CSCI 132:
Basic Data Structures and Algorithms

Inheritance

Reese Pearsall

Spring 2024

Announcements

Others : why do you always use i,j variabes in loops?

Programmers :

IT'S THE LAW

- Lab 3 due **tomorrow** at 11:59 PM

- Program 1 is posted, due on Feb 16
  (I updated the .txt file ☺)

I will be handing out rubber ducks this Friday (2/9) and the following Monday

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

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

```java
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;
}
```

Inherited!

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;       ✗   Not inherited! (but
private int emp_id;        ✗   the getter methods
private int salary;        ✗   are)

✓ 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

`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;
        }
```

Now this instance fields will be inherited ☺

Employee.java

`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;
}
```
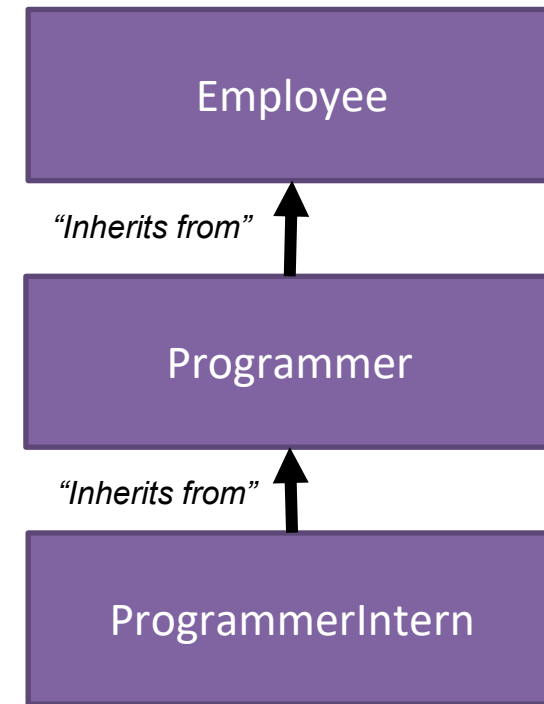
Employee.java

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{
}
```

Employee

"Inherits from"
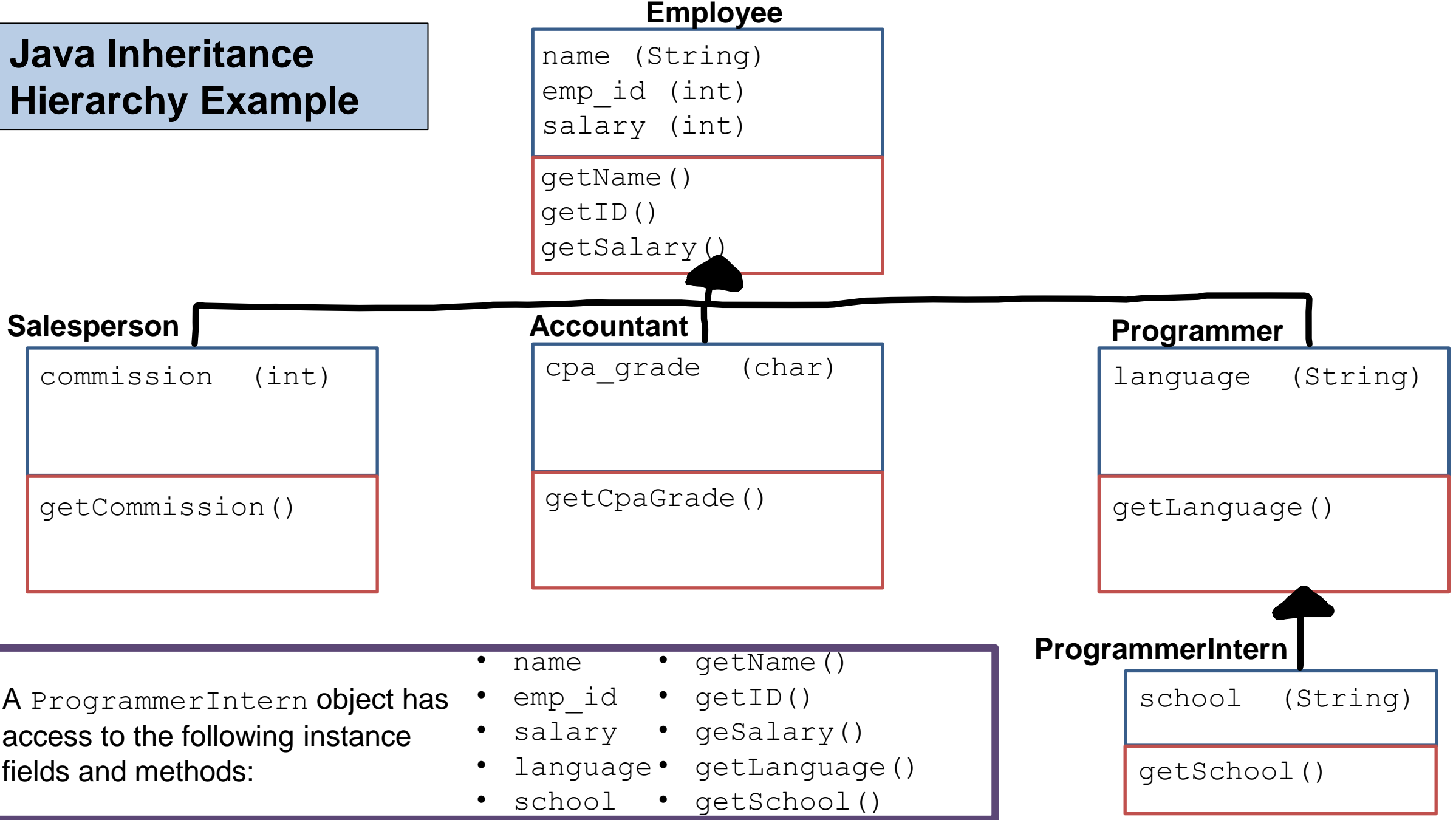
Programmer

"Inherits from"

ProgrammerIntern

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

Employee

name (String)
emp_id (int)
salary (int)
getName()
getID()
getSalary()

Salesperson

commission (int)
getCommission()

Accountant

cpa_grade (char)
getCpaGrade()

Programmer

language (String)
getLanguage()

ProgrammerIntern

school (String)
getSchool()

MONTANA STATE UNIVERSITY

Java Inheritance Hierarchy Example

Employee

name (String)
emp_id (int)
salary (int)
getName()
getID()
getSalary()

Salesperson

commission (int)
getCommission()

Accountant

cpa_grade (char)
getCpaGrade()

Programmer

language (String)
getLanguage()

ProgrammerIntern

school (String)
getSchool()

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

- name
- emp_id
- salary
- language
- school

- getName()
- getID()
- geSalary()
- getLanguage()
- getSchool()

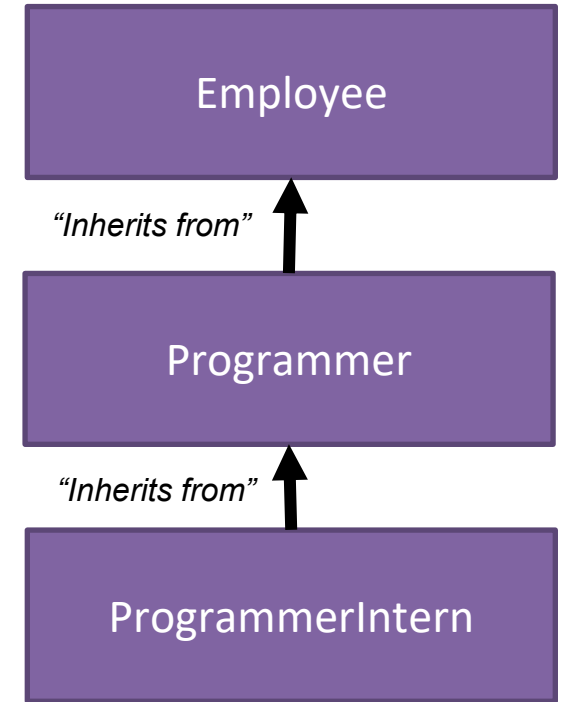(and also the local + parent constructor method)

Method Precedence

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

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

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


Employee

"Inherits from"

Programmer

"Inherits from"

ProgrammerIntern

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

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

intern1.getName()
```
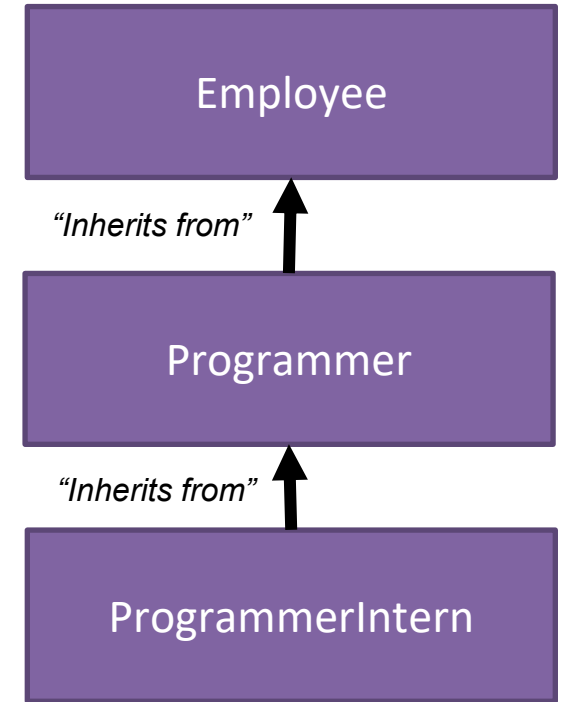
What will get printed out?

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)");
}
```

Employee

"Inherits from"

Programmer

"Inherits from"

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?
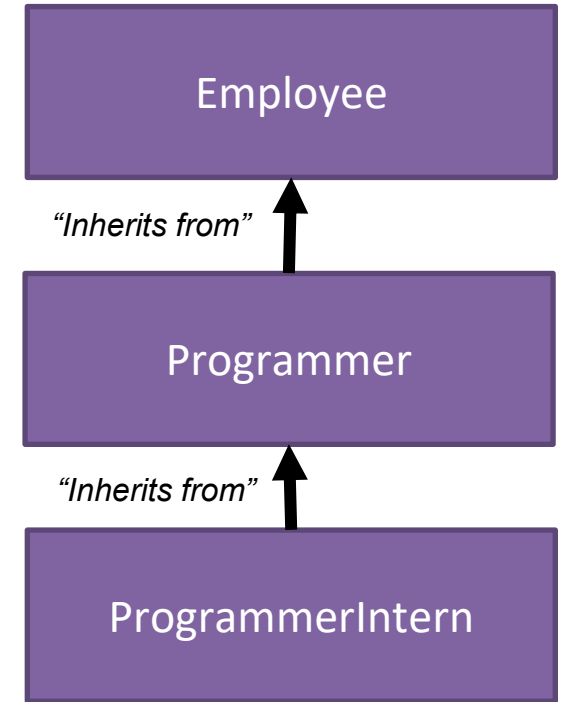
Output

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

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

Method Precedence

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

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

ProgrammerIntern.java

(method deleted)

Employee

"Inherits from"

Programmer

"Inherits from"

ProgrammerIntern

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

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

intern1.getName()
```

Output

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

What will get printed out?

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