Version 1.0 : Released March 17th

Assignment 2

Company Management Tool Prototype (Part 2)

Due: not later than Wednesday, March 31th, 2018, 11:59 p.m.

Description:

In this assignment you'll add new features to your existing Company Management Tool application, as described in this document. In the process you will demonstrate your understanding of the following course learning requirements, as listed in the *CST8284—Object Oriented Programming (Java)* course outline:

1. Install and use Eclipse IDE to debug your code (CLR I, VIII)
2. Write java programming code to solve a problem, based on the problem context, including UML diagrams, using object-oriented techniques (CLR II)
3. Use basic data structures, such as ArrayList (CLR III)
4. Implement an inheritance hierarchy involving polymorphism (CLR IV)
5. Prepare program documentation using prescribed program specifiers (CLR IX)
6. Debug program problems using manual methods and computerized tools in an appropriate manner (CLR X)
7. Identify appropriate strategies for solving a problem (CLR XI)

Worth
6%
of your total mark

Assignment 2

Company Management Tool Prototype (Part 2)

I. Create a new project folder

a. In Eclipse, open your existing Assignment1 folder, right click on the `src` folder, and select 'Copy' from the drop down menu. (Alternately, in place of your Assignment 1 code, you can use the solution code provided, available on Brightspace.)

b. Create a new project called CST8284_W19_Assignment2. Right click on the project folder, and select 'Paste' from the drop down menu. This effectively copies all the files in the Assignment1 `src` folder into the new assignment folder.

c. Re-configure the `testcompany` package for JUnit by (1) right-clicking on the package (2) selecting New >> JUnit Test Case (3) in the text box next to Name, add any class name you like (4) allow Eclipse to add JUnit to the build path (5) delete the new class you just added from the `testcompany` package.

II. Add the following new features to the Company Management Tool Prototype application

a) **Use ArrayList in place of arrays**

Convert the `Employee[]` array to an ArrayList of type `Employee`. Note that the access modifier remains private.

You'll need to refactor all the methods that depend on the `Employee[]` array (such as `addEmployee()`) to utilize appropriate ArrayList methods, as addressed in class and covered in the course notes. Also, you can remove any reference to `numberEmployees`, since you can now return `employees.size`() to find out how many employees are in the ArrayList. Similarly, you can remove `MAXNUMEMPLOYEES`, since we no longer need to hardcode the `Employee[]` array length at start-up—the number of employees in the company is now limited only by the number of entries that can be held in memory. To reflect this fact, modify the `isMaximumEmployees()` method so that it always returns `false`. (We'll correct this in Assignment 3, when we add exception handling to our code.)

b) **Add two new methods to the Company class**

In the Company class, add the `findEmployee()` method indicated in the UML diagram (see page 5.) This method is passed the employee number, an `int`, and uses this to find the *first* Employee having that employee number in the `employees` ArrayList.

Additionally, add a `deleteEmployee()` method, which also takes an employee number, but uses it to delete an employee from the `employees` ArrayList. Note that both methods return the Employee just found or deleted.

In the CompanyConsole class, add the two matching private methods indicated in the UML diagram. Here, `deleteEmployee()` prompts the user for the Employee number to delete, invokes `startup.deleteEmployee()`, and prints out the name of the employee just deleted, assuming it was found, otherwise the message "An employee with that number could not be found." The CompanyConsole `findEmployee()` method performs a similar task, but without removing the Employee from the `employees` ArrayList. Here, the Employee returned from `startUp.findEmployee()` is used to output the name of the Employee returned by that method, as described above.

c) **Add subclasses to the Employee class**

First, make the Employee class abstract, and add the following abstract method header:

```
public abstract
        void loadExtraInfo();
```

(Note that if you haven't already added a default constructor to the Employee class, you'll need to do so now.)

Next, add the following three Employee subclasses to your project: Manager, Staff, and Temp.

Add the fields, getters and setters indicated in the UML diagram for each of the three new Employee subclasses. Add a concrete `loadExtraInfo()` method in each of the three subclasses, thereby overriding the abstract method in the superclass, as required. Each `loadExtraInfo()` method should prompt the user to load the features that are new to that subclass, its setters. In other words, `loadExtraInfo()` prompts the user to finish the business of initializing the subclass fields not present in the superclass.

For example, the Manager's `loadExtraInfo()`method should prompt the user to enter the manager's title (e.g. "President", "VP", "Supervisor", etc.). The other two subclasses should have prompts, appropriate to their behaviour, designed to load the features particular to each class. As always, you should use the public setters of the class rather than 'talk' to its private fields directly.

Next, modify the Company's `addEmployee()` method, as indicated in the UML. Start by adding a new parameter at the end of the parameter list for the method. (If you loaded the three default employees indicated in Assignment 1, you'll need to add a new, fifth parameter to their `addEmployee()` calls.) The new value is an integer to indicate which type of employee to add to the `employees` ArrayList, where:

```
MANAGER = 1
STAFF = 2
TEMP = 3
```

according to the named constants indicated in CompanyConsole in the UML diagram.

Inside Company's `addEmployee()` method, add a switch() statement and choose which of the three subclasses to instantiate based on the new parameter. As with Assignment 1, use the first four parameters of `addEmployee()` to initialize your new Manager(), Staff() or Temp() constructor. (Inside each of these you'll need to use `super(...)` to allow the Employee superclass fields to be initialized.)

Where default values need to be passed to a constructor, make *appropriate* choices. The actual values you choose doesn't really matter here, since these will be overwritten when `loadExtraInfo()` is executed, as described below. However, that doesn't mean you should just assign *`null`* or '0' to every unspecified parameter: *make appropriate choices*. The same goes for the no-arg constructors in the subclasses.

As a final change to the Company's `addEmployee()` method, make sure it returns whichever of the three Employee subclasses was instantiated in the method, according to the value of the fifth parameter passed, representing the type of the employee. So if the original call to `addEmployee()` was

```
addEmployee("Ross Chuttle", 23,
     new OurDate(22,2, 2010),
     43269, MANAGER);
```

then the return type will be a new Manager object, correctly instantiated in the Manager() base constructor.

Note: this is a somewhat primitive example of a Factory method, a design pattern that produces new objects upon request. You'll explore this pattern in more detail in CST8288.

Last, in the CompanyConsole's `addEmployee()` method, save the Employee object returned from the call to `startUp.addEmployee()` to a temporary Employee-declared variable.

[Recall: a *declaration* is the part of a definition to the left of the "=" sign. Here, your *declaration* should be of type Employee. That doesn't mean you need to instantiate a new Employee object. If fact, since Employee is now abstract, it *can't* be instantiated. But nothing prevents you from storing the reference value from a concrete subclass object in an abstract superclass Employee variable, and then using its methods. This is an example of *polymorphism*.]

With this Employee variable set equal to the subclass object returned by `startUp.addEmployee()`, call the `loadExtraInfo()` method. The actual method invoke will depend upon the subclass created.

As the final change to the original code from Assignment 1, the CompanyConsole `addEmployee()` method needs to be updated to ask the user of the code which of the three types of employees the new employee actually is. The value returned (which must be a 1, 2, or 3—you'll need to check), will be passed as the final parameter to `startUp.addEmployee()`.

This last step completes the loading of the Employee type selected by the user, first by using the Company's `addEmployee()` method to instantiate the appropriate Employee subclass, and then using that subclass's `loadExtraInfo()` method to finish the initialization particular to each subclass.

d) **Override toString() and equals() in the new subclasses**

Override each subclass `toString()` method so that it appends the new information to the existing (i.e. superclass) `toString()` method.

Similarly, each subclass `equals()` method will need to compare the new features of each subclass (checking *`this`* subclass with the compared *`Object`*'s subclass), while using the superclass `equals()` method to compare the Employee's fields. Note: these two methods are not listed in the UML due to lack of space; they still need to be done.

e) Modify the menu.

Add new options to the menu to reflect these changes. First, you'll need to add the following named constants to the list of already defined in the CompanyConsole class

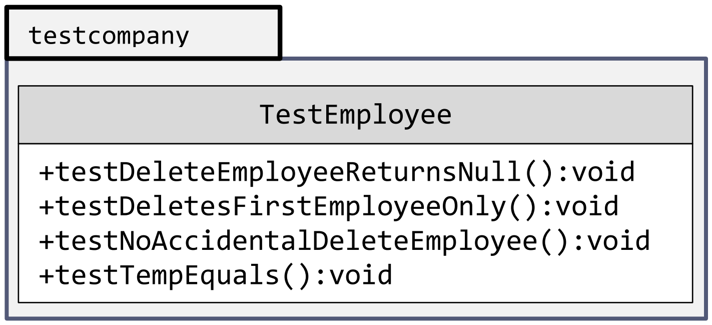
FIND_EMPLOYEE = 4
DELETE_EMPLOYEE = 5

You must use *these constant in a switch() statement, not an if…else if….* (Note: as a rule of thumb, whenever you need three or more if…else if… statements, you should consider using switch…case instead.)

f) **JUnit tests**

First, correct any existing unit tests in the TestEmployee class that may have been affected by the above changes.

Next, add the following four unit tests to TestEmployee


testcompany
TestEmployee
+testDeleteEmployeeReturnsNull():void
+testDeletesFirstEmployeeOnly():void
+testNoAccidentalDeleteEmployee():void
+testTempEquals():void

where:

`testDeleteEmployeeReturnsNull()`

checks that the `deleteEmployee()` method returns `null` when the employee number is not in the `employees` ArrayList;

`testDeletesFirstEmployeeOnly()`

that checks that `deleteEmployee()` deletes only the *first* of two Employee entries having the same employee number;

`testNoAccidentalDeleteEmployee()`

checks that when an employee number is not found, `deleteEmployee()` does not accidentally delete an entry anyway;

`testTempEquals()`

checks that a new Temp employee's `equals()` method works correctly.

[JUnit tests are run during code development to help ensure that code that *was* working has not been adversely affected by recent updates. Unit tests should be built so that they can be run throughout the software development life cycle. Therefore, your tests should be entirely automated; they should never require user input.]

g) **Document your code correctly**

Add documentation to the new classes/methods you've added, and modify the documentation for those methods with new features, functionality, or operations.

Use the Assignment 1 sample code document as your guide. Note that most of the documentation presented by students in that assignment fell far short of any acceptable standard (even in cases where full marks were awarded). In this assignment, your documentation will be held to a higher standard.

In particular, it is *not* enough to write something like

```
// getter for name
```

(Seriously, if that's all there was to it, why even bother?) What would be more appropriate would be a comment such as:

```
/* returns a String
   representing the full name
   (i.e. the first name plus
   the last name as a single
   String, separated by a
   space) of the Employee
   stored in the private name
   field of the Employee class.
*/
```

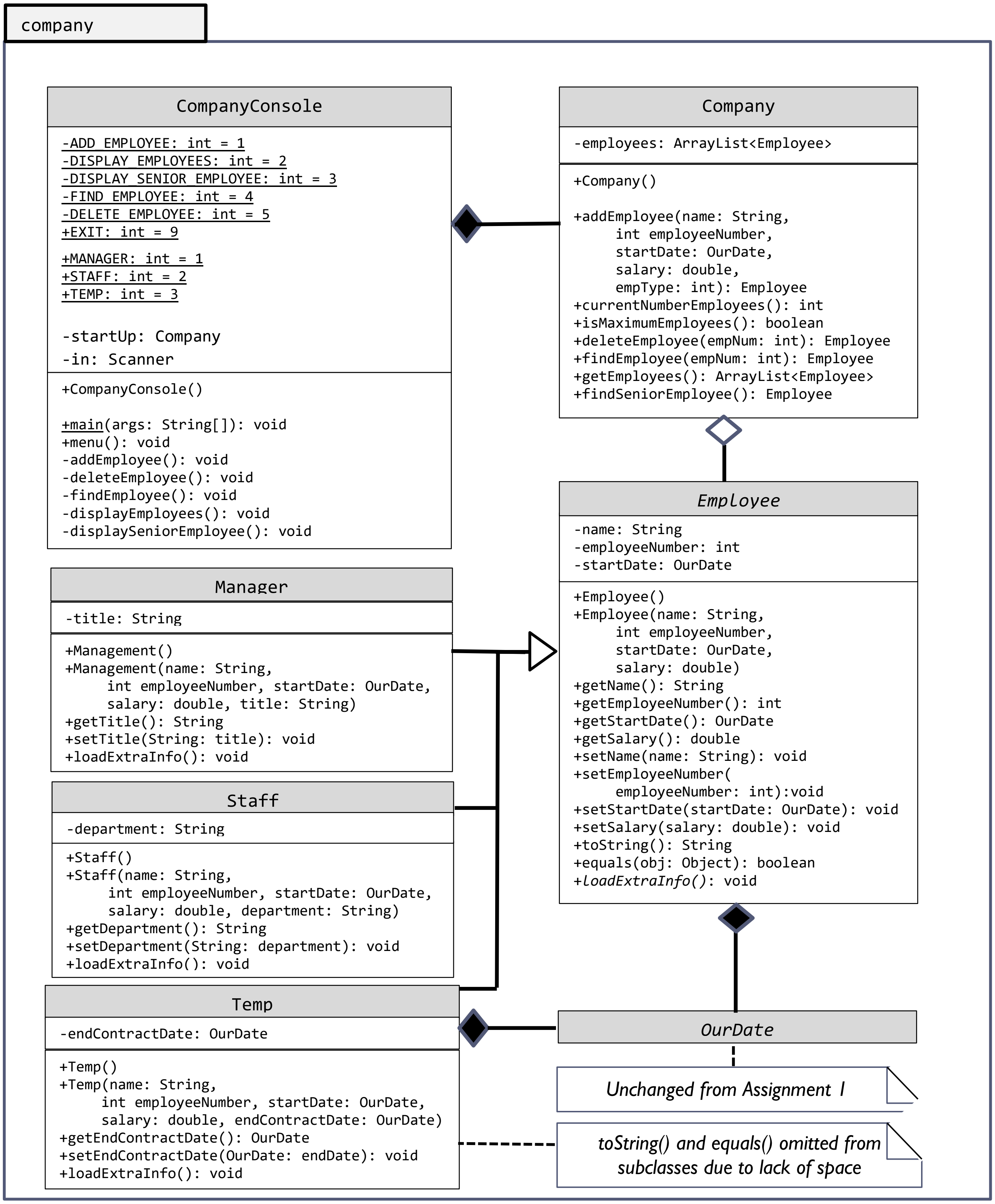
..and you need to do this for each method.

Documentation is useless if it:

1. Only states the obvious
2. Answers 'What' without answering 'Why' and, where appropriate, 'How', along with any appropriate details

For an example of this latter point, see `findSeniorEmployee()` in the sample solution.

If your original documentation was not up to this standard, then you'll need to revise it for this assignment.


company
Unchanged from Assignment 1
toString() and equals() omitted from subclasses due to lack of space

```
CompanyConsole
-ADD_EMPLOYEE: int = 1
-DISPLAY_EMPLOYEES: int = 2
-DISPLAY_SENIOR_EMPLOYEE: int = 3
-FIND_EMPLOYEE: int = 4
-DELETE_EMPLOYEE: int = 5
+EXIT: int = 9

+MANAGER: int = 1
+STAFF: int = 2
+TEMP: int = 3

-startUp: Company
-in: Scanner
---
+CompanyConsole()

+main(args: String[]): void
+menu(): void
-addEmployee(): void
-deleteEmployee(): void
-findEmployee(): void
-displayEmployees(): void
-displaySeniorEmployee(): void
```

```
Company
-employees: ArrayList<Employee>
---
+Company()

+addEmployee(name: String,
     int employeeNumber,
     startDate: OurDate,
     salary: double,
     empType: int): Employee
+currentNumberEmployees(): int
+isMaximumEmployees(): boolean
+deleteEmployee(empNum: int): Employee
+findEmployee(empNum: int): Employee
+getEmployees(): ArrayList<Employee>
+findSeniorEmployee(): Employee
```

```
Employee
-name: String
-employeeNumber: int
-startDate: OurDate
---
+Employee()
+Employee(name: String,
     int employeeNumber,
     startDate: OurDate,
     salary: double)
+getName(): String
+getEmployeeNumber(): int
+getStartDate(): OurDate
+getSalary(): double
+setName(name: String): void
+setEmployeeNumber(
     employeeNumber: int):void
+setStartDate(startDate: OurDate): void
+setSalary(salary: double): void
+toString(): String
+equals(obj: Object): boolean
+loadExtraInfo(): void
```

```
Manager
-title: String
---
+Management()
+Management(name: String,
     int employeeNumber, startDate: OurDate,
     salary: double, title: String)
+getTitle(): String
+setTitle(String: title): void
+loadExtraInfo(): void
```

```
Staff
-department: String
---
+Staff()
+Staff(name: String,
     int employeeNumber, startDate: OurDate,
     salary: double, department: String)
+getDepartment(): String
+setDepartment(String: department): void
+loadExtraInfo(): void
```

```
Temp
-endContractDate: OurDate
---
+Temp()
+Temp(name: String,
     int employeeNumber, startDate: OurDate,
     salary: double, endContractDate: OurDate)
+getEndContractDate(): OurDate
+setEndContractDate(OurDate: endDate): void
+loadExtraInfo(): void
```

```
OurDate
```

III. Notes, Suggestions, and Warnings

a. As with Assignment 1, while you are told how this program is expected to behave, you have a certain amount of freedom to implement the code in any reasonable fashion, provided the program execution is bug-free. However, you would be well-advised to consult the sample code provided for Assignment 1 before proceeding.

b. The UML diagram is not a general guideline for how your code *might* be written; it is a specification for how your code *must* be written if it is to comply with the standards of the organization you are working for. *You cannot add or ignore methods as you please*. All the methods indicated in the UML must be implemented, with the signature indicated, and all of them should function correctly, even if they are not being demonstrated in the sample output.

   Note in particular that uppercase, lowercase, camelCase, access modifiers, static methods, class names, package names, return types etc. must appear in your code according to the UML. If you strongly suspect a mistake has been made in the UML diagram, then you should probably direct your concerns to me ASAP at houtmad@algonquincollege.com so I can correct and update the UML. (Check for any updates before contacting me, since the reported bug may already have been fixed.) Otherwise, treat the UML diagram as an absolute guideline, to be followed to the letter.

c. Before contacting the instructor about problems you are having with your code, be sure to first use debug to isolate the location of the problem. This involves setting a breakpoint in your code to just before the 'last known good' location (as indicated in the red output message that appear in the console), and step until the error is encountered, using the Variables windows to check the state of the fields at various stages. (And if the 'last know good' location is the first line of the program, then that's where you'll set your breakpoint.) Fix the bugs as they appear, run the code again, and repeat until there are no 'red' error messages. Then check for runtime errors by rigorously testing your working code.

   Contact your lab professor if you are stuck, but only after you've made a valid attempt to fix your program using debug.

d. Students are reminded that:

   - You may use the Assignment 1 code as your basis for building Assignment 2, either in whole or in part. You do not need to cite the professor/author by name when using this code.

   - You should not need to use code/concepts that lie outside of the ideas presented in the course notes, labs or hybrids. If you *do* employ concepts in your assignment that are not covered in the course, you *must* cite the source of this information. Failure to do so *will* result in a charge of plagiarism.

   - Students must be able to explain the execution of their code. If they can't, then it is reasonable to question whether they actually wrote the code or not. Partial marks, including a mark of zero, may be awarded if a student is unable to explain the execution of their code, and a charge of plagiarism may be levelled if there is good reason to believe the student did not author the code they submitted.

e. New versions of this document will appear as bugs are uncovered, or as further clarification is required regarding certain questions that arise. Check Brightspace periodically for revisions.

IV.Submission Guidelines

To submit your project, in Eclipse, right click on the project name (the project, NOT the package name), select Export... In the Export dialog, select General >> Archive File, then click the Next button. In the Archive File menu, (making sure your Assignment1 project is selected, along with all its components, at right), in the text box, name your zip file according to the naming convention indicated below.

Save this zip file to any location on a drive where you can find it, and select Finish. This is the zip file you should upload.

The correct format for labelling your zipped project is:

`Lastname_Firstname_Assignment2_SecXXX.zip`

where Lastname and Firstname are, of course, *your* last name and first name, and XXX is your lab section number. (Note the underscores, the use of upper and lower case, and no spaces; this format must be followed exactly when you name your archived file.)
Lab times and section numbers are:

Tuesday 8:00 a.m. – 10:00 a.m.: Sec 301
Tuesday 11:00 a.m. – 1:00 p.m.: Sec. 312
Tuesday 1:00 p.m. – 3:00 p.m.: Sec. 303
Tuesday 3:00 p.m. – 5:00 p.m.: Sec. 313
Tuesday 6:00 p.m. – 8:00 p.m.: Sec. 314
Friday 10:00 a.m. – 12:00 a.m.: Sec. 302
Friday 2:00 p.m. – 4:00 p.m.: Sec. 304
Friday 4:00 p.m. – 6:00 p.m.: Sec. 311

Failure to label your submission according to these rules will cost you marks.

Note:

- Since there are always some students who upload empty zip files with their submission, it is highly advisable that before you upload your final submission to Brightspace, you copy it to another location first and unzip it, making sure it contains the correct information. If you've followed the instructions in this document correctly, when you open your zip, you should see the CST8284_W19_Assignment2 folder, and inside that, appropriate subfolders, including `src`, which in turn should contain the `company` package and the `testcompany` package, as separate folders. If you don't see all of these as described, then you've made a mistake, and you need to follow the instructions at left again, otherwise marks will be deducted.
- You can upload as many attempts as you'd like, but only the final attempt is marked: all other assignments are ignored.

V. Marking

Marks will be awarded according to the following marking scheme. Note that marks may be deducted at the discretion of the marker for code that is excessively sloppy, repetitive, poorly documented, does not use existing code correctly, does not follow standard conventions established in earlier courses, etc. In other words, you will lose marks for doing anything you should have learned in your Level I Java course. At this level, you should be writing fairly clean, well-written code, even if this isn't explicitly stated in the marking scheme.

Item	Marks
Submission guidelines followed, according to Section IV of this document	**3**
Documentation is complete. This includes revisions to the documentation submitted with Assignment 1, according to the guidelines given in Section III(g)	**4**
ArrayList used in place of array throughout the document, according to Section III (a)	**3**
Two new methods have been added to the Company class, and they function as required according to Section III (b)	**2**
The project contains the three subclasses indicated in Section III (c), along with all the additional methods indicated in the UML, including appropriate getters and setters, and overridden loadExtraInfo() methods, appropriate to the class, along with correctly overridden toString() and equals() methods as per Section III (d)	**5**
Company's addEmployee() method of Section III (c) returns the correct type of Employee subclass, according to the type requested by the client of the code.	**5**
The menu has been modified in CompanyConsole to reflect the changes to the code, in particular the next options available to the client, as per Section III (e)	**1**
All previous JUnit tests work, and the four new added tests perform the specific task they are designed to test, according to Section III (f)	**4**
The code follows the UML document (excepting revisions to code released prior to the due date for this assignment).	**3**
TOTAL :	**30**

Sample Output:

To be released in Version 1.1, due out shortly

Version Information:

Corrections to this document will be posted as required, so check Brightspace periodically for notification of corrections. Version-specific changes to this document will be listed below