

```
/*  
By:          Ark - www.engineeringserver.com  
Date:       April/06/2008  
Tutorial:   Employee application  
Version:    1.0
```

```
*/
```

```
/*  
 * requirements:
```

First: Write an Employee class:

- a. Write the code for an Employee class, with private data variables String name and double salary.
- b. This class should have two constructors. One should take no parameters. The other should take parameters to initialize both of the instance variables.
- c. For each variable you will need two methods, one to set the value and one to get the value. Let me suggest these names for the methods: void setName(), String getName(), void setSalary(), double getSalary().
- d. This class should have a method, String toString(), which follows the pattern ineptly presented on pages 372-373 of the book. In other words, toString() should return a string containing the following information: The name of the class, followed by a left bracket, followed by the names of the variables contained in an object and their values, followed by a right bracket, all appearing on a single line.

Second: You also need to write the code for a class Manager which is a subclass of the class Employee:

- a. This new class has an additional instance variable, String department.
- b. The new class should also have two constructors. One should take no parameters. The other should take three parameters, one for each instance variable that would belong to an object. Note that it will have to make use of a call to the super constructor.
- c. This class should also have its own methods, void setDepartment() and String getDepartment().
- d. This class should also have a method, String toString(). In the toString() method you do not make use of any toString() method defined above you in the class hierarchy. Instead, you make use of the "get" methods defined above you in the hierarchy to get the values of the inherited instance variables. A toString() method should state the name of the class that it occurs in, not the superclass, for example, but it should also return all instance variable values, including inherited ones.

Third: You also have to write the code for a class Executive which is a subclass of the class Manager.

- a. Note that the new class does not have an additional instance variable in it.
- b. You will need to write two constructors for the new class, one with no parameters and one with three.
- c. Since the class doesn't have any new variables, no new "set" or "get" methods should be needed.
- d. However, once again you need to write a toString() method. You can't simply use the inherited one because the name of the class is different. At the same time, all of the variable values that will be printed out are inherited.

Test your class hierarchy by writing a program that constructs at least one object of each class, and makes sure each instance variable has a value, whether this is accomplished by passing parameters in the constructor or whether it's done by using the "set" methods. Then make use of the objects' respective toString() methods to see their contents as output. (Note that if toString() is correctly implemented, you will know this because a call to System.out.println() will successfully print out the contents of an object without an explicit call to the toString() method.)

Part 2, 8 pts.

This is related to book exercise P9.5. To get a general idea, take a look at the way problem 9.5 is described in the book. Then refer to the specific directions given here. I am essentially asking you to do the same things, but in order to save you some needless work, I have rewritten the problem to make use of the Employee class and subclasses given above.

First: This requires no work on your part, just understanding. The variable salary already appears in the class hierarchy. For the purposes of this part of the assignment, let that variable stand for an hourly pay rate. As far as I'm concerned, you do not have to rename the variable. You may if you want to. The important thing is to treat the value stored in a salary variable as an hourly rate.

Second: Write the code for a new subclass of Employee, which will be called HourlyWorker. After completing the first part of the assignment you should be able to determine in general what belongs in the class definition for this class. In addition to any of that standard stuff, you need to implement a new method, double computePay(double hours). This method should compute and return the weekly pay of an hourly worker, giving standard rate to hours under 40 and time and a half to any hours over 40.

Third: Let your existing Manager class play the role of what the book calls the SalariedWorker. In other words, implement a double computePay(double hours) method in the Manager class also. However, in this implementation, you should actually ignore the number of hours passed down to it. Most companies still track the hours worked by salaried employees, but they are paid on the basis of a 40 hour work week no matter what. So to compute the pay for a manager, simply multiply the rate times 40 and return this value.

Fourth: Write a static method which should appear in the Employee class. This method should compute the pay for any employee. Getting this to work is a little complicated, but there is a reason for this madness. After it's all over with, if you can understand the point of the assignment, you've made progress on understanding how object orientation works.

a. Before you can write the static method, you need to write a double computePay(double hours) method in the Employee class. It is impossible to write a "correct" version of the method here because you need to know whether an employee is hourly or salaried. In other words, you have to write something bogus at the Employee class level. For the time being what you can do is simply put a statement like "return 0;" in the body of the method. In order to write the static method as directed below you have to have a method in the Employee class with the same name, parameter type, and return type as in the subclasses. Since this is a do-nothing implementation of the method it is clear that you would never actually want to use it on an Employee object, but it has other purposes.

b. Now, write the code for the static method in the Employee class, which will take this form: static double staticComputePay(Employee empin, double hoursin). The body of the method should contain a call like this: empin.computePay(hoursin). The reason for step "a", above, is that the parameter empin is typed as an Employee, and in the code you make a call empin.computePay(). This will only work if there is a computePay() method for the Employee class. This is the key point: This is a static method. You can call it with the explicit Employee parameter actually being either an HourlyWorker or a Manager and it should work correctly. The system will accept objects of either of those two subclasses as an Employee parameter, and when the call empin.computePay() is made, the system will correctly determine which computePay() method to call based on what kind of object the parameter actually is. This is an illustration of the topic referred to under the heading of polymorphism in the book.

Test your new methods by writing a program that constructs at least one object each of the classes Employee, HourlyWorker, and Manager, making sure each instance variable has a value, whether this is accomplished by passing parameters in the constructor or whether it's done by using the "set" methods. Then make calls of the form myobject.computePay(). Then make calls of the form Employee.staticComputePay(myobject...). When the parameter is an Employee the results should be bogus, but when the parameter is an HourlyWorker or a Manager you should get real results.

*/

```
package employee;

public class Main {

    public static void main(String[] args) {
        Employee E1 = new Employee("Edward Duke", 10.00);
        System.out.println(E1.toString() + "\t$" + E1.computePay(32));

        Employee E2 = new Employee("Peter Pattern", 11.00);
        System.out.println(E2.toString() + "\t$" + E2.computePay(32));

        Employee E3 = new Employee("Karl Bedlam", 22.00);
        System.out.println(E3.toString() + "\t$" + E3.computePay(32));

        System.out.println();

        HourlyWorker HW1 = new HourlyWorker("Edward Duke", 33.00, 32);
        System.out.println(HW1.toString());

        HourlyWorker HW2 = new HourlyWorker("Peter Pattern", 44.00, 48);
        System.out.println(HW2.toString());

        HourlyWorker HW3 = new HourlyWorker("Peter Pattern", 55.00, 48);
        System.out.println(HW3.toString());

        System.out.println();

        Manager M = new Manager("John Hulkie", 66.00, "Sony");
        System.out.println(M);

        System.out.println();

        Executive EX = new Executive("Homer Dipsaus", 800.00, "Microsoft");
        System.out.println(EX);
    }
}
```