# Homework 05 – The Office Fire

Authors: Ariel, Tomas, Sooraj, David, Allan
Topics: ArrayList, Generics, Interfaces, Comparable, Big-O

## Problem Description

Please make sure to read all parts of this document carefully.

Dwight is the office fire marshal. As such, he needs to make sure that the office follows the safety code, which means that they need to have a certain number of fire extinguishers. Furthermore, in case of fire Dwight needs to make sure that everyone made it out safely. For that, he asks you to develop some algorithms that will help him accomplish his task.

## Solution Description

You need to turn in 4 classes: Office.java, Department.java, Employee.java, and Dwight.java

### *Department.java*

**Variable(s):**

All variables must not be allowed to be directly modified outside the class in which they are declared, unless stated otherwise:

- `String name` – the name of the department
- `boolean hasFireExtinguisher` – indicates if the department has a fire extinguisher.

**Constructor(s):**

- A constructor that takes in the name and if the department has a fire extinguisher.

If necessary, add any getters and setters for the variables in this class.

### *Employee.java*

This file represents an employee. We need a way to compare employees, as such, this class will need to implement the Comparable interface with generics.

**Variable(s):**

All variables must not be allowed to be directly modified outside the class in which they are declared, unless stated otherwise:

- `String name` – the name of the employee
- `int height` – the height of the employee

**Constructor(s):**

- A constructor that takes in `name` and `height` of the employee

**Methods**:

All methods in this class should be public, unless stated otherwise:

- `toString()`
    - This method should properly override `Object`'s `toString()` method
    - "My name is `{name}` and I am `{height}` inches tall"
- `equals()`
    - This method should properly override `Object`'s `equals()` method
    - If two employees have the same `name` and `height` they should be equal.
- `compareTo()`
    - This method will implement the compareTo method from comparable.
    - A employee is greater than another employee if their height is bigger than the other employee.
    - Make sure we are **only** allowed to compare employees with employees. **Think about how the class should implement comparable.**

If necessary, add any getters and setters for the variables in this class.

## *Office.java*

### Variable(s):

All variables must not be allowed to be directly modified outside the class in which they are declared, unless stated otherwise.

- `String name` – the name of the office
- `ArrayList<Department> departments` – An ArrayList with the departments available
- `ArrayList<Employee> employees` – An ArrayList with the employees of the office. Note that this will always be sorted by descending height and no two employees are the same height.

### Constructor(s):

- A constructor that takes in a `name`. This constructor should create an empty list of `departments` and `employees`.

### Method(s):

All methods in this class should be public, unless stated otherwise:

- `addDepartment()`:
    - A method that takes in a department and adds it to the end of the list of departments. This method should not return anything. This method should run in O(1) time complexity.
- `addEmployee()`:
    - A method that takes in an Employee and adds it to the employees. Assume the employee is not in the list. Note that the employee list should always be sorted by descending height and that no two employees are the same height. This

method should not return anything. This method should run in O(n) time complexity.

- `removeEmployee()`:
  - A method that takes in an Employee and removes it from the employees if it is in the list of employees. Check for value equality. Should return the old employee or null if not found.

If necessary, add any getters and setters for the variables in this class.

## *Dwight.java*

This class describes Dwight Schrute, one of the most special employees in the office. As a fire marshal, he has the responsibility to make sure the office is safe at every moment. Furthermore, in case of fire he needs to make sure everyone made it out of the office. This class should inherit from the Employee class.

**Variables:**

- No extra instance variables

**Constructor(s):**

- A constructor that takes in no values and assigns his name to Dwight Schrute and height to 74.

**Methods:**

All methods in this class should be public, unless stated otherwise:

- `toString()`
  - This method should properly override Employee's `toString()` method
  - It should print the following:
    "My name is Dwight and I am the fire marshal"
- `checkOffice()`
  - This method should take an office object and return a boolean that tells whether at least 50% of the departments have a fire extinguisher. This method should run in O(n) time.
- `findEmployee()`
  - This method should take in an employee and an array list of employees, which you should assume is sorted.
  - The method should check whether the employee is in the list of employees. This method should have an O(log(n)) time complexity. (Note that the list of employees will be sorted in descending order by height and no two employees are the same height).
  - Hint: think about which searching algorithm runs in this time complexity.
  - Return a boolean that tells whether the employee is found in the array list.
- `doRecount()`
  - This method should take an ArrayList with the employees and an office object. It should then return a new ArrayList with the employees that are in the office but are not in the in the employees ArrayList passed in. This method should have an O(n log(n)) time complexity.
  - Hint: Think about how you can use the findEmployee() method.

If necessary, add any getters and setters for the variables in this class.

## Testing

Reuse your code when possible. These tests and the ones on Gradescope are by no means comprehensive, so be sure to create your own!

## Checkstyle

You must run checkstyle on your submission (To learn more about Checkstyle, check out cs1331-style-guide.pdf under CheckStyle Resources in the Modules section of Canvas.) **The Checkstyle cap for this assignment is 30 points.** This means there is a maximum point deduction of 30. If you don't have Checkstyle yet, download it from Canvas -> Modules -> CheckStyle Resources -> checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the checkstyle cap mentioned in the Rubric section). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

For additional help with Checkstyle see the CS 1331 Style Guide.

## Turn-In Procedure

### *Submission*

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `Office.java`
- `Department.java`
- `Employee.java`
- `Dwight.java`

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any**

**autograder test are provided as a courtesy to help “sanity check” your work and you may not see all the test cases used to grade your work.** You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via Piazza for clarification.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

## *Gradescope Autograder*

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g. non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

## Allowed Imports

To prevent trivialization of the assignment, you are only allowed to import java.util.ArrayList.

## Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`
- `System.arraycopy`

## Collaboration

Only discussion of the Homework (HW) at a conceptual high level is allowed. You can discuss course concepts and HW assignments broadly, that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

## Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- **Check on Ed Discussion for a note containing all official clarifications and sample outputs**

It is expected that everyone will follow the Student-Faculty Expectations document, and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. No inappropriate language is to be used, and any assignment, deemed by the professor, to contain inappropriate, offensive language or threats will get a zero. You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.