实施雇员体系结构以进行薪资处理

步骤1
列出属于该层次结构的类。

在我们的情况下，问题描述明确列出三个类：HourlyEmployee、SalariedEmployee和Manager。我们需要一个类来表达它们的共通性：Employee。

步骤2
将类组织到继承层次结构中。

在这里是这些类的继承关系图。

步骤3
确定类的共同职责。

为了发现它们的共同职责，为处理对象编写伪代码。

对于每个雇员
   打印雇员的姓名。
   读取工作小时数。
   计算应支付的工资。

我们得出结论，Employee超类具有这些职责：
   获取姓名。
   计算给定数目的小时应支付的工资。

WORKED EXAMPLE 9.1

实施雇员体系结构以进行薪资处理

您需要实现薪资处理程序，用于不同类型的雇员。

- 时薪雇员按小时率支付，但如果他们在一周内工作超过40小时，额外工作时间以“时薪的1.5倍”支付。
- 薪资雇员按薪资支付，无论他们工作多少小时。
- 经理是薪资雇员，他们按薪资和奖金支付。

您的程序应能计算雇员的支付。对于每个雇员，提示他们输入一周的工作小时数，然后显示他们应得的薪资。
Step 4  Decide which methods are overridden in subclasses.

In our example, there is no variation in getting the employee’s name, but the salary is computed differently in each subclass, so weeklyPay will be overridden in each subclass.

```java
/**
   * An employee with a name and a mechanism for computing weekly pay.
   */
public class Employee
{
    ...  
    /**
       * Gets the name of this employee.
       * @return the name
       */
    public String getName() { ... }

    /**
       * Computes the pay for one week of work.
       * @param hoursWorked the number of hours worked in the week
       * @return the pay for the given number of hours
       */
    public double weeklyPay(int hoursWorked) { ... }
}
```

Step 5  Declare the public interface of each class.

We will construct employees by supplying their name and salary information.

```java
public class HourlyEmployee extends Employee
{
    ...  
    /**
       * Constructs an hourly employee with a given name and weekly wage.
       */
    public HourlyEmployee(String name, double wage) { ... }
    ...  
}
```

```java
public class SalariedEmployee extends Employee
{
    ...  
    /**
       * Constructs a salaried employee with a given name and annual salary.
       */
    public SalariedEmployee(String name, double salary) { ... }
    ...  
}
```

```java
public class Manager extends SalariedEmployee
{
    ...  
    /**
       * Constructs a manager with a given name, annual salary and weekly bonus.
       */
    public Manager(String name, double salary, double bonus) { ... }
    ...  
}
```

These constructors need to set the name of the Employee object. We will add a method setName to the Employee class for this purpose:
public class Employee
{
    //
    public void setName(String employeeName) {
        //
    }
    //
}

Of course, each subclass needs a method for computing the weekly wages:

// This method overrides the superclass method
public double weeklyPay(int hoursWorked) {
    //
}

In this simple example, no further methods are required.

Step 6
Identify instance variables.

All employees have a name. Therefore, the Employee class should have an instance variable name. (See the revised hierarchy below.)

What about the salaries? Hourly employees have an hourly wage, whereas salaried employees have an annual salary. While it would be possible to store these values in an instance variable of the superclass, it would not be a good idea. The resulting code, which would need to make sense of what that number means, would be complex and error-prone.

Instead, HourlyEmployee objects will store the hourly wage and SalariedEmployee objects will store the annual salary. Manager objects need to store the weekly bonus.

Step 7
Implement constructors and methods.

In a subclass constructor, we need to remember to set the instance variables of the superclass:

public SalariedEmployee(String name, double salary)
{
    setName(name);
    annualSalary = salary;
}

Here we use a method. Special Topic 9.1 shows how to invoke a superclass constructor. We use that technique in the Manager constructor:

public Manager(String name, double salary, double bonus)
{
    super(name, salary)
    weeklyBonus = bonus;
}
The weekly pay needs to be computed as specified in the problem description:

```java
public class HourlyEmployee extends Employee {
    ...
    public double weeklyPay(int hoursWorked) {
        double pay = hoursWorked * hourlyWage;
        if (hoursWorked > 40) {
            // Add overtime
            pay = pay + ((hoursWorked - 40) * 0.5) * hourlyWage;
        }
        return pay;
    }
}
```

```java
public class SalariedEmployee extends Employee {
    ...
    public double weeklyPay(int hoursWorked) {
        final int WEEKS_PER_YEAR = 52;
        return annualSalary / WEEKS_PER_YEAR;
    }
}
```

In the case of the Manager, we need to call the version from the SalariedEmployee superclass:

```java
public class Manager extends Employee {
    ...
    public double weeklyPay(int hours) {
        return super.weeklyPay(hours) + weeklyBonus;
    }
}
```

**Step 8** Construct objects of different subclasses and process them.

In our sample program, we populate an array of employees and compute the weekly salaries:

```java
Employee[] staff = new Employee[3];
staff[0] = new HourlyEmployee("Morgan, Harry", 30);
staff[1] = new SalariedEmployee("Lin, Sally", 52000);
staff[2] = new Manager("Smith, Mary", 104000, 50);

Scanner in = new Scanner(System.in);
for (Employee e : staff) {
    System.out.print("Hours worked by "+ e.getName()+": ");
    int hours = in.nextInt();
    System.out.println("Salary: "+ e.weeklyPay(hours));
}
```

The complete code for this program is contained in the ch09/worked_example_1 directory of your source code.