CSCI 132:
Basic Data Structures and Algorithms

Lecture 5: More Java + Inheritance

Reese Pearsall

Fall 2023

*All images are stolen from the internet

Announcements

- Good job on lab 1. Lab 2 will be posted later today. After today, you can complete it

- Program 1 will be posted later, but won't be due for another couple weeks

NO CLASS ON MONDAY (HOLIDAY)

Announcements

How to determine where a method should go

Announcements

Downloading and importing an entire Java project into your IDE

Warm Up

Write a program that will randomly generate a number between 0 and 10, and the user will have to guess what the number is. The program should give hints if the random number is higher or lower than their guess

Inheritance is a mechanism in Java that allows for a class to acquire instance fields and methods from another class

In Java, we use the **`extends`** keyword to indicate that a class is inheriting from another

```
public class Programmer extends Employee {
}
```

The `Programmer` class inherits from the `Employee` class

```java
public class Programmer extends Employee {

private String programming_language;

public Programmer(String name, int id, int salary, String lan) {
        super(name,id,salary);
        this.programming_language = lan;
}

public String getLanguage() {
        return this.programming_language;
}


}
```

Programmer.java

```java
public class Employee {

private String name;
private int emp_id;
private int salary;

public Employee(String name, int id, int salary) {
        this.name = name;
        this.emp_id = id;
        this.salary = salary;
}

public String getName() {
        return this.name;
}
```

Employee.java

The `Programmer` class inherits from the `Employee` class

```java
public class Programmer extends Employee {

private String programming_language;

public Programmer(String name, int id, int salary, String lan) {
        super(name,id,salary);
        this.programming_language = lan;
}

public String getLanguage() {
        return this.programming_language;
}


}
```

Programmer.java

```java
public class Employee {

private String name;
private int emp_id;
private int salary;

public Employee(String name, int id, int salary) {
        this.name = name;
        this.emp_id = id;
        this.salary = salary;
}

public String getName() {
        return this.name;
}
```

Employee.java

```java
Programmer reese = new Programmer("Reese Pearsall", 1234, 90000, "Python");
System.out.println(reese.getName());
```

`getName()` is not defined in the `Programmer` class, but because the `Programmer` class *inherits* from the `Employee` class, the reese object has access to the **`getName()`** method

```java
public class Programmer extends Employee {

private String programming_language;

public Programmer(String name, int id, int salary, String lan) {
        super(name,id,salary);
        this.programming_language = lan;
}

public String getLanguage() {
        return this.programming_language;
}


}
```

Programmer.java

```java
public class Employee {

private String name;
private int emp_id;
private int salary;

public Employee(String name, int id, int salary) {
        this.name = name;
        this.emp_id = id;
        this.salary = salary;
}

public String getName() {
        return this.name;
}
```

Employee.java

Inherited!

```java
Programmer reese = new Programmer("Reese Pearsall", 1234, 90000, "Python");
System.out.println(reese.getName());
```

`getName()` is not defined in the `Programmer` class, but because the `Programmer` class *inherits* from the `Employee` class, the reese object has access to the **`getName()`** method

```java
public class Programmer extends Employee {

private String programming_language;

public Programmer(String name, int id, int salary, String lan) {
        super(name,id,salary);
        this.programming_language = lan;
}

public String getLanguage() {
        return this.programming_language;
}


}
```

Programmer.java

```java
public class Employee {

private String name;
private int emp_id;
private int salary;

public Employee(String name, int id, int salary) {
        this.name = name;
        this.emp_id = id;
        this.salary = salary;
}

public String getName() {
        return this.name;
}
```

Not inherited! (but the getter methods are)

Employee.java

`private` instance fields and methods are **not** inherited

```java
public class Programmer extends Employee {

private String programming_language;

public Programmer(String name, int id, int salary, String lan) {
        super(name,id,salary);
        this.programming_language = lan;
}

public String getLanguage() {
        return this.programming_language;
}


}
```

Programmer.java

```java
public class Employee {

protected String name;
protected int emp_id;
protected int salary;

public Employee(String name, int id, int salary) {
        this.name = name;
        this.emp_id = id;
        this.salary = salary;
}

public String getName() {
        return this.name;
}
```

Employee.java

Now this instance fields will be inherited ☺

`private` instance fields and methods are **not** inherited

We can make instance fields `protected`, which means they are still private to other classes, but now they can be inherited

```java
public class Programmer extends Employee {

private String programming_language;

public Programmer(String name, int id, int salary, String lan) {
        super(name,id,salary);
        this.programming_language = lan;
}

public String getLanguage() {
        return this.programming_language;
}


}
```

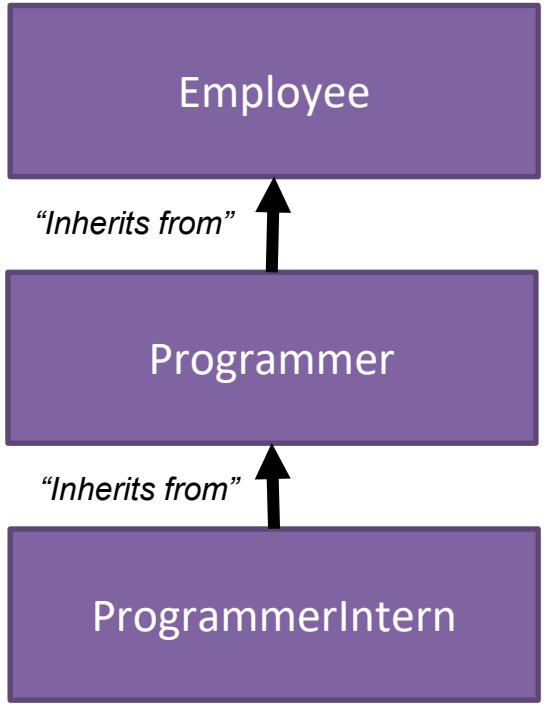
Programmer.java

```java
public class Employee {

protected String name;
protected int emp_id;
protected int salary;

public Employee(String name, int id, int salary) {
        this.name = name;
        this.emp_id = id;
        this.salary = salary;
}

public String getName() {
        return this.name;
}
```

Employee.java

The `super` keyword is used to reference the parent class. Just using `super()` will call the parent constructor

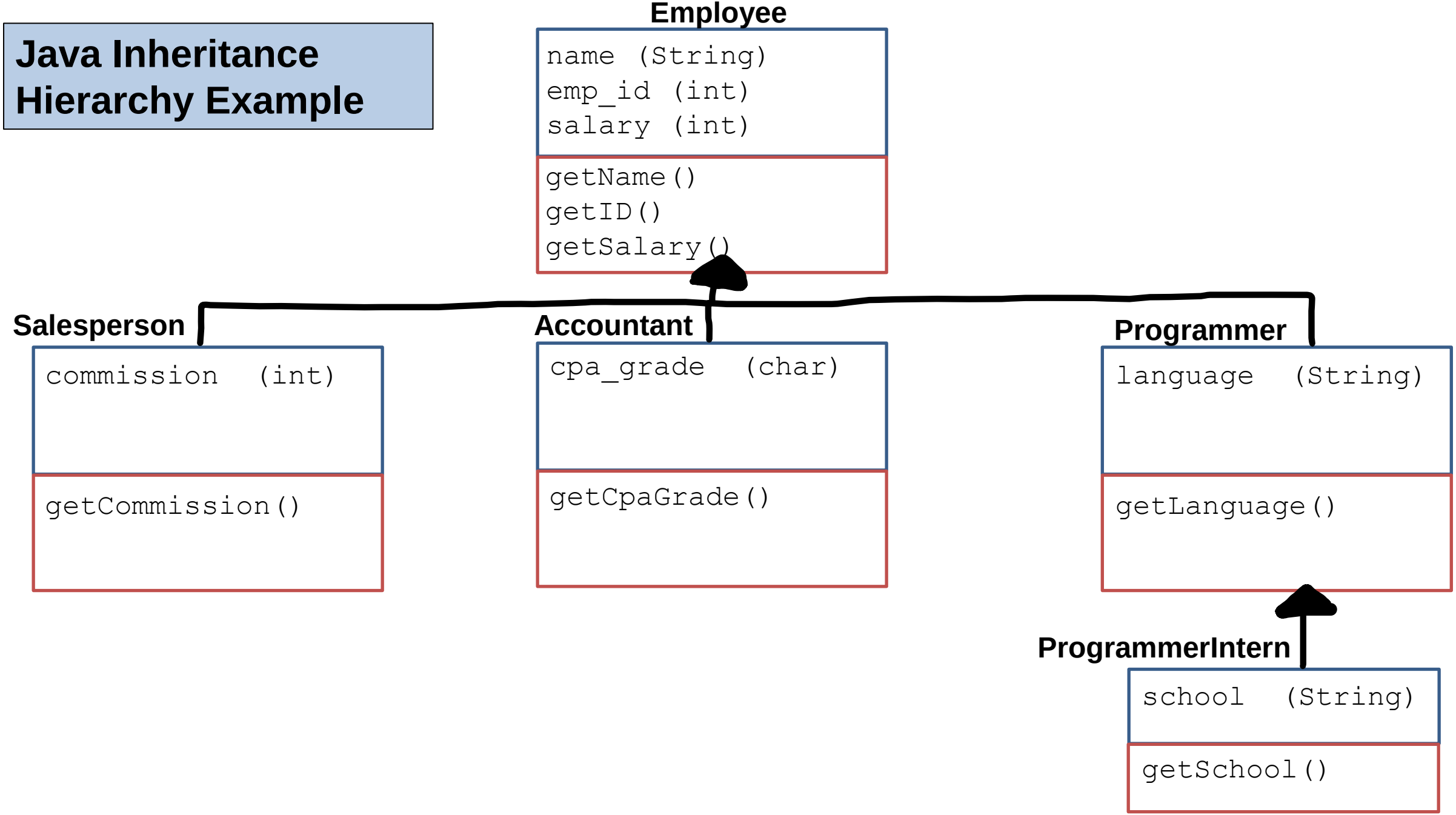
```java
public class Employee {

}

public class Programmer extends Employee {

}


public class ProgrammerIntern extends Programmer{
}
```



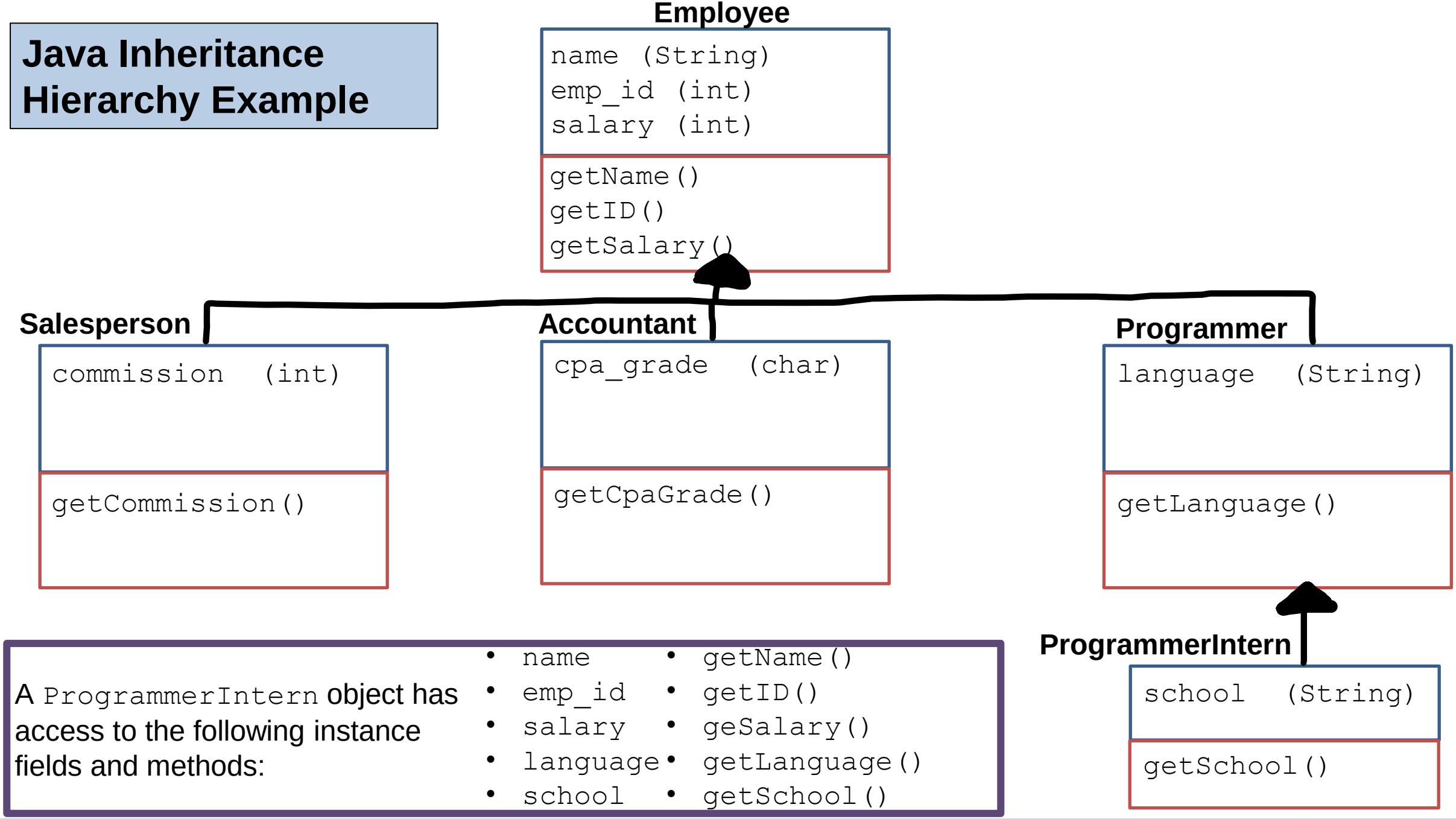
In Java, we can only inherit from *one* class (but that one class we inherit from can also inherit from another class)

In this example, `ProgrammerIntern` indirectly has access to the `Employee` class instance fields/methods because the `Programmer` class inherits from `Employee`

Java Inheritance Hierarchy Example



Java Inheritance Hierarchy Example



A `ProgrammerIntern` object has access to the following instance fields and methods:

- `name`
- `emp_id`
- `salary`
- `language`
- `school`
- `getName()`
- `getID()`
- `geSalary()`
- `getLanguage()`
- `getSchool()`

(and also the local + parent constructor method)

Method Precedence

Employee.java

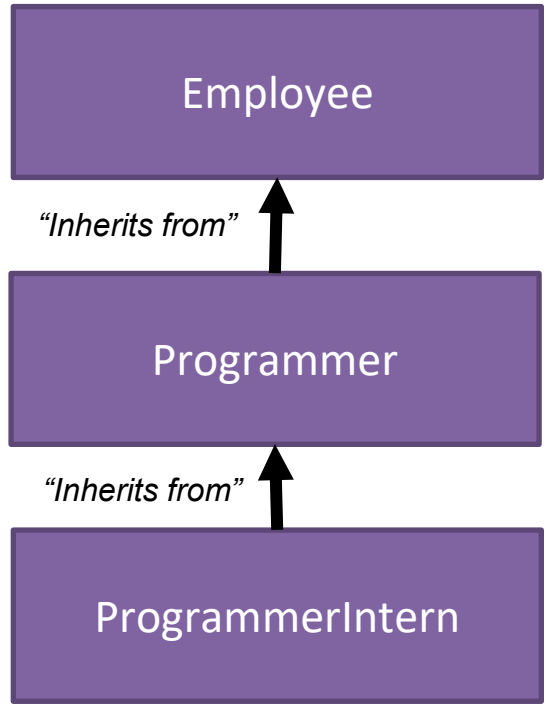
```
public String getName() {
        System.out.println("Method #1 (Employee)");
}
```

Programmer.java

```
public String getName() {
        System.out.println("Method #2 (Programmer)");
}
```

ProgrammerIntern.java

```
public String getName() {
        System.out.println("Method #3 (ProgrammerIntern)");
}
```



What if we define the exact same method in three different classes?

```
ProgrammerIntern intern1 = new ProgrammerIntern("Sally", …);
intern1.getName()
```

What will get printed out?

Method Precedence

Employee.java

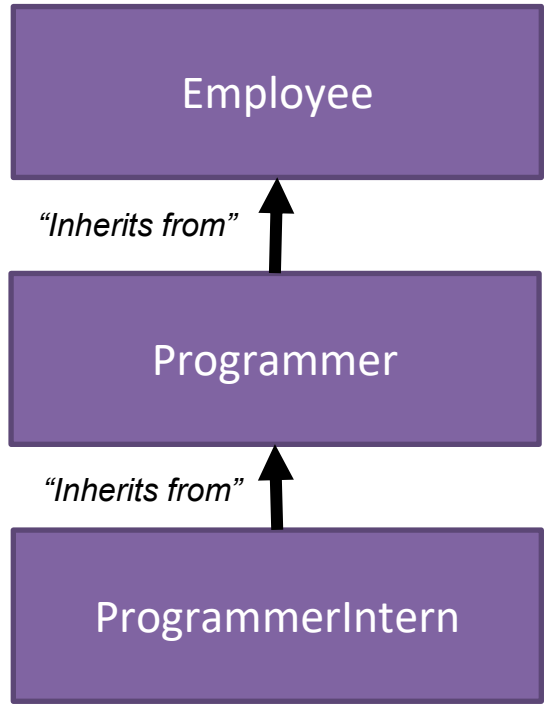
```
public String getName() {
        System.out.println("Method #1 (Employee)");
}
```

Programmer.java

```
public String getName() {
        System.out.println("Method #2 (Programmer)");
}
```

ProgrammerIntern.java

```
public String getName() {
        System.out.println("Method #3 (ProgrammerIntern)");
}
```



What if we define the exact same method in three different classes?

```
ProgrammerIntern intern1 = new ProgrammerIntern("Sally", …);
intern1.getName()
```

What will get printed out?

Output

```
Method #3 (ProgrammerIntern)
```

Takeaway: Java will first look at the child class, and then the parent class

Method Precedence

Employee.java

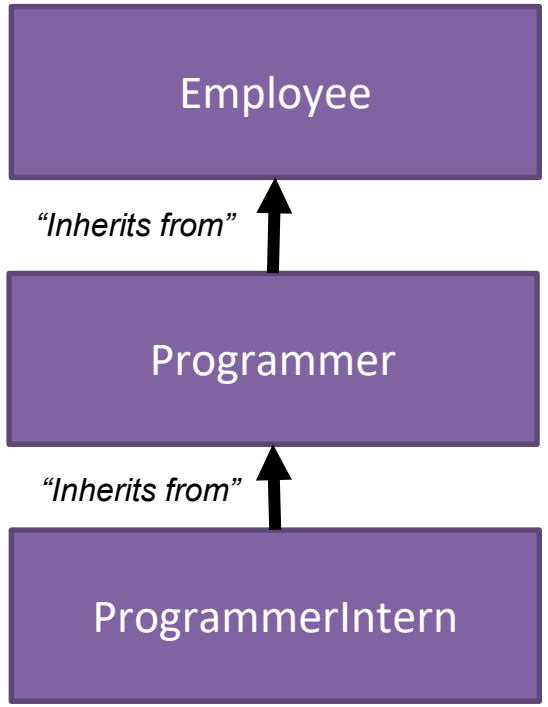
```
public String getName() {
        System.out.println("Method #1 (Employee)");
}
```

Programmer.java

```
public String getName() {
        System.out.println("Method #2 (Programmer)");
}
```

ProgrammerIntern.java

(method deleted)



What if we define the exact same method in three different classes?

```
ProgrammerIntern intern1 = new ProgrammerIntern("Sally", …);
intern1.getName()
```

What will get printed out?

Output

```
Method #2 (Programmer)
```

Takeaway: Java will first look at the child class, and then the parent class