# 44-542 Object-Oriented Programming Your Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Practice Lab Exam 100 pts

For each Java file that you create, you must include your first and last name as a comment at the beginning of the program file, using the **@author** notation.

**The Goal**

You are given employee data in a text file. Each employee record consists of four lines such as the following (two employees are shown):



http://www.nsacct.org/nsacctwww/nsastore/cpe/acat-practice-exams

Neel Sen

10000.0

10.0

Fine and Performing Arts

Atharv Sengupta

6.0

2.0

Humanities and Social Sciences

There are two types of employees: salaried and hourly. For each employee, there are four pieces of data:

* The full name of the employee (first and last, which can be kept combined in this exercise)
* The annual salary or hourly pay rate, for salaried and hourly employees respectively
* The extra pay or extra hours, for salaried and hourly employees, respectively
* The department to which the employee belongs

Process this week’s data from a text file and calculate the total amount owed to all employees in the file combined.

Read the whole description first – then begin creating your classes.

**The Process**

Create a new Java Application project in NetBeans, using your **last name**, followed by **Payroll**, as the project name, e.g. **CasePayroll or BandiPayroll**. Create a **lastnameanalyzer** package if it didn’t automatically.

1. (60 pts) Create a driver class (the application) called **LastNamePayroll** (e.g. BandhiPayroll).
2. (20 pts) Create an abstract super class called **AbstractEmployee**.
3. (10 pts) Create a subclass called **SalaryEmployee**.
4. (10 pts) Create a subclass called **HourlyEmployee**.

More information about each of these classes is included below.

**AbstractEmployee (20 pts)**

Create your public abstract **AbstractEmployee** class with two private attributes, one abstract method, a **toString()** method, a two-argument constructor, two getters, and two setters as described below.

1. **name** (of the type **String)**
2. **department** (of the type **String)**
3. **double calcPay**() (abstract method)
4. **toString()**that returns "AbstractEmployee{" + "name=" + name + ", department=" + department + '}';.
5. **AbstractEmployee** constructor that takes a **String** name and **String** department, in that order.
6. Create a getter and setter for each attribute.

**HourlyEmployee (10 pts)**

Create a public **HourlyEmployee** class that extends your abstract class, has two class variables, two class constants, two public methods that override the methods in the abstract class, and one constructor as described below.

1. a **double** **hourlyRate** (which will come from each employee data record)
2. a **double** **extraHours** (which will come from each employee data record)
3. a constant **OVERTIME double** (equal to 1.5)
4. a constant **BASE\_HOURS** **double** (equal to 40.0). Assume all hourly employees work 40 hours each week.
5. **double calcPay()** (return a **double** that is **hourlyRate**\***BASE\_HOURS** + **hourlyRate** \* **extraHours** \* **OVERTIME**)
6. use the default **toString()**autogenerated by Netbeans.
7. a 4-arg constructor (that takes **name**, **department**, **hourlyRate**, and **extraHours**)

**SalaryEmployee (10 pts)**

Create a public **SalaryEmployee** class that extends your abstract class, has two class variables, one class constant, two public methods that override the methods in the abstract class, and one constructor as described below.

1. a **double** **annualRate** (which will come from each employee data record)
2. a **double** **extraPay** (which will come from each employee data record)
3. a constant **WEEKS double** (equal to 52.0)
4. **double calcPay()**(return a **double that is annualRate/WEEKS + extraPay**)
5. use the default **toString()**autogenerated by Netbeans.
6. a 4-arg constructor (that takes **name**, **department**, **annualRate**, and **extraPay**)

**LastNamePayroll (60 pts) – See starting file.**

1. (5 pts) At the top of your driver class (**LastNamePayroll)**, before the main method,create the following **private** **final** attributes:

* A new **ArrayList** of type **AbstractEmployee** with the name **employees**.
* A new **ArrayList** of type **HourlyEmployee** with the name **salaryEmployees**.
* A new **ArrayList** of type **SalaryEmployee** with the name **hourlyEmployees**.
* A **String** **SMALL\_FILE\_NAME** set to "**output1.txt**";
* A **String** **LARGE\_FILE\_NAME** set to "**output2.txt**";

1. (2 pts) Create the **main** method as follows.

**public static void main(String[] args) {**

**final String DIR = System.getProperty("user.dir");**

**final String PARTIAL = "/src/ yournameanalyzer /";**

**final String PATH = DIR + PARTIAL;**

**YourNamePayroll. readDataCalculatePayAndDisplayInfo (PATH + SMALL\_FILE\_NAME);**

**// YourNamePayroll. readDataCalculatePayAndDisplayInfo (PATH + LARGE\_FILE\_NAME);**

**}**

The code can process two data files. Start with the small file until your program works correctly. When you have finished processing the small file, uncomment the line to process the larger file as well.

1. (50 pts) Complete your class by creating a public static void **readDataCalculatePayAndDisplayInfo** method that takes a single String parameter (providing the file path and name) to process the data as directed below.
2. (3 pts) In the body of this method, create and initialize thee local variables as follows:

* Initialize a **double** variable named **totalPay** to zero.
* Initialize a **File** variable named **file** representing the data file.
* Declare a Scanner variable and initialize it to null.

1. (6 pts) Continue to create the method as follows:

* Create a **Scanner** to read from the **file**.
* Type alt-enter to surround this with a try catch block.
* In the event of a **FileNotFoundException**, output a line to the console that consists of

SCANNER ERROR: followed by a space and the message from the exception.

1. (3 pts) Use the **Scanner** to read the data using a while loop.
2. (6 pts) In the body of this **while** loop, read the four parts of each employee record.
3. (3 pts) Create double variables to hold the numerical values. Hint: use the **Double.parseDouble** method.
4. (6 pts) If the pay rate (the first double) is greater than one hundred, create a new employee of type **SalaryEmployee** using your 4-arg constructor.

* Add the employee to your **employees** **ArrayList**.
* Add the employee to your **salaryEmployees** **ArrayList**

1. (6 pts) Otherwise, create a new employee of type **SalaryEmployee** using your 4-arg constructor.

* Add the employee to your **employees** **ArrayList**.
* Add the employee to your **hourlyEmployees** **ArrayList**.

1. (3 pts) You are done with our local scanner object, so close it.
2. (5 pts) Process each element of **employees** with an enhanced **for** loop. In the body of the **for** loop:

* Calculate the employee pay by calling the **calcPay()** method on each employee.
* Add each employee pay to the **totalPay**.

After the loop, output to the console:

* **"The total pay = " +** **totalPay**.

1. (5 pts) Call a static method named displaySalariedEmployees() that takes no arguments, and processes each element of **salaryEmployees** with an enhanced **for** loop. In the body of the loop, output the String returned by the **SalaryEmployee** **toString**() method.
2. (5 pts) Call a static method named displayHourlyEmployeeNames () that takes no arguments, and processes each element of **hourlyEmployees** with an enhanced **for** loop. In the body of the loop, output the String returned by the **HourlyEmployee** **getName**() method.
3. (3 pts) Call a static method named **displaySummary**() that outputs two **Strings**:

* "**The number of salaried employees = " +** **m**
* **"The number of hourly employees = " +** **n**

Where m and n are the count of each type.

1. (6 pts) Output: Once you get the following output for the smaller data file, uncomment the line to process the larger data file as well.

/\*

*output1.txt results:*

Total pay = 1567

SalaryEmployee{WEEKS=52.0, annualRate=10000.0, extraPay=10.0}

SalaryEmployee{WEEKS=52.0, annualRate=45000.0, extraPay=16.0}

SalaryEmployee{WEEKS=52.0, annualRate=25000.0, extraPay=4.0}

The number of salaried employees = 3

The number of hourly employees = 0

BUILD SUCCESSFUL (total time: 0 seconds)

\*/

/\*

*output2.txt* ***partial*** *results:*

Total pay = 390601

(…lots of details…)

The number of salaried employees = 396

The number of hourly employees = 407

\*/

**Scoring:**

|  |  |
| --- | --- |
| Class | Points |
| AbstractEmployee | 20 |
| HourlyEmployee | 10 |
| SalaryEmployee | 10 |
| LastNamePayroll | 60 |
| Exam Total | 100 |

|  |  |
| --- | --- |
| Payroll Part | Points |
| 1 | 3 |
| 2 | 6 |
| 3 | 3 |
| 4 | 6 |
| 5 | 3 |
| 6 | 6 |
| 7 | 6 |
| 8 | 3 |
| 9 | 5 |
| 10 | 5 |
| 11 | 5 |
| 12 | 3 |
| 13 | 6 |
| Payroll Total | 60 |

**Contents of output1.txt:**

Neel Sen

10000.0

10.0

Fine and Performing Arts

Aegir Bose

45000.0

16.0

Professional Education

Reyansh Sengupta

25000.0

4.0

English and Modern Languages