Homework 01 – House Hunting!

Authors: Lucas Morais Henrique, Ishuma Lahoti, Brandon Noll, Sreya Yadlapalli, Julia Zhu, Allan Nguyen
Topics: Classes and objects, constructors, public and private visibility, getters and setters

Problem Description

Please make sure to read all parts of this document carefully.
As you learn more about Object-Oriented Programming and Java, you decide to apply your skills to help in your search for a new house. To do this, you will need to create the classes `House.java` and `Neighborhood.java` to keep yourself organized during this search.

Solution Description

Create files `House.java` and `Neighborhood.java` that will be used to store houses you've found while surfing the web. You will be creating several fields and methods for each file. Based on the description given for each variable and method, you will have to decide whether the variables/method should be static, and whether it should be private or public. To make these decisions, you should carefully follow the guidelines on these keywords as taught in lecture. In some cases, your program will still function with an incorrect keyword.

Hint: A lot of the code you will write for this assignment can be reused. Try to think of what keywords you can use that will help you!

House.java

This file defines a House object. 🏠

Variables:

All variables must not be allowed to be directly modified outside the class in which they are declared, unless otherwise stated in the description of the variable. The `House` class must have these variables:

- `address` - the address of the House, which can be made of any combination of characters
- `yearBuilt` - the year the House was built as an integer (note: this value cannot be negative)
- `numBaths` – the number of bathrooms of the House as a double
  (note: only nonnegative values divisible by 0.5 are allowed here)
- `isRenovated` - the renovation status of the House as a boolean

Constructor(s):

- A constructor that takes in the `address`, `yearBuilt`, `numBaths`, and `isRenovated`.
- A constructor that takes in the `address` and `numBaths`. In this case, `yearBuilt` should be assumed to be 1984 and `isRenovated` should be assumed to be false.
- A constructor that takes in no arguments. In this case, `yearBuilt` should be assumed to be 1984, `isRenovated` should be assumed to be false, `address` should be assumed to be "North Ave NW, Atlanta, GA 30332" and `numBaths` should be assumed to be 1.

- Assume that inputs to the constructor are valid inputs.
- You must minimize the amount of code written and maximize code reuse when writing these constructors. Hint: what keyword have we learned that allows you to call another constructor?

Methods:

All methods must have the proper visibility to be used where it is specified they are used.

- Getters and setters for each of the instance variables. Do not forget about the conditions stated for `yearBuilt` and `numBaths`! If the inputs are not valid, do not change the value.
- Optional: a helper method to determine whether a number of baths is valid and any other helper methods you may think of. However, remember these methods should NOT be able to be called from outside of this class.

Neighborhood.java

This file defines a Neighborhood object. This class will have some instance variables and functionality designed to store House objects.

Variables:

All variables must not be allowed to be directly modified outside the class in which they are declared, unless otherwise stated in the description of the variable.
The `Neighborhood` class must have these variables:

- `numHouses` - a value that represents the number of Houses in the neighborhood, represented by an integer
- `houses` - a value that represents the Houses stored in the neighborhood, represented by an array

Constructor(s):

- A constructor that takes in a `House` array. Make sure to initialize `numHouses` field to the correct value based on the house array parameter.
- A constructor that takes in no parameter but initializes `numHouses` to 0 and houses to an array of size 5 with no `House` objects.
- You must minimize the amount of code written and maximize code reuse when writing these constructors. Hint: what keyword have we learned that allows you to call another constructor?

Methods:

All methods must have the proper visibility to be used where it is specified they are used.

- `displayNewHouses`
    - Given a year, display all Houses that were built during that year or later.
    - Print each House on a new line in the format "House located at {address} was built in {`year`}, has {`numBaths`} number of bathrooms, and {has/has not} been renovated recently."
    - {has/has not} depends on the value of `isRenovated`
- `addHouse`

    - Given an index and a `House` parameters, add that House to the given index in the houses array and return the House that was previously stored at that index. Remember to update `numHouses` if you are increasing the number of houses stored in the array.
    - If that index was already occupied by a House, do not update `numHouses`.
    - If the index was invalid, the input House was null, or there was no house previously stored at that index, simply return null.
- `removeHouse`
    - Given an index, remove a House from that index in the houses array and return the House that was previously there. Remember to update `numHouses`.
    - If that index was not occupied by a House, return null and do not update `numHouses`.
    - If the index was invalid, simply return `null`.
- `isFull`
    - If every spot in the neighborhood is occupied by a House, return true. Otherwise, return false.
- Getters and Setters as necessary.
- Optional: any helper methods you may need. However, remember these methods should NOT be able to be called from outside of this class.

Testing

HouseHunting.java

This Java file is a driver, meaning it will contain and run House and Neighborhood objects and "drive" their values according to a simulated set of actions. Use this to test your code. You will not be turning this into us. Here are some tips to help you get started:

- Create a few Houses
- Create a Neighborhood
- Add your Houses to the Neighborhood
- Display all Houses in that Neighborhood built after a year you choose
- Remove a few Houses
- Display all Houses in that Neighborhood built after the same year you chose
- Try to change instance variables of Houses into "illegal" values

Reuse your code when possible. Certain methods can be reused using certain keywords. 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 10 points.** This means there is a maximum point deduction of 10. 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!

Turn-In Procedure

Submission

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

- `House.java`
- `Neighborhood.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 not allowed to import any classes or packages.

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.