change of Dynamic Type (2.1)



Student(String name)
void register(Course c)
double getTuition()

String name
Course[] courses /* *registered courses (rcs)* */
int noc /* *number of courses* */

/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()

/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

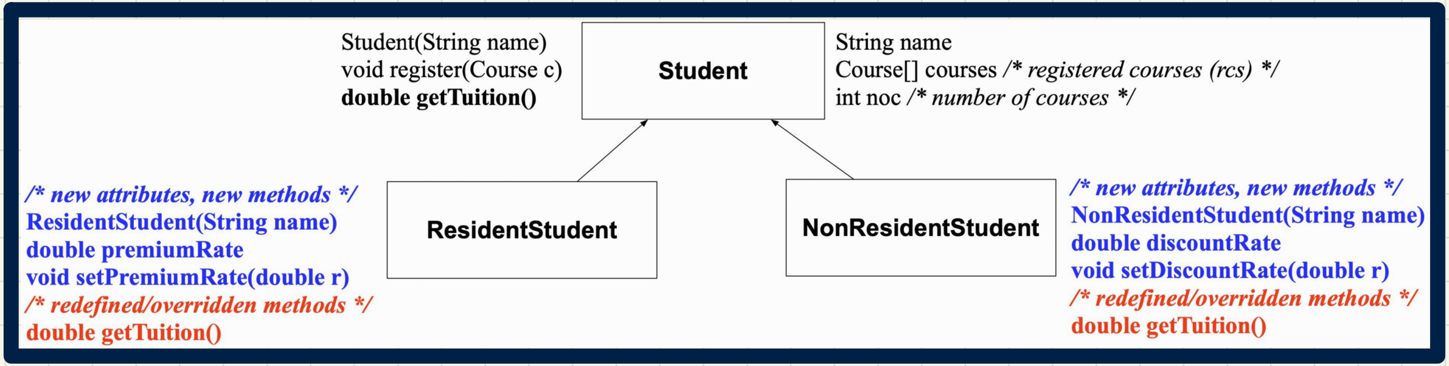
Given:

```
Student jim = new Student(...);
ResidentStudent rs = new ResidentStudent(...);
NonResidentStudent nrs = new NonResidentStudent(...);
```

Example 1:

```
jim = rs;
println(jim.getTuition());
jim = nrs;
println(jim.getTuition());
```

Change of Dynamic Type (2.2)



Student(String name)
void register(Course c)
double getTuition()

Student

String name
Course[] courses /* *registered courses (rcs)* */
int noc /* *number of courses* */

ResidentStudent

/ new attributes, new methods */*
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/ redefined/overridden methods */*
double getTuition()

NonResidentStudent

/ new attributes, new methods */*
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/ redefined/overridden methods */*
double getTuition()

Given:

```
Student jim = new Student(...);
ResidentStudent rs = new ResidentStudent(...);
NonResidentStudent nrs = new NonResidentStudent(...);
```

Example 2:

```
rs = jim;
println(rs.getTuition());
nrs = jim;
println(nrs.getTuition());
```

Polymorphism and Dynamic Binding

Polymorphism:

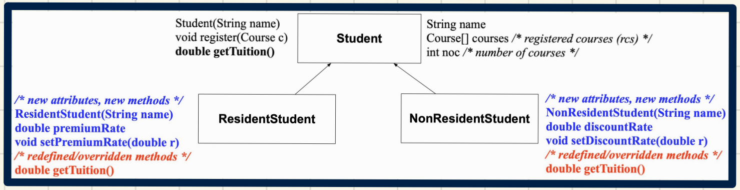
An object's **static type** may allow **multiple** possible **dynamic types**.

⇒ Each **dynamic type** has its **version** of method.

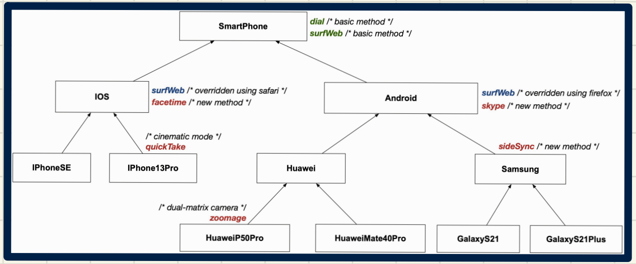
Dynamic Binding:

An object's **dynamic type** determines the **version** of method being invoked.

```
Student jim = new ResidentStudent(...);
jim.getTuition();
jim = new NonResidentStudent(...);
jim.getTuition();
```


Student(String name)
void register(Course c)
double getTuition()
Student
String name
Course[] courses /* registered courses (rcs) */
int noc /* number of courses */
/* new attributes, new methods */
ResidentStudent(String name)
double premiumRate
void setPremiumRate(double r)
/* redefined/overridden methods */
double getTuition()
ResidentStudent
NonResidentStudent
/* new attributes, new methods */
NonResidentStudent(String name)
double discountRate
void setDiscountRate(double r)
/* redefined/overridden methods */
double getTuition()

```
SmartPhone sp1 = new IPhone13Pro(...);
SmartPhone sp2 = new GalaxyS21(...);
sp1.surfWeb();
sp1 = sp2;
sp1.surfWeb();
```



Recap: Static Types vs. Dynamic Types

```
C1 v1 = new C3(...);
C2 v2 = new C4(...);
v1.m();
v2.m();
v1 = v2;
v1.m();
v2.m();
```

Exercises on Eclipse:

+ SMS (variable assignments)
+ Smart Phones (hierarchy + variable assignments)

Static Types and Anticipated Expectations

```
class A {
  void m1() { … }
}
class B extends A { }


class C extends A {}
```

```
B obj1 = new A();


A = obj2 = new A();
B obj3 = (B) obj2;
```