**Pre-requisite Knowledge**

Before attempting to complete this assignment, you should know:

1. Everything needed for program 6.
2. Class inheritance and polymorphism.
3. File input and output.
4. File exception handling.

**Task**

Write an abstract superclass encapsulating an employee:  An employee has two attributes:  a name and a position.  It has two abstract methods:  returning by how much the employee should be paid in the current week, and another returning the number of vacation days the employee has remaining in the calendar year.

This class has three non-abstract subclasses:  one encapsulating a manager, one encapsulating an hourly employee, and the other encapsulating an intern. In addition, there is a temporary employee, which is a subclass of an hourly employee.

A manager has attributes which are their yearly salary, the number of vacation days they have per year and the number of vacation days taken in the current calendar year. Managers are not paid overtime.

An hourly employee has attributed indicating the hourly wage of the employee, the number of hours worked during the current week, the number of vacation days they have per year and the number of vacation days taken in the current calendar year. Hourly employees are paid time and a half for all hours over 40 worked in the current week.

A temporary hourly employee is just like an hourly employee, except they have no vacation.

An intern has attributes that represent their weekly wage and who their supervisor is. They are not paid overtime, and they have no vacation days.

You will also create an interface class to represent whether or not the employee is married and how many children they have. There is an abstract method calcDeduct() which calculates how much money should be deducted from the withheld taxes, up to $500. If the employee is married, the deduction is $100, and the employee gets an additional $100 for each child, up to a maximum of $500. So if an employee is married with 2 children, only $200 (500 - 300) is withheld for taxes.

Design and write all these classes along with a client program that will read in all the employee information in from a file (see below for format) and then print out all the information associated with the employees, along with their pay for the week.  You can assume that all employees have $500 withheld from their pay for taxes, minus their deductions. Use polymorphism to allow all the employees to be stored in the same collection and for printing purposes.  Be sure and include all the possible exceptions that may occur within the reading of the file.  The program should not crash, but should produce error messages. You should also calculate the total number of vacation days remaining for all employees.

You must create the API documentation for your program.  This may be done in the class (javadoc format or not) or you may create a separate piece of documentation for this purpose (but I will still expect some comments in the code, they just do not need to be as complete).  Make sure in your main program that I can tell what test cases you are running.  Your program should be neat, easy to read, correctly indented, with appropriate comments.

The format of the input file should be as follows:

Line 1 – number of employees included in the file

For each employee in the file a set of lines will include:

String representing type of employee ("Manager", "Hourly", "Temporary", "Intern")

String representing the name

Wage - this will be the yearly salary for managers, the hourly wage for hourly and temporary, and the weekly wage for interns

For managers, the next line will contain number of vacation days, and number of vacation days taken. This is followed by a line that will contain an 'n' for not married or 'y' for married. The last line contains the number of children.

For hourly and temporary employees, the next line will contain the number of hours worked. For hourly only, the next line will contain number of vacation days, and number of vacation days taken. For both hourly and temporary employees, this is followed by a line that will contain an 'n' for not married or 'y' for married. The last line contains the number of children.

For interns, the next line will contain the name of the supervisor. This is followed by a line that will contain an 'n' for not married or 'y' for married. The last line contains the number of children.

Here is an example for one file (with comments added that are NOT in the data file):

2             // number of employees in file  
Hourly      // type of employee  
Donald Duck   // name of employee  
20.00      // hourly wage  
40.0        // hours worked  
10  1       // vacation days - earned followed by taken  
y            // married - y is yes, n is no  
5            // number of children  
Intern     // type of employee  
Ronald Weasley   // name of employee  
1000.00   // weekly wage  
Snape      // supervisor  
y            // married  
0           // number of kids

Grading Criteria

10% - Employee class is well designed and has all the required methods

10% - Manager class is well designed and has all the required methods

10% - Hourly class is well designed and has all the required methods

10% - Intern class is well designed and has all the required methods

10% - Temporary class is well designed and has all the required methods

5% - Married interface class is well designed and has all the required methods

5% - Documentation

15% - client source code

15% - exception handling

10% - output and execution correct.

Submitting Your Work

Click the "dropbox" link and submit your work through that portion of the course in D2L.  Make sure you submit \*ALL\* files, and that your name is in every file. You should have the following files:

* employee.java
* manager.java
* hourly.java
* temporary.java
* intern.java
* married.java (interface file)
* client file (the one that contains main)
* exception class if you create one

Hints

1) Start by designing the classes using UML. Include the attributes and methods that you need for each class (don't forget about equals and toString). Then write those classes. Then, test each class separately. Here's a sample of what I used for one class:

Manager m1 = new Manager( );  
m1.setName("Danny");  
m1.setPosition("Manager");  
m1.setDays(2);  
m1.setTaken(0);  
m1.setSalary(52000);  
m1.setMarried(false);  
Manager m2 = new Manager("Martha", "Manager", 520000, 14, 2, true, 2); System.out.println(m1); System.out.println(m2);

2) Then write the code to read from the file, ignoring exceptions at this point (consider only the "good" path). You will need to read in the number of employees and then loop that number of times. Based on employee type, you know the next lines that you have to read. Once you've read everything for a single employee, create the instance and add to the collection (of Employees). So it kind of looks like this:

for (int i = 0; i < count && good; i++)  
{  
            empType = inFile.nextLine(); // read employee type  
            name = inFile.nextLine();  
            salary = inFile.nextDouble();  
                             
            // switch on employee type  
            if (empType.equals("Manager"))  
            {                      
                days = inFile.nextInt();  
                taken = inFile.nextInt();  
                trash = inFile.nextLine();  // get past end of line  
                m = inFile.nextLine();  
                if (m.charAt(0) == 'n')  
                    married = false;  
                else  
                    married = true;  
                kids = inFile.nextInt();  
                trash = inFile.nextLine();  
                e = new Manager(name, empType, salary, days, taken, married, kids);  
                eList.add(e);  
             }

3) It is really helpful to use some debugging print statements for printing to see what you are really reading from the file:

boolean debug = true;

days = inFile.nextInt();  
if (debug) System.out.println("days is " + days);  
taken = inFile.nextInt();  
if (debug) System.out.println("taken is " + taken);  
trash = inFile.nextLine();  // get past end of line  
if (debug) System.out.println("trash is " + trash);

etc.

4) Once you can read in the file and get everything in a collection, then you can loop through and calculate the pay and count the vacation days.

5) Then you can add in the error conditions. The easiest way to do this is to restrict the try/catch blocks to the places where something can go wrong. These include if the system can't find the file, if the file is empty, or if the first line isn't a number. After that, about the only thing that can go wrong is in creating the instance of a class (bad parameters, etc.). I found it easiest to create a new exception type (IllegalEmployeeArgument) and just throw it anytime there was an error within the class. This way you can limit the try/catch block to the creation of the instance and adding it to the collection. Then, even if there is an error in the middle of the file, the system will continue reading the rest of the file. (In the input file, the last two employees have errors.)

Something to remember:

If you read an integer or a double from a file, there is still an "end of line" marker on the line that you haven't processed. If you read a string immediately after, you will just get a blank string (just the end of line marker). So, like in the above example, I just use a "trash" variable to read past that end of line marker to get to the