WORKED EXAMPLE 9.1

Implementing an Employee Hierarchy for Payroll Processing

Problem Statement Your task is to implement payroll processing for different kinds of employees.

- Hourly employees get paid an hourly rate, but if they work more than 40 hours per week, the excess is paid at "time and a half".
- Salaried employees get paid their salary, no matter how many hours they work.
- Managers are salaried employees who get paid a salary and a bonus.

Jose Luis Pelaez Inc./Getty Images, Inc.

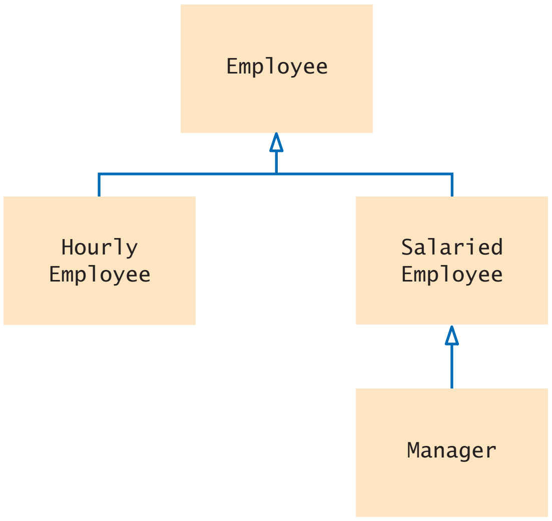
Your program should compute the pay for a collection of employees. For each employee, ask for the number of hours worked in a given week, then display the wages earned.

Step 1 List the classes that are part of the hierarchy.

In our case, the problem description lists three classes: `HourlyEmployee`, `SalariedEmployee`, and `Manager`. We need a class that expresses the commonality among them: `Employee`.

Step 2 Organize the classes into an inheritance hierarchy.

Here is the inheritance diagram for our classes:



Step 3 Determine the common responsibilities of the classes.

In order to discover the common responsibilities, write pseudocode for processing the objects.

```
For each employee
    Print the name of the employee.
    Read the number of hours worked.
    Compute the wages due for those hours.
```

We conclude that the `Employee` superclass has these responsibilities:

```
Get the name.
Compute the wages due for a given number of hours.
```

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.

```
/**
   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.

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

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

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) { . . . }
```

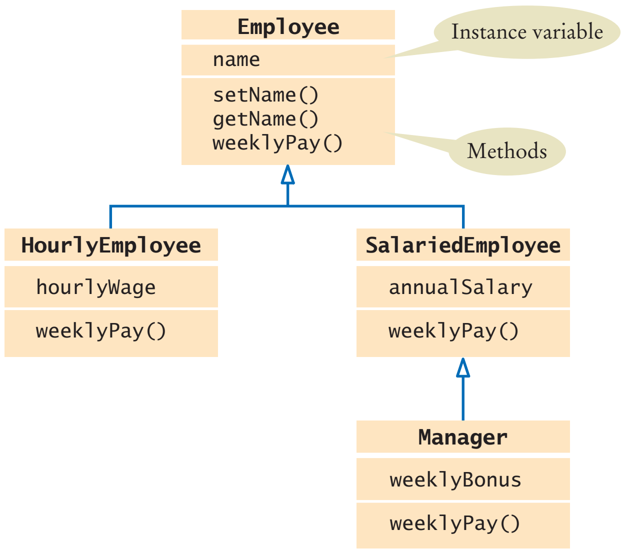
In this simple example, no further methods are required.

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:

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

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:

```
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:

```
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.