Assignment 3

COMP 250 Fall 2022

posted: Friday, Nov. 18, 2022
due: Monday, Dec. 5, 2022 at 23:59

General Instructions

- **Submission instructions**
  - Late assignments will be accepted up to 2 days late and will be penalized by 10 points per day. Note that submitting one minute late is the same as submitting 23 hours late. We will deduct 10 points for any student who has to resubmit after the due date (i.e. late) irrespective of the reason, be it wrong file submitted, wrong file format was submitted or any other reason. This policy will hold regardless of whether or not the student can provide proof that the assignment was indeed "done" on time.
  - Don't worry if you realize that you made a mistake after you submitted : you can submit multiple times but only the latest submission will be graded. We encourage you to submit a first version a few days before the deadline (computer crashes do happen and Ed may be overloaded during rush hours).
  - This is the file you should be submitting on Ed:
    - `CatCafe.java`

    Do not submit any other files, especially .class files. Any deviation from these requirements may lead to lost marks
- Please note that the class you submit should be part of a package called `assignment3`.
- **Do not change any of the starter code that is given to you. Add code only where instructed, namely in the "ADD YOUR CODE HERE" block.** You may add helper methods to the `CatCafe` class (and in fact you are highly encouraged to do so), but you are not allowed to modify any other class.
- The assignment shall be graded automatically. Requests to evaluate the assignment manually shall not be entertained, so please make sure that you follow the instruction closely or your code may fail to pass the automatic tests. Note that for this assignment, you are NOT allowed to import any class that is not already imported in the starter code. **Any failure to comply with these rules will give you an automatic 0.**
- Whenever you submit your files to Ed, you will see the results of some exposed tests. These tests are a mini version of the tests we will be using to grade your work. If your code fails those tests, it means that there is a mistake somewhere. Even if your code passes those tests, it may still contain some errors. We will test your code on a much more challenging set of

examples. We highly encourage you to test your code thoroughly before submitting your final version.

- In a week we will share with you a `Minitester` class that you can run to test if your methods are correct. This class is equivalent to the exposed tests on Ed. Please note that these tests are only a subset of what we will be running on your submissions. We encourage you modify and expand these classes. We encourage you modify and expand this class. You are welcome to share your tester code with other students on Ed. Try to identify tricky cases. Do **not** hand in your tester code.
- You will automatically get 0 if your code does not compile.
- Failure to comply with any of these rules will be penalized. If anything is unclear, it is up to you to clarify it by asking either directly a TA during office hours, or on the discussion board on Ed.

Learning Objectives

In this assignment we will bring together binary search trees and max heaps. Working on this problem will allow you to better understand how to manipulate trees and how to use recursion to exploit their recursive structure.

A Crowded Cat Cafe Chain

You have just been named CEO of a chain of Cat Cafes. A cat cafe is a coffee shop in which cats are roaming and interact with customers. At the time of your promotion, you are informed that the chain does not have any repository of the cats, and maintains no information about their seniority. You are extremely upset upon hearing this as you understand those cats are your most important staff members and, like any other employees, deserve salary and working conditions that improve with seniority.

You immediately call your best friend, who recently told you they were a computer science expert. You expect them to build you a state-of-the-art database for all your cats. Upon hearing your request, they seem to hesitate a bit and, after a moment of wavering, confess that they are only taking their second computer science course and cannot yet build a professional database. However, they recently learned about binary search trees and how they can be leveraged to efficiently access data. They explain they have also heard about heaps and they'd like to combine the two ideas together in an attempt to utilize trees to their full potential. They email you a quick draft and you start getting excited about it (after all, cats do love trees), until they specify they cannot help you because they are currently busy with their new job, which they then begin describing. It sounds pretty cool until they try to convince you to invest in their "company" by buying a bunch of knives to re-sell. You try to tell them it sounds like a pyramid scheme but they start yelling something about those only ever occurring in Ancient Egypt, and they hang up on you. You try to call them back but it appears they blocked your number.

It looks like you are going to have to build your tree on your own. You have a second look at their email to see what you have to work with, and what is left to do.

The database of all cats is represented by a binary tree `CatCafe`. This tree has a single field `root` which contains a reference to the root node. The nodes are defined by a nested class `CatNode` (the class is defined `public` for testing purposed). Each `CatNode` is associated with data stored in a `Cat` object (code provided).

You manage your business month to month, so for simplicity, since the company began operating on January 1st, 2000, you are identifying each month of operation with a simple integer associated with the number of months elapsed since then. Thus, January 12, 2000 was in month 0. September 25, 2015 was in month 189, and November 2022 is month 275. You keep track of the month in which each cat was hired by the Cafe in each cat's file. To have an easier time searching through the files, these are kept in order based on the cats' seniority. These cats are truly your most valuable asset, and regular grooming appointments are a must to keep them healthy and happy. The company provides regular trips to the cat salon and covers all the expenses. To schedule the trips and help you keeping your budget on track, you decided to store all the relevant information on each of these

cats' files.

There's no denying it, fluffy cats are gorgeous, they can truly steal one's heart. You were once a loyal customer of the cafe chain yourself, and nothing says 'cozy and laidback coffee shop' more than a fluffy furry friend curled up on a rocking chair by the window. As the new CEO, you have decided to create an hall of fame to honour the cats which were blessed with an exceptional fur. This recognition is very close to your heart, hence every time a new cat is hired by the chain the average thickness of the cat's fur in millimeters (measured professionally) is measured and also kept on file.

Note that, due to budget constraints, you are never able to hire more than one cat per month (i.e. you can assume that there will not be cats in the cafe with the same seniority).

For this assignment (as for any coding project you decide to undertake) it is extremely important that you test your code as you write it using small tests. **Do NOT start testing your code only after you are done writing the entire assignment. It will be extremely hard to debug your program and you should not expect us (instructor, TAs, and mentors) to be able to fix your code just by looking at an error for few minutes during office hours.** If you need help with a bug you need to make sure to put in the time necessary to work on it so that you'll be able to identify a very specific reason and place in your code where the error originates (note that I am not referring to the line displayed by the stack trace!).

The Cat class

To represent a cat, we have provided you with a class called `Cat`. This class contains the following fields:

- A name for the cat (`String`)
- The month in which the cat was hired (`int`)
- The thickness of its fur in millimeters (`int`)
- The number of days until their next grooming appointment (`int`)
- The expected cost (in dollars) of their next grooming appointment (`double`)

Note that this class implements `Comparable` and several `public` methods are available to you. Please note that **you must not modify this class**.

The Cafe

A template for the `CatCafe` class has been provided to you. This class contains a `CatNode` inner class, and a field `root` which contains a reference to the root node of the tree representing the cafe. This class also implements the interface `Iterable` and an inner class called `CatCafeIterator` is also included. Inside the `CatCafe` class you will notice several `public` methods that have already been implemented. These are all non-static methods which will be called on objects of type `CatCafe`. Most of these methods call methods from the `CatNode` class that you will need to implement.

The tree we are using to represent a cat cafe is a tree where we store elements of type `Cat`. It is a binary search tree when we look at the cats' seniority, and a max heap when we look at the cats' fur thickness.

The `CatNode` inner class is used to represent a node in the tree and it contains the following fields:

- `Cat catEmployee`: a reference to an object of type `Cat`. This is the element/key stored in the node. It contains information regarding a cat employed at the cafe.
- `CatNode junior`: this node is the root of the left subtree which stores information about cats that are more junior (i.e. hired later) than the cat stored in the current node.
- `CatNode senior`: this node is the root of the right subtree which stores information about cats that are more senior (i.e. hired earlier) than the cat stored in the current node.
- `CatNode parent`: this node is the parent node of the current node. Hence it stores information about a cat whose fur is thicker than the one of the cat stored in the current node (and he has therefore higher chances of getting in that hall of fame of yours ;)).

For this assignment you need to implement the following methods/classes. Please note that your code will be tested on time efficiency, so make sure to implement the methods exploiting the structure/properties of the tree and using what we learned in class. Note also that you might want to implement the following methods in a different order than the one in which the tasks are listed. For this reason, we would highly suggest to first read through the entire list before jumping into your IDEs.

- `CatCafe.CatCafe()` [8 points]
  This constructor takes as input a `CatCafe` object `cafe` and creates a *shallow* copy of it. The new `CatCafe` object is a tree storing exactly the same objects of type `Cat`, but with different `CatNode`s.

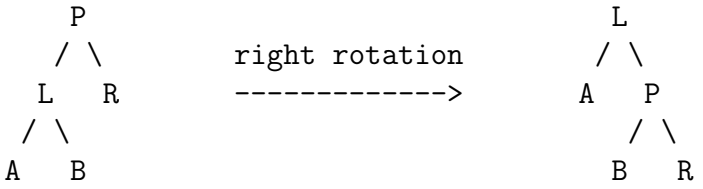
- `CatNode.hire()` [20 points]
  This method takes as input a `Cat` object `c` and adds it to the tree rooted at the `CatNode` on which the method was called. The method returns the root to the new tree that now contains `c`.

  As explained above the tree we are trying to implement is a binary search tree when we look at the cats' seniority, and a max heap when we look at the cats' fur thickness. To preserve the tree's properties, adding a new cat `c` to the tree has to be done in two steps:

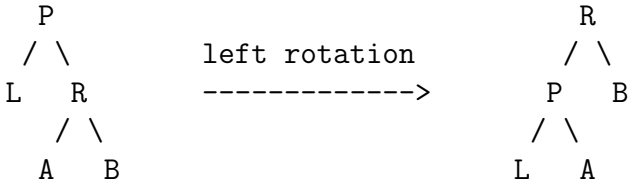
  1. Add a new node to the tree in the leaf position where binary search determines a node for `c` should be inserted. (see `add()` for binary search tree learned in class)
  2. Fix the trees so that the properties of the max heap are maintained (i.e. the parent node must have thicker fur than its children). This means that we need to perform upheap if needed. Note that since this is also a binary search tree, we need to make sure that when performing upheap we don't break the properties of the binary search tree. To ensure this, instead of performing upheap as seen in class, we will need to implement a tree rotation that reverses the parent-child relationship whenever necessary. Depending

if the child that has to be swap in the parent position is the left or the right child, we will need to perform a right rotation or a left rotation respectively. This is how the rotations work:

– To swap the left child in the parent position, we perform a right rotation as follows:



– To swap the right child in the parent position, we perform a left rotation as follows:



The tricky part about implementing this is to make sure to fix all the necessary pointers.

For a couple of examples, see the end of the pdf.

- `CatNode.retire()` [25 points]
  This method takes as input a `Cat` object `c` and removes the key *equals* to `c` (e.g. compare them using `equals()`) from the tree rooted at the `CatNode` on which the method was called. The method returns the root of the tree obtained by removing `c` from the latter.

  As explained above the tree we are trying to implement is a binary search tree when we look at the cats' seniority, and a max heap when we look at the cats' fur thickness. Hence, removing a cat `c` from the tree has to be done while preserving both properties.

  1. Remove the node that contains a cat equal to `c` similarly to how we did it in class. **Differently from the algorithm seen in class**, when the node to be removed has both children, find the most senior cat `seniorC` in the left subtree, use `seniorC` to replace the cat to be removed, and remove `seniorC` from the left subtree (the algorithm we have seen in class was doing the same thing, but using instead the smallest key from the right subtree).
  2. If the changes made from removing the node above broke the properties of the max heap, then perform downheap to fix the tree. Here you want to think about which are the situations in which removing a node would break the properties of the max heap and tackle only such situations. *You do not want to blindly perform downheap through the entire tree.* As for upheap, also downheap will have to be implemented differently from how we have seen in class, since we need to make sure to also preserve the properties of the binary search tree. Once again, you will have to determine which of the two children

should be swapped in the parent position and based on that, perform the correct tree rotation as explained above.

For a couple of examples, see the end of the pdf.

- `CatNode.findMostSenior()` [5 points]
  This method returns the most senior cat in the tree rooted at the `CatNode` on which the method is called.

- `CatNode.findMostJunior()` [5 points]
  This method returns the most junior cat in the tree rooted at the `CatNode` on which the method is called.

- `CatCafe.buildHallOfFame()` [7 points]
  This method takes as input one integers `numOfCatsToHonor`. You can assume that the input is a number greater than or equal to zero. The method returns a list of containing `numOfCatsToHonor` cats from the cafe with the thickest fur. The cats should be appear in the list in descending order of fur thickness. If there are less than `numOfCatsToHonor` in the cafe, then all of the cats from the cafe should appear in the list.

- `CatCafe.budgetGroomingExpenses()` [5 points]
  This method takes as input an integer `numDays` indicating the number of days for which we want to budget the expenses. It returns a double indicating the expected amount of dollars that the cafe will need to spend for grooming its cat employee in the next `numDays` (inclusive).

- `CatCafe.getGroomingSchedule` [10 points]
  This method takes no inputs and returns an array list of array lists of cats. The array list at index 0 contains all the cats that need grooming in next week (i.e. within the next 7 days, including today which is represented by a 0). The array list at index 1 contains all the cats that need grooming in a week from now (i.e. within the next x days, where x is greater than or equal to 7 and less than 14). In general, the array list at index i contains all the cats that need grooming in i weeks from now. In each sublist, the cats appear in ascending order of seniority.

- `CatCafeIterator` [15 points]
  Implement the inner class `CatCafeIterator`. The method `CatCafe.iterator()` returns a `CatCafeIterator` object which can be used to iterate through all the cats in the cafe. The iterator should access the cats in ascending order of seniority. Please note that, as per documentation, the method `next()` must raise a `NoSuchElementException` if called when there are no more elements to iterate on. The exception has been imported for you in the template.

  You can implement the iterator however you like. We will test it by examining the order in which the cats are accessed. It is ok if your iterator uses an `ArrayList` to store references to all the cats in the tree. We have imported the class in the template. There are more space efficient ways to implement an iterator for trees, but you will not be tested on space efficiency.

Examples

Let's now look at a couple of examples. Consider the following cats:

```
Cat B = new Cat("Buttercup", 45, 53, 5, 85.0);
Cat C = new Cat("Chessur", 8, 23, 2, 250.0);
Cat J = new Cat("Jonesy", 0, 21, 12, 30.0);
Cat JJ = new Cat("JIJI", 156, 17, 1, 30.0);
Cat JTO = new Cat("J. Thomas O'Malley", 21, 10, 9, 20.0);
Cat MrB = new Cat("Mr. Bigglesworth", 71, 0, 31, 55.0);
Cat MrsN = new Cat("Mrs. Norris", 100, 68, 15, 115.0);
Cat T = new Cat("Toulouse", 180, 37, 14, 25.0);
```

Assume we have created an object of type `CatCafe` `cafe` using the constructor that takes no inputs. This means that to begin with, the cafe is empty. That is, the field `root` of `cafe` contains `null`.

We then call `hire()` on `cafe` with input `B` (i.e. `cafe.hire(B)`). After its execution, the cafe is represented by a tree that has only a root node as follows:

```
B(45, 53)
```

I indicated between brackets the month in which the cat was hired, followed by its fur thickness in millimeters.

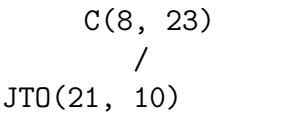
After executing the statement `cafe.hire(JTO)`, the tree representing the cafe will look as follows:

```
B(45, 53)
      \
   JTO(21, 10)
```

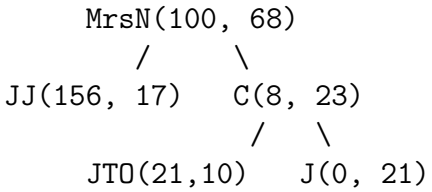
After executing the statement `cafe.hire(C)`, the tree representing the cafe will look as follows (note that here one rotation was necessary in order to maintain the max heap property of the tree):

```
B(45, 53)
      \
     C(8,23)
      /
 JTO(21, 10)
```

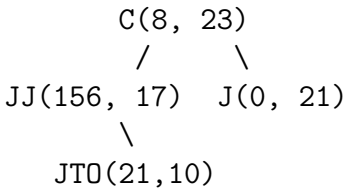
After executing the statement `cafe.retire(B)`, the tree representing the cafe will look as follows:



After executing the statements `cafe.hire(JJ)`, `cafe.hire(J)`, and `cafe.hire(MrsN)` , the tree representing the cafe will look as follows:

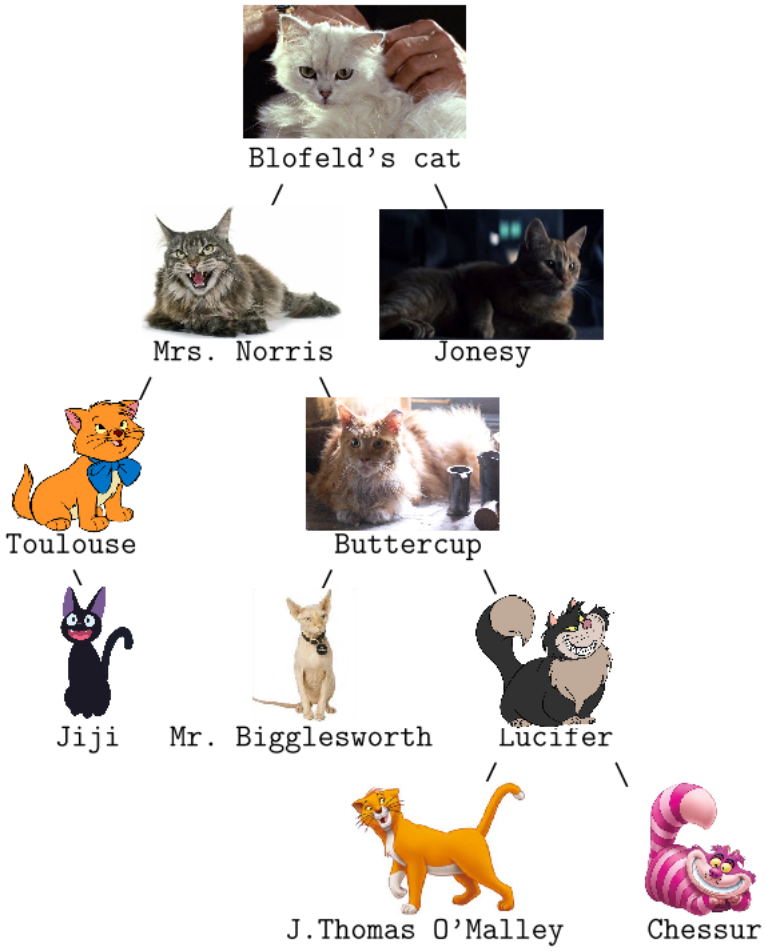


After executing the statement `cafe.retire(MrsN)`, the tree representing the cafe will look as follows:



In the following page, you can find the tree representing the cafe after the following statements have been executed:

```
cafe.hire(B);
cafe.hire(T);
cafe.hire(MrB);
cafe.hire(MrsN);
cafe.hire(new Cat("Blofeld's cat", 6, 72, 18, 120.0));
cafe.hire(new Cat("Lucifer", 10, 44, 20, 50.0));
```



We can also test the other methods as follows:

```
System.out.println(cafe.findMostSenior()); \\ displays Jonesy(0 , 21)

System.out.println(cafe.findMostJunior()); \\ displays Toulouse(180 , 37)

System.out.println(cafe.buildHallOfFame(3));
\\ displays [Blofeld's cat(6 , 72), Mrs. Norris(100 , 68), Buttercup(45 , 53)]

System.out.println(cafe.budgetGroomingExpenses(13)); \\ displays 415.0

System.out.println(cafe.getGroomingSchedule());
/*
 * displays
 * [[JIJI(156 , 17), Buttercup(45 , 53), Chessur(8 , 23)],
 *  [J. Thomas O'Malley(21 , 10), Jonesy(0 , 21)],
 *  [Toulouse(180 , 37), Mrs. Norris(100 , 68), Lucifer(10 , 44), Blofeld's cat(6 , 72)],
 *  []
 *  [Mr. Bigglesworth(71 , 0)]]
 */
```