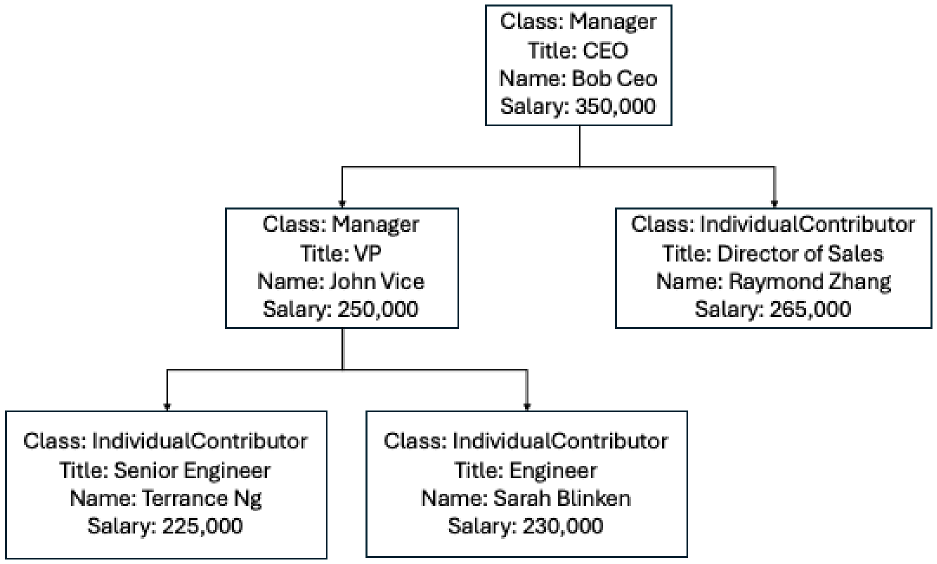
Introduction to Java CS9053
Fall 2025
Midterm Coding Section
50 points
Oct 9th, 2025
Due: Oct. 17th 2025 11:59pm

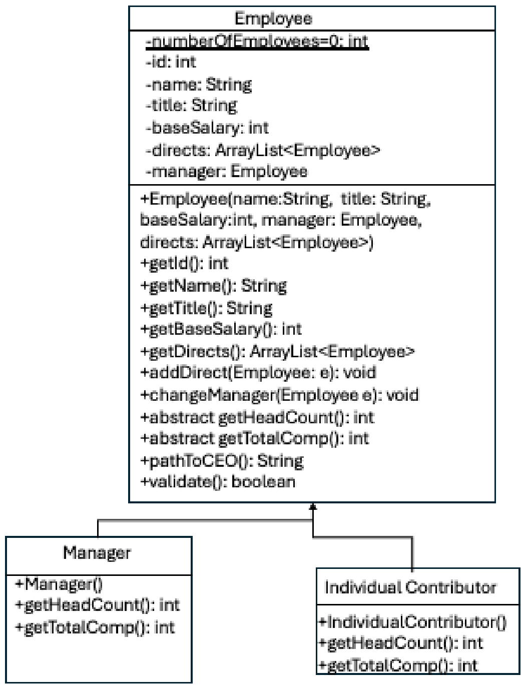
Part I:

This is an org chart:



As you can see, there are managers who have people who report to them (sometimes other managers), and individual contributors who report to managers. You are going to build an object hierarchy that implements these and then build an org.

The inheritance diagram is not that complicated:



I have left out the arguments for the constructors for Manager and IndividualContributor, but you should be able to figure those out. As you can see, the ids are auto-generated.

Important points:

- IndividualContributor objects have no direct reports, but Managers do
- The CEO has a "manager" value of null
- If an Employee has a "Manager", then that Employee will appear in the "directs" ArrayList of that Manager
- "getHeadCount()" should return the total number of people, including the employee, in their subtree. An individualContributor has a headcount of "1". "John Vice" has a headcount of "3". "Bob Ceo" has a headcount of "5"
- getTotalComp() should return the total compensation of everyone in the Employee's subtree, including the Employee. An IndividualContributor has a "total comp" of his base salary. A Manager has a "total comp" of his base salary plus the total comp values of his direct reports.
- "pathToCeo" should output a String that shows the name & id of each employee from the Employee it's called on to the CEO. If Sarah Blinken has a reference variable sb, then sb.pathToCeo() returns: "Sarah Blinken – John Vice – Bob Ceo"
- validate() should find the root of the tree, confirm both that "manager" appears in "reports" of the manager and that there are no cycles in the tree

Create the objects in the org chart. You can start with the IndividualContributor objects, and then pass an arraylist of those IndividualContributor objects to the constructor of their Manager, where you will set the manager of everyone in that ArrayList to the Manager you have created.

How you confirm that the Employee's Manager contains that Employee in the Manager's "directs" ArrayList is up to you.

The ArrayList you pass in to the Manager constructor should be COPIED to a NEW ArrayList that is created in the constructor that you assign to "directs". getDirects()

Part II

You're going to read in a file of Movies and their ratings and analyze them. Each line of the file contains these fields:

Title,Year,Genre,Rating

An excerpt looks like this:

```
Inception,2010,Action,8.8
Toy Story,1995,Animation,8.3
Avatar,2009,Action,7.8
Interstellar,2014,Science Fiction,8.6
The Godfather,1972,Crime,9.2
```

You're going to read these into an **ArrayList** of **Movie** objects.

A **Movie** class contains a title, year, genre, and rating (just like the file!). Create a class with a constructor and getters and setters and toString that will store this information.

Then you will create a **MovieAnalayzer** class that contains the main method and static methods **readMovies(String filename)** which returns an **ArrayList** of movies. It will call **createMovie(String inline)** which takes a line from the csv file and returns a **Movie** object. Some lines may be malformed. You have to catch any exceptions that occur while parsing the file. That method should raise an **unchecked** exception called "MovieLineParseException" if the input line is unparseable. If for some reason a line is malformed or unparseable, you should keep going.

Hint: the String.split() method is helpful here.

Finally, there is a static method called **analyze(ArrayList<Movies> movieList)**. This method should print out the ArrayList in descending order according to Rating and then again in Ascending order according to genre. For each of these, **you should sort the array list both times using a lambda function in the sort method.**

You should do this for the longer file **movies.csv**. There is another file **movies_short.csv** that you can use to experiment with since it is shorter.