Assignment 2
CS 3443 Application Programming

Company Bookkeeping

A company employs and pays 4 types of employees (each with its own job code):

> **Hourly**. Each hourly employee has an hourly wage and is paid weekly. If an employee works 40 hours or less in a week, then the employee's pay is the product of the number of hours worked and the hourly wage. If an employee works more than 40 hours in a week, then the employee is paid the hourly wage for the first 40 hours of work and 1.5 times the hourly rate for the hours worked in excess of 40. The code for this job type is "HRLY".
>
> **Salaried**. A salaried employee is paid the same amount each week. The code for this job type is "SLRY".
>
> **Commission**. The pay for employees that work on commission is a percentage of their weekly sales. The code for this job type is "COMM".
>
> **Limited Commission**. A limited commission employee is paid similarly to a commission employee except that a limited commission employee is guaranteed a minimum amount, his or her base pay, but commission pay is limited to no more than twice the base pay. The code for this job is type "LCOM".

The above pay descriptions do not include bonuses. Employees may also earn bonuses that are in addition to the pay the earn as described above. Each week, any bonus that an employee has earned is added to the pay that is calculated by job type.

The company also pays for utilities such as electricity, internet, telephone, and water. The amount of a utility bill is the sum of two parts. The first part is calculated based on the usage of the utility by the company. This part is calculated as the product of the amount of the utility used by the company for the week and the billing rate for the utility. The second part is a base fee that is charged each week regardless of the amount of the utility that is used.

Programming the Bookkeeping

Create a Java program that handles some bookkeeping tasks for the company by defining the following interfaces and classes. Your program must implement the UML class diagram that is at the end of this assignment description and as described below. You must use arrays, but do not use any predefined data structures such as `ArrayList`.

1. The `PayableEntity` interface requires its implementers to define an `amountOwed` method that takes no arguments and returns a `double` value.

2. The `Employee` abstract class implements the `PayableEntity` interface. It represents employees of all types. It has a static `numEmployees` variable to count each Employee instance as it is created. It has non-static `name` and `bonus` instance variables to represent an employee's name and bonus amount. The class has the following methods.

- A constructor that takes the employee's name as an argument and increments the employee count.
- An abstract `calcPrebonusPay` method for calculating an employee's without including any bonus.
- An abstract `getJobCode` method that returns a four-character string representing the employee's job type.
- A static `getNumEmployees` method that returns the number of Employee instances that have been created.
- A `calcTotalPay` method that calls the `calcPrebonusPay` method and adds the employee's bonus to the result.
- An `amountOwed` method that returns the result of calling the `calcTotalPay` method.
- Two getters: `getName` and `getBonus`; and one setter: `setBonus`
- A `toString` method that returns a string containing the instance's name, job code, pre-bonus pay, and total pay formatted as follows. A minimum of 15 characters is dedicated to the employee's name which is left justified. This is followed by a single space and then the employee's 4-character job code. This is followed by a single space and a dollar sign. After the dollar sign a minimum of 8 characters is dedicated to the employee's pre-bonus pay which

includes a decimal point and two digits after the decimal point. Following that is a space, a dollar sign and then at least 8 characters for the employee's total pay including a decimal point and two following digits. Both pre-bonus pay and total pay are right justified. (See Figure 1). Use `String.format` to generate the string. Use only space characters for spacing. Do not include leading or trailing whitespace (space, tab, new line, etc.) in the string.

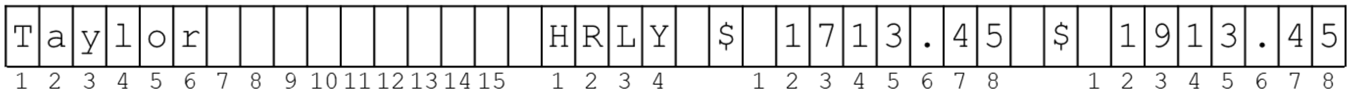


Figure 1. Example of the form of a string returned by the toString method of the Employee class

3. The `HourlyEmployee` class is a subclass of the `Employee` class. It has instance variables for the hourly employee's hourly wage and hours worked. In addition to having concrete definitions for `calcPrebonusPay` and `getJobCode`, this class has the following methods.

- A constructor that takes the hourly employee's name and hourly wage as arguments. The constructor should call the `Employee` constructor so that the hourly employee is counted.
- A `setHoursWorked` method that takes as its only argument the number of hours worked by the employee

4. The `SalariedEmployee` class is a subclass of the `Employee` class. It has instance variables for the salaried employee's weekly pay. In addition to having concrete definitions for the `calcPrebonusPay` and `getJobCode`, this class has the following method.

- A constructor that takes the salaried employee's name and weekly pay as arguments. The constructor should call the Employee constructor so that the salaried employee is counted.

5. The `CommissionEmployee` class is a subclass of the `Employee` class. It has instance variables for the commission employee's commission rate and sales amount. In addition to having concrete definitions for `calcPrebonusPay` and `getJobCode`, this class has the following methods.

- A constructor that takes the hourly employee's name and commission rate as arguments. The constructor should call the `Employee` constructor so that the employee is counted.
- A `setSales` method that takes as its sole argument the amount of sales of the employee.

6. The `LimitedCommissionEmployee` class is a subclass of the `CommissionEmployee` class. It has an instance variable for the employee's base pay. Like the other subclasses of `Employee`, this class has concrete definitions for `calcPrebonusPay` and `getJobCode`, however, its `calcPreBonusPay` method calls the `calcPrebonusPay` method of the `CommissionEmployee` class and then modifies the returned amount so that the employee's pay is at least his/her base pay and no more than twice the base pay.

7. The `Utility` class implements the `PayableEntity` interface. It has instance variables for the utility's name, usage, rate, and base fee. Along with a definition of an `amountOwed` method required by the `PayableEntity` interface, the class has the following methods.

- A constructor that takes the utility's name, rate, and base fee as arguments.
- A `setUsage` method that takes as its only argument the amount of usage of the utility.
- A `toString` method that returns a string containing the instance's name and amount owed formatted as follows. A minimum of 20 characters is dedicated to the utility's name which is left justified. This is followed by a single space and a dollar sign. After the dollar sign a minimum of 10 characters is dedicated to the amount owed which includes a decimal point and two digits after the decimal point. The amount owed is right justified. (See Figure 2). Use `String.format` to generate the string. Use only space characters for spacing. Do not include leading or trailing whitespace (space, tab, new line, etc.) in the string.

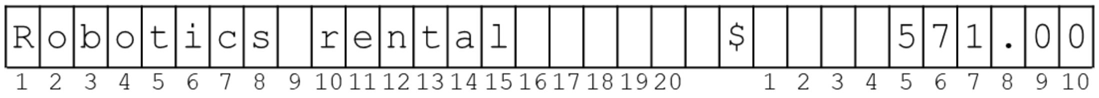


Figure 2. Example of the form of a string returned by the toString method of the Utility class

8. The `Company` class is for representing companies that pay both employees and utilities. It has instance variables for its name, an array of employees and an array of utilities in addition to two instance variables that serve as indices to the employee and utility arrays. The `Company` class has the following methods.

- A constructor that that takes as its arguments, the company name, its maximum number of employees, and its maximum number of utilities.
- A `getName` method that returns the company name.
- An `addEmployee` method that takes an instance of the `Employee` class as its only argument.
- An `addUtility` method that takes an instance of the `Utility` class as its only argument.
- A `createPayrollListing` method that takes no arguments and returns a string describing the pay received by each of its employees. (Use `Employee.toString`) The listing has 4 columns of data. The first column contains employee names, the second column is for employee job codes, the third column contains pre-bonus pay and the fourth column is for total pay. The returned string starts with the title: "<Company Name> Payroll" where <Company Name> is the company's name. After the title is a line of column labels: "Name", "Code", "Pay", and "Total Pay". Align the column labels with the output of `Employee.toString` as in the following example:

  ```
  Acme Corporation Payroll
  Name           Code       Pay Total Pay
  Smith          HRLY $ 1050.00 $ 1050.00
  ```

  Use only space characters for spacing within a line. Except for new line characters, do not include leading or trailing whitespace on each line.

- A `createUtilityListing` method that takes no arguments and returns a string describing the amount owed to each utility. (Use `Utility.toString`) The listing has 2 columns of data. The first column contains utility names, and the second column has amounts owed. The returned string starts with the title: "<Company Name> Utilities" where <Company Name> is the company's name. After the title is a line of column labels: "Name", and "Bill Amount". Align the column labels with the output of `Utility.toString` as in the following example:

  ```
  Acme Corporation Utilities
  Name                Bill Amount
  Electric Company    $    130.00
  ```

  Use only space characters for spacing within a line. Except for new line characters, do not include leading or trailing whitespace on each line.

- A private `calcExpenditures` method that takes an array of instances of `PayableEntity` and returns the total of the amounts owed to each element of the array. If an array location does not refer to an object (it is null), then that location is ignored.
- A `calcTotalExpenditures` method that takes no arguments and returns the total amount that the company owes to its employees and utility providers. This method must use the `calcExpenditures` method to calculate the total amount owed to employees and the total amount owed to utility providers.

Testing

After defining and testing the interface and classes described above individually, download the Lab2.java file and import it into your project. Run Lab2. Its output should be as in Figure 3. Do not modify Lab2.java.

```
Acme Corporation Payroll
Name            Code       Pay Total Pay
Smith           HRLY $ 1050.00 $ 1050.00
Jones           HRLY $ 1200.00 $ 1250.00
Brown           SLRY $ 2000.00 $ 2000.00
Green           COMM $ 2000.00 $ 2000.00
Anderson        LCOM $ 1000.00 $ 1000.00
Fletcher        LCOM $ 2000.00 $ 2000.00
Cooper          LCOM $ 3000.00 $ 3100.00

Acme Corporation Utilities
Name                 Bill Amount
Electric Company     $    130.00
Internet Provider    $    100.00

Acme Corporation total expenditures: 12630.0

Spacely Sprockets Payroll
Name            Code       Pay Total Pay
Davis           LCOM $ 3600.00 $ 5600.00
Wilson          HRLY $ 1286.50 $ 1286.50

Spacely Sprockets Utilities
Name                 Bill Amount
Water Company        $   1000.00

Spacely Sprockets total expenditures: 7886.5

Total employees in all companies: 9
```

Figure 3. Expected output of the Lab2 class

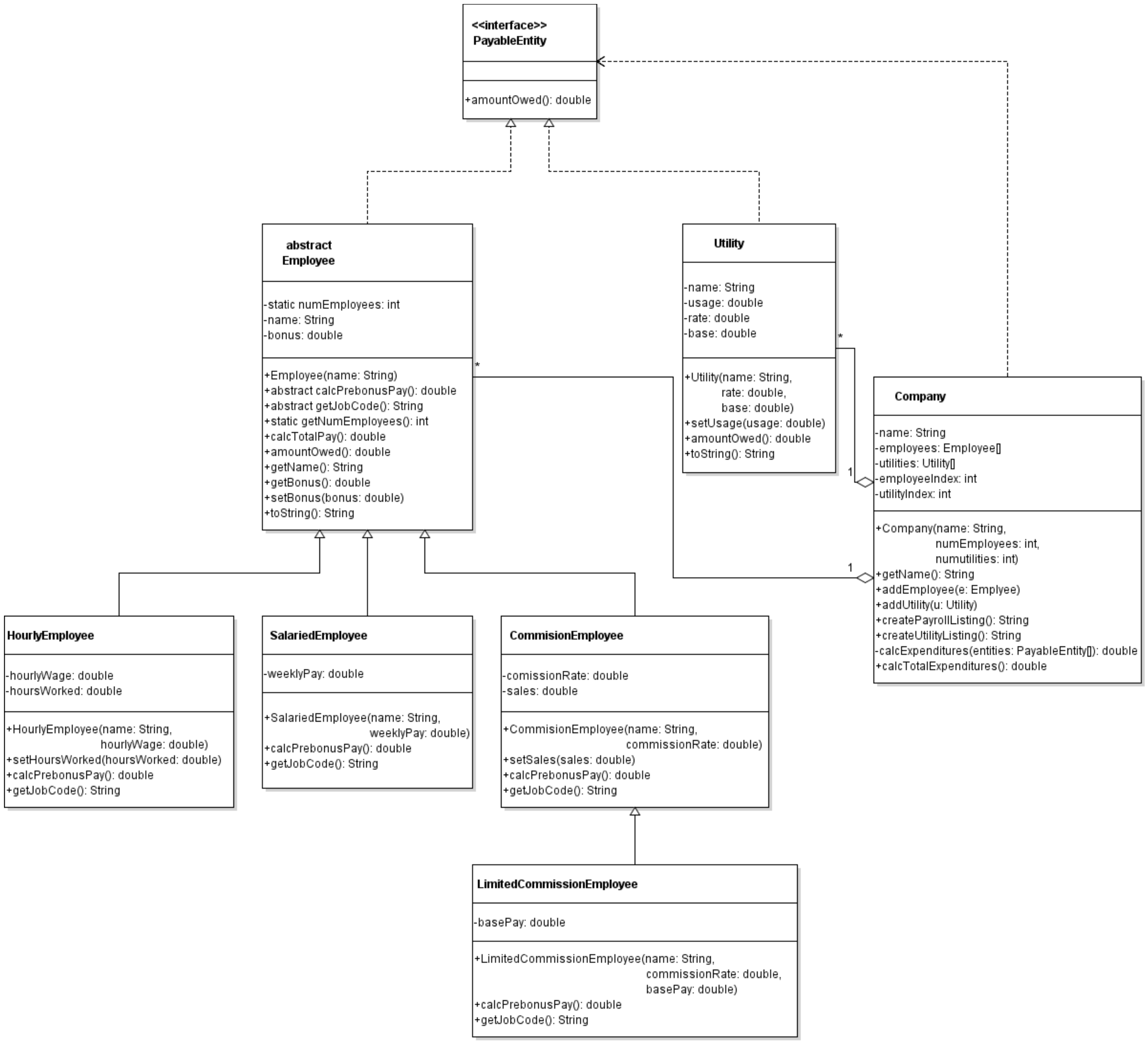
Deliverables

- Create an Eclipse project named a2_abc123 where abc123 is your UTSA ID. Do not create any packages.
- Implement the classes as described. You may define additional methods in the classes.
- Export your project to a zip file named a2_abc123.zip where abc123 is your UTSA student ID.
- Verify that the zip file contains your work.
- Submit your zip file on Canvas
- Verify that you have submitted your zip file correctly.

Rubric

1.	Good programming style: Comments, indentation, names	10%
2.	Correctly created Eclipse project and zip file	10%
3.	Correctly defined `PayableEntity` interface	10%
4.	Correctly defined `Employee` class	10%
5.	Correctly defined `HourlyEmployee` class	10%
6.	Correctly defined `SalariedEmployee` class	10%
7.	Correctly defined `CommissionEmployee` class	10%
8.	Correctly defined `LimitedCommissionEmployee` class	10%
9.	Correctly defined `Utility` class	10%
10.	Correctly defined `Company` class	10%
11.	Use of disallowed classes	-30%
	Total	100%

No credit is given for submissions that do not compile.


<<interface>>
PayableEntity
+amountOwed(): double
abstract
Employee
-static numEmployees: int
-name: String
-bonus: double
+Employee(name: String)
+abstract calcPrebonusPay(): double
+abstract getJobCode(): String
+static getNumEmployees(): int
+calcTotalPay(): double
+amountOwed(): double
+getName(): String
+getBonus(): double
+setBonus(bonus: double)
+toString(): String
*
Utility
-name: String
-usage: double
-rate: double
-base: double
+Utility(name: String,
rate: double,
base: double)
+setUsage(usage: double)
+amountOwed(): double
+toString(): String
*
1
Company
-name: String
-employees: Employee[]
-utilities: Utility[]
-employeeIndex: int
-utilityIndex: int
+Company(name: String,
numEmployees: int,
numutilities: int)
+getName(): String
+addEmployee(e: Emplyee)
+addUtility(u: Utility)
+createPayrollListing(): String
+createUtilityListing(): String
-calcExpenditures(entities: PayableEntity[]): double
+calcTotalExpenditures(): double
1
HourlyEmployee
-hourlyWage: double
-hoursWorked: double
+HourlyEmployee(name: String,
hourlyWage: double)
+setHoursWorked(hoursWorked: double)
+calcPrebonusPay(): double
+getJobCode(): String
SalariedEmployee
-weeklyPay: double
+SalariedEmployee(name: String,
weeklyPay: double)
+calcPrebonusPay(): double
+getJobCode(): String
CommisionEmployee
-comissionRate: double
-sales: double
+CommisionEmployee(name: String,
commissionRate: double)
+setSales(sales: double)
+calcPrebonusPay(): double
+getJobCode(): String
LimitedCommissionEmployee
-basePay: double
+LimitedCommissionEmployee(name: String,
commissionRate: double,
basePay: double)
+calcPrebonusPay(): double
+getJobCode(): String