Subclasses, Constructors and Overriding Methods

We created a class to represent employees paid by the hour.

We had the class extend Employee so that we can use everything Employee already has:

public class HourlyEmployee extends Employee {

Note that we do not have to give HourlyEmployee a name or number because it inherits the getter/setter methods for name and number from Employee.

We just need to add fields and getter/setter methods for the new features of hourly employee's: the number of hours worked and the amount paid per hour

Remember, that we want our fields private and the getter/setter methods public.

private int hours;

private double hourlyRate;

public int getHours() {

return hours;

}

public void setHours(int hours) {

this.hours = hours;

}

public double getHourlyRate() {

return hourlyRate;

}

public void setHourlyRate(double hourlyRate) {

this.hourlyRate = hourlyRate;

}

Now, if we compile this code, we get an error! The error states that there is a problem with the Employee constructor!

Recall how constructors work:

1) The first line of a constructor must be a call to another constructor.

If you do not explicitly give a constructor call, Java adds a call to the constructor of the parent class that takes no input.

2) The constructors do the following when called:

a) The constructor call that is the first line of the constructor is called.

b) All fields of the instance are initialized.

c) The rest of the constructor body is executed.

Why did we get an error?

We did not provide a constructor for HourlyEmployee and so Java automatically provided a constructor that takes no input and calls the constructor of Employee that takes no input.

- But Employee does not have a constructor that takes no input!

The solution is to create a constructor for HourlyEmployee.

To call the constructor of Employee, we use super() with the appropriate input.

super acts the same as this except it refers to the parent class of the current class.

We chose to call the constructor of Employee that takes two Strings, a number and a name.

public HourlyEmployee(String number, String name) {

super(number, name);

}

Overriding Methods

HourlyEmployee inherits all non-private methods of Employee. (Constructors are not inherited.)

This is a problem because HourlyEmployee inherits getSalary and setSalary, but HourlyEmployees do not have a salary.

A class can override any method of the parent class by creating a method with the same name and parameter signature.

The return type must also be the same or, if the return type is not primitive, narrower.

We chose to override setSalary so that it does not set the salary of the hourly employee.

public void setSalary(int salary) {

}

Another Override Example:

Inside Employee, we created a method to compare salaries:

public boolean earnsMoreThan(Employee e) {

return this.getSalary() > e.getSalary();

}

Note that the parameter can be an HourlyEmployee object. Remember polymorphism! An object whose type is HourlyEmployee also has type Employee and also has type Object.

So the HourlyEmployee object can be used anywhere that expects a value of type Employee.

The problem is that an hourly employee could earn more than an regular salaried employee, but that is not reflected in the salary.

The use of salary was a bad design decision, but sometimes we are stuck with using classes in an API that do not exactly meet our design needs.

Here, we can get around the poor design of Employee by having HourlyEmployee override the getSalary method.

public int getSalary() {

return (int)(hours \* hourlyRate);

}

Note, that the Employee class at least uses getSalary() to get the salary amount. We were lucky that the code did not use the field directly:

public boolean earnsMoreThan(Employee e) {

return this.salary > e.salary; // a poor technique!

}

Using the fields would make it harder for HourlyEmployee to overcome the issues with Employee. If Employee used salary in earnsMoreThan, we would have to make sure that both

the setHours() and setHourlyRate() methods called super.setSalary() to update the salary field. (Likewise any other code that changed the hours or hourly rate would need to update

the salary field through setSalary().

Instead, because the Employee class used getSalary() in the earnsMoreThan method, we only had to override getSalary() in HourlyEmployee to make earnsMoreThan work with hourly employees.

GOOD OBJECT-ORIENTED STYLE: Always use the getter/setter methods instead of the fields in your (non-constructor) code. That way classes that extends your class can override the getter/setter methods if needed.

So, let's improve the getSalary overridden method to also not use fields:

public int getSalary() {

return (int)(getHours() \* getHourlyRate());

}