CSCI 132:
Basic Data Structures and Algorithms

Inheritance

Reese Pearsall

Spring 2025

Announcements

- Lab 3 due tomorrow at 11:59 PM

I will be handing out rubber ducks this Friday (2/7)

- I'll bring them for most of next week too, and office hours

We’ve been hired to keep track of employees at the university

MSU has the following employees:

- Professors
- Teaching Assistants
- Advisors
- Programmers

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;
}
```

Employee.java

Not inherited! (but the getter methods are)

`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;
}


}
```

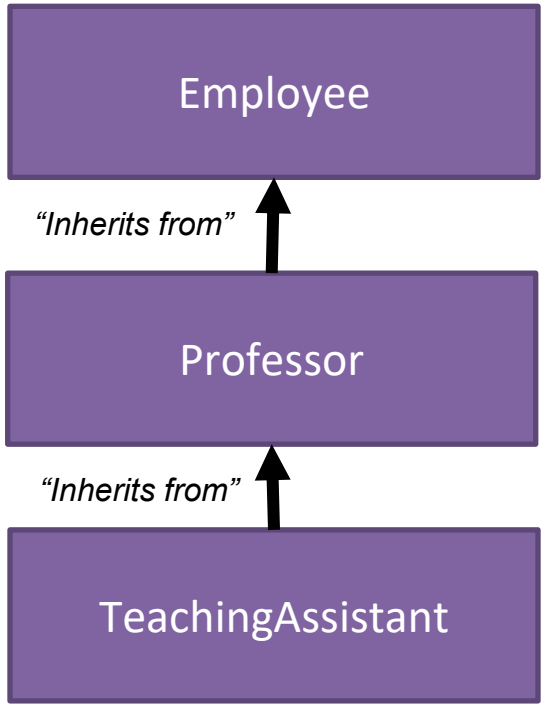
Programmer.java

```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

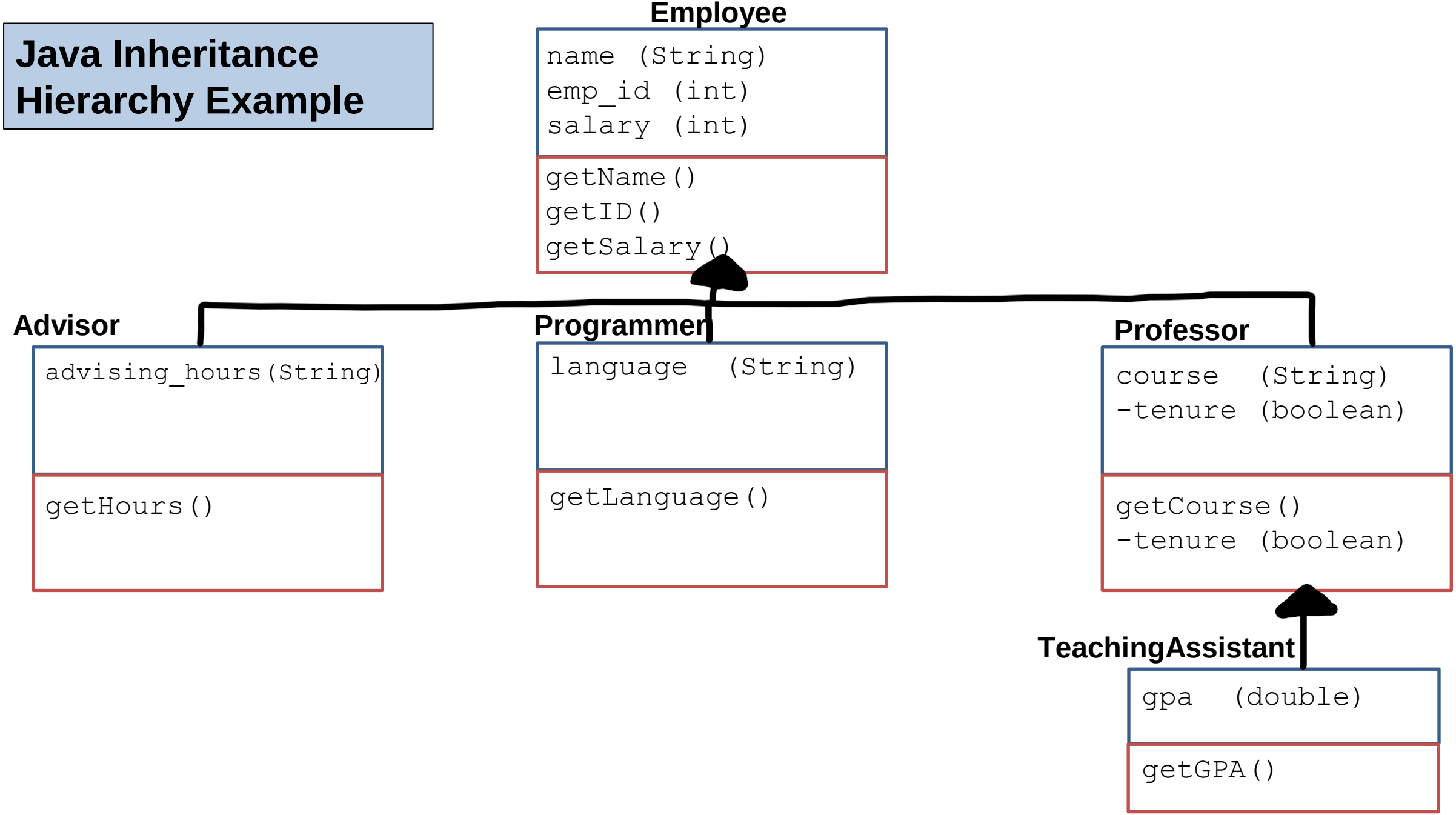
```java
public class Employee {

}

public class Professor extends Employee {

}


public class TeachingAssistant extends Professor{
}
```



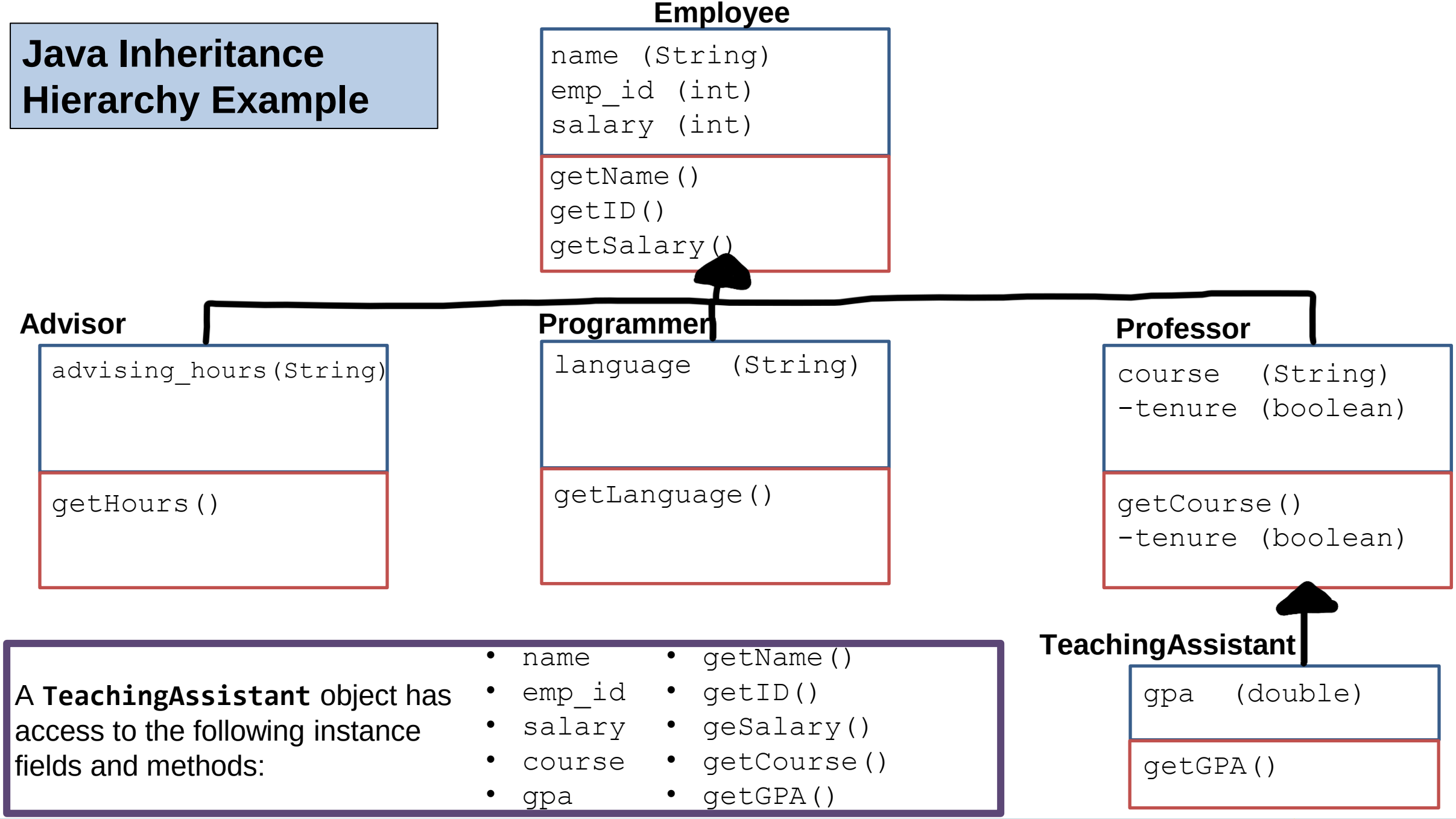
In Java, we can only inherit from *one* class (but that one class we inherit from can also inherit from another class)

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

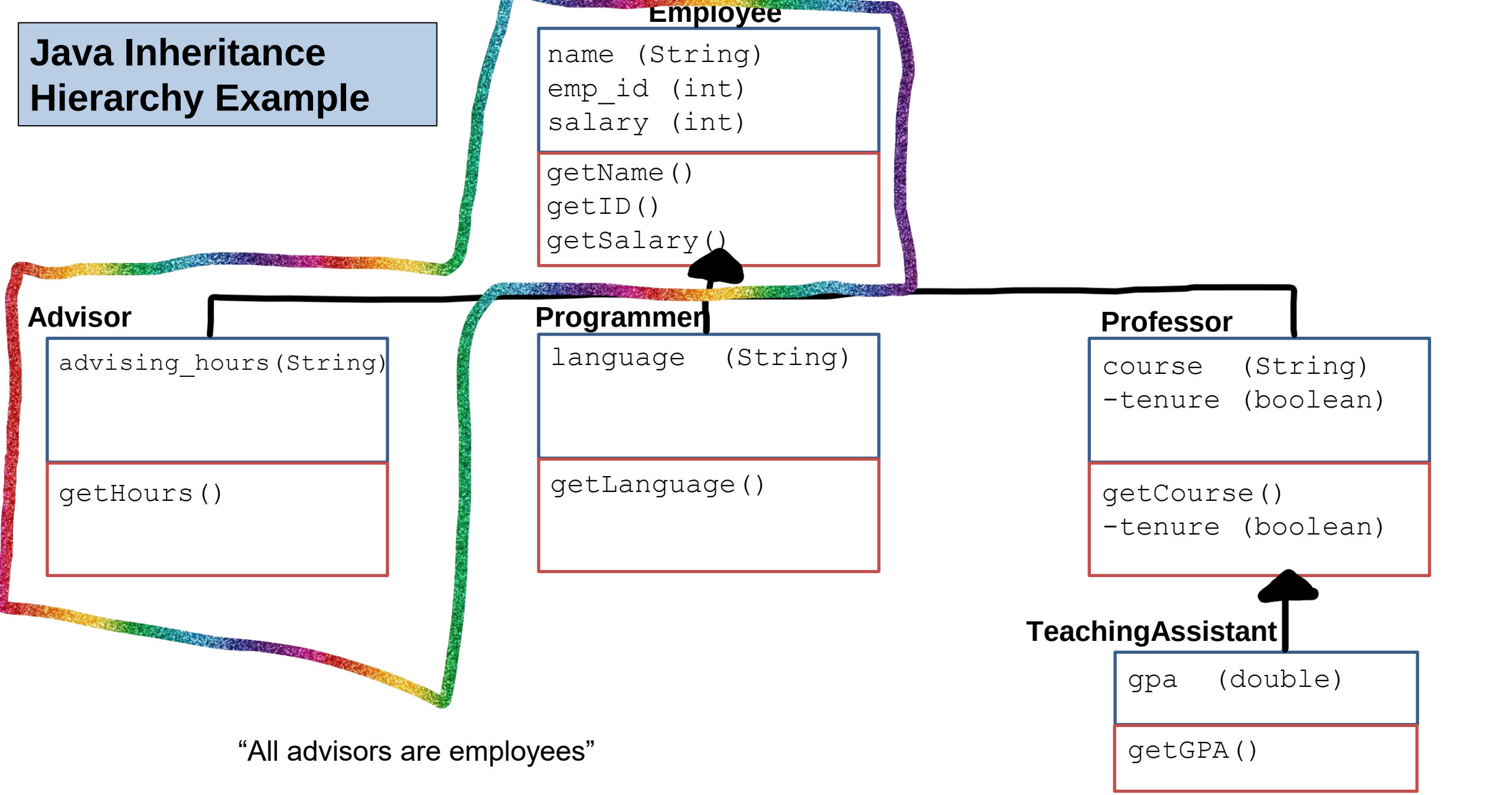


A **`TeachingAssistant`** object has access to the following instance fields and methods:

- `name`
- `emp_id`
- `salary`
- `course`
- `gpa`
- `getName()`
- `getID()`
- `geSalary()`
- `getCourse()`
- `getGPA()`

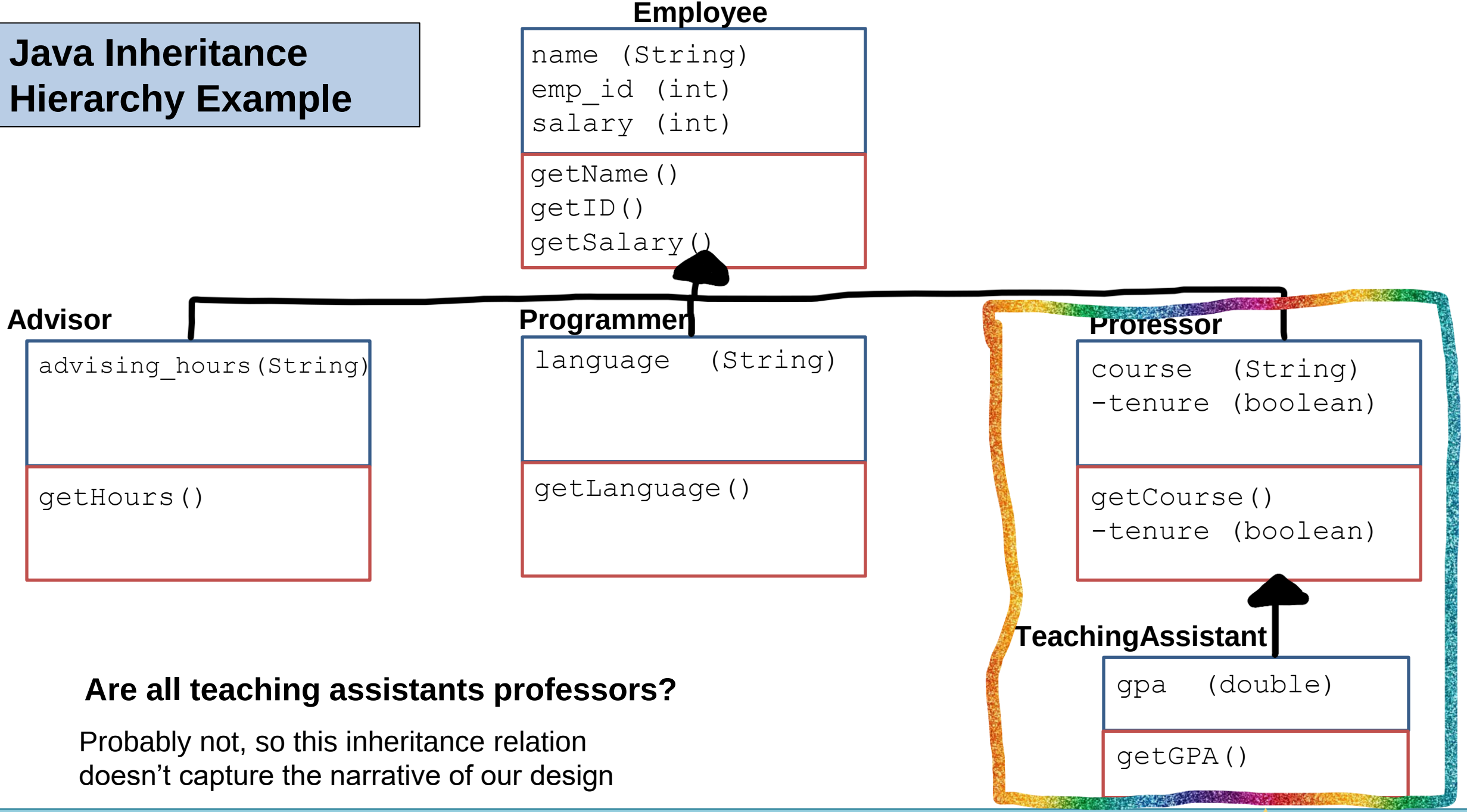
(and also the local + parent constructor method)

Java Inheritance Hierarchy Example



"All advisors are employees"

Java Inheritance Hierarchy Example



Are all teaching assistants professors?

Probably not, so this inheritance relation doesn't capture the narrative of our design

In Java, we can have multiple constructors.
→ Constructors must have different arguments

```java
//constructor for normal professor
public Professor(String n, int eid, int s, String course, boolean t) {
        super(n, eid, s);
        this.course = course;
        this.tenure = t;
}
//constructor for TA inheritee
public Professor(String n, int eid, int s, String course) {
        super(n, eid, s);
        this.course = course;
}
```

This constructor is used when making a professor

This constructor is used when the TA class calls their super constructor

(TA's do not have tenure, so we run a slightly different constructor)

Method Precedence

Employee.java

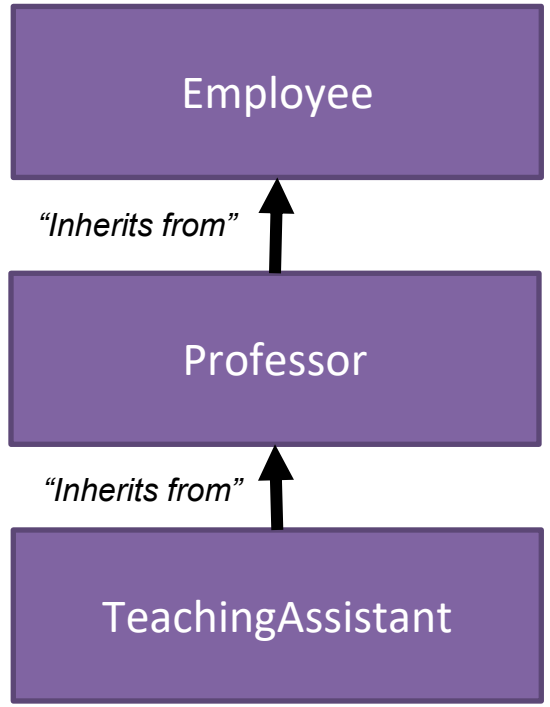
```
public String getName() {
        System.out.println("Method #1 (Employee)");
}
```

Professor.java

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

TeachingAssistant.java

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



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

```
TeachingAssistant ta1 = new TeachingAssistant("Sally", …);
ta1.getName()
```

What will get printed out?

Method Precedence

Employee.java

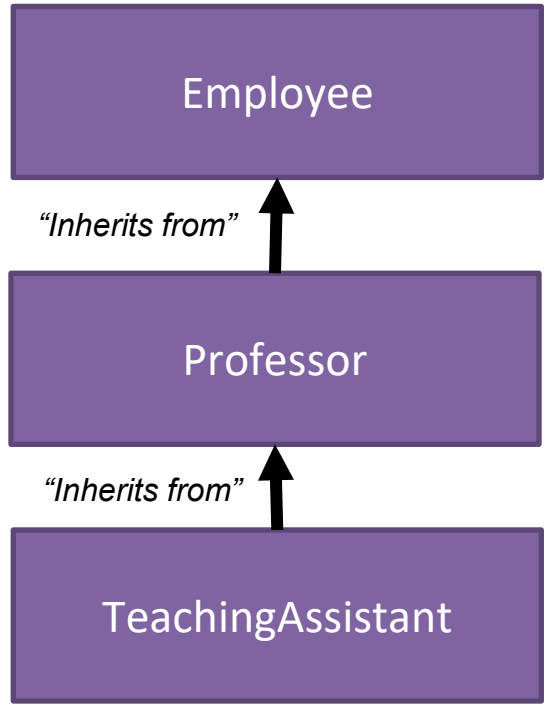
```
public String getName() {
        System.out.println("Method #1 (Employee)");
}
```

Professor.java

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

TeachingAssistant.java

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



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

```
TeachingAssistant ta1 = new TeachingAssistant("Sally", …);
ta1.getName()
```

What will get printed out?

Output

```
Method #3 (TeachingAssistant)
```

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

Method Precedence

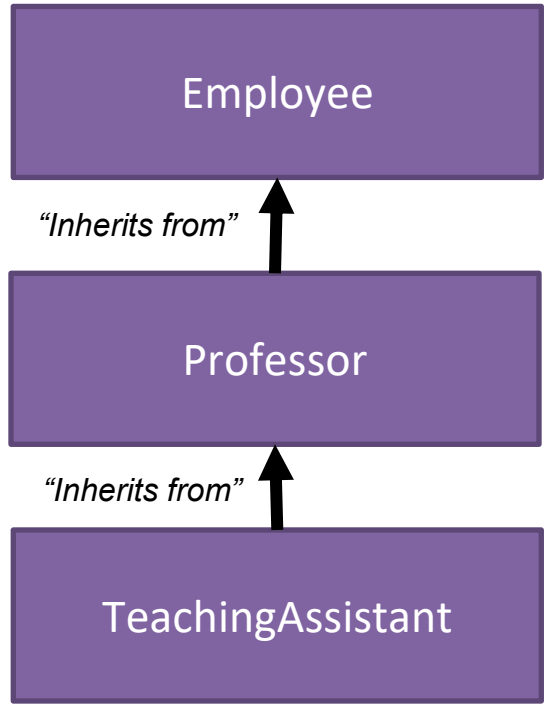
Employee.java

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

Professor.java

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

(method deleted)



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

```
TeachingAssistant ta1 = new TeachingAssistant("Sally", …);
ta1.getName()
```

What will get printed out?

Output

```
Method #2 (Professor)
```

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

Abstract Classes are restricted classes that cannot be used to create objects. To access it, it must be inherited from another class.

```
public abstract class Employee{

        …

}
```

```
Employee e = new Employee("Sally", 55555, 65000);
```

```
Error: Cannot instantiate the type Employee
```

You **cannot** create instances of an abstract class.

```
Professor p1 = new Professor("Clem", 12345, 100000, "ESOF 422", true);
```

Instead, we use objects from another class that inherits from the abstract class