Programming Exercise 05 - equals, toString, hashCode

Problem Description

This assignment will give you practice writing good `equals`, `toString`, and `hashCode` methods. You will also get practice writing a class that extends from another class.

Solution Description

You will be given three classes: `Building`, `Company`, and `City`. Each class will be missing one of the essential elements of a good class, i.e. an `equals` method, a `toString` method, or a `hashCode` method. Your job will be to fill in the missing methods using the techniques taught in class. If you are confused on any of these methods, look at how we have implemented it in one of the provided methods. You will also need to implement a `SmallBusiness` class that is a subclass of `Company`.

Building.java

In this class you will be responsible for writing an `equals` method. The toString and hashCode methods have been provided for you. This class has a variety of fields, and you should use your knowlege of `hashCode`'s contract to figure out which fields must be included in your equals method.

Company.java

In this class you will be responsible for writing another `equals` method and a `toString` method. Again use your knowlege of `hashCode`'s contract to determine which fields should be included in the equals method. Your `toString` method should **return** a `String` in the following format: "The {name} company has {numEmployees} employees and their headquarters is in the {headQuarter's name} building." *Note: you should not include the curly braces in your returned `String` and all words surrounded with curly braces in the example `String` should be replaced with the appropriate variables.*

City.java

In this class you will be provided an equals method and must implement the `hashCode` method. Your `hashCode` implementation must follow the appropriate contract for `hashCode` and must be a "good" implementation (i.e. do not just return a constant value, even though that is technically legal by the `hashCode` contract).

SmallBusiness.java

This class's behavior is identical to that of `Company` with one exception. This class should not allow the `numEmployees` field to become larger than 25. If a user tries to set it higher than 25 (Think what are all the ways that a user could set `numEmployees`) then the field should just become 25.

Allowed Imports

To prevent trivialization of the assignment, you can not import or use anything except the files we provide and java.lang.

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

Collaboration

Collaboration Statement

To ensure that you acknowledge collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Recall that comments are special lines in Java that begin with `//`.

Turn-In Procedure

Submission

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

- Building.java
- Company.java
- City.java
- SmallBusiness.java

Make sure you see the message stating "PE05 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

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 last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds 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:

1. Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2. 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.

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 Piazza for all official clarifications