P09 Chugidex

1 Overview

For this project you will be collecting Chugimon pocket monsters. There are 22,650 (completely unique and not at all derivative) Chugimon which populate the magical land of Yudabuyu. That's a lot of Chugimon!

In order to store your collectable Chugimon you will be writing a **recursive** binary search tree (BST) to provide storage for a Chugidex. No loops allowed! A binary search tree is a data structure used to reduce to the average time complexity of certain operations like `add`, `lookup`, etc.

To visualize your collected Chugimon, we have provided a GUI which will work once your BST methods are implemented. Hooking up the GUI isn't a required part of the assignment, but it can help you determine if your `add`, `previous`, and `next` methods are working correctly. We have also added an extra challenge for those who would like to make the Chugidex GUI more full-featured.

Grading Rubric

5 points	**Pre-Assignment Quiz:** accessible through Canvas before **11:59PM on Thursday 12/01/2022**. You CANNOT take the pre-assignment quiz for credit after its deadline, but you may still access it. It contains hints which can help you in the development of this assignment.
15 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident of this, you should write and run additional tests of your own.
20 points	**Additional Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but may test different parts of your submission in different ways.
10 points	**Manual Grading Feedback:** After the deadline for an assignment has passed, the course staff will begin manually grading your submission. We will focus on looking at your algorithms, use of programming constructs, and the style and readability of your code. This grading usually takes about a week from the hard deadline, after which you will find feedback on Gradescope.

Learning Objectives

The goals of this assignment include:

- Implement common Binary Search Tree (BST) operations.
- Further developing your experience in recursive problem-solving.
- Improve your experience in developing unit tests.

Additional Assignment Requirements and Notes

(Please read carefully!)

- Pair programming is **ALLOWED** but not required for this assignment. If you decide to work with a partner on this assignment, REGISTER your partnership NO LATER than **11:59PM on Thursday 12/01/2022** and MAKE SURE that you have read and understood the CS300 Pair Programming Policy.
- **The ONLY external libraries** you may use in your submitted files are relevant exceptions.

  ```
  import java.util.NoSuchElementException;
  ```
- Only your submitted `ChugidexTester` class can contain a **main** method.
- **Javadocs** of the classes to be implemented in this assignments are available here.
- You CAN define local variables (inside a method) or **private** helper methods that you may need to implement the methods defined in this program.
- You MUST NOT add any additional fields either instance or static to your program.
- You MUST NOT add any **public** methods either static or instance, other than those defined in this write-up to the following classes:
  - `BSTNode`,
  - `Chugimon`, and
  - `ChugidexTree`.
- Any source code provided in this specification may be included verbatim in your program without attribution.
- All the String comparisons in this assignment should be CASE-SENSITIVE.
- ALL your test methods MUST be implemented in your `ChugidexTester` class.

- In addition to the required test methods, we HIGHLY recommend (not require) that you develop your additional own unit tests (**public static methods that return a boolean**).
- You can submit your work in progress multiple times on gradescope. Your submission may include methods not implemented or with partial implementation or with a default return statement. But avoid submitting a code which does not compile. A submission which contains compile errors won't pass any of the automated tests on gradescope.
- Ensure that your code for every assignment is styled in conformance to CS300 Course Style Guide.
- You MUST adhere to the Academic Conduct Expectations and Advice.
- You MAY NOT use any iterative strategies (loops, etc) or import additional data structure classes (eg, ArrayList) in `ChugidexTree`. Recursive strategies only.

2 Getting Started

Start by creating a new Java Project in Eclipse named **P09 Chugidex**. Make sure that you are using **Java 17**, don't add a module, and that you use the default package.

You will need the following files. All of them have been provided for you in part or in full except `Chugimon.java` which you will implement from scratch.

- `p9Utility.jar` - Provides methods for getting names/types/stats for a specific Chugimon; Add to Build Path
- `BSTNode.java` - Node class used in the BST has been provided for you in full; DO NOT SUBMIT to gradescope
- `ChugidexStorage.java` - Interface which is implemented by ChugiTree; DO NOT SUBMIT to gradescope
- `ChugiTree.java` - The main class you will be implementing; COMPLETE and SUBMIT to gradescope
- `ChugidexTester.java` - The tester for your BST; COMPLETE and SUBMIT to gradescope
- `Chugimon.java` - The data class which will be stored in the BST. You will write this from scratch in the next section; COMPLETE and SUBMIT to gradescope

3 Implement the Chugimon class

Each Chugimon has two integer IDs (`private final int FIRST_ID` and `private final int SECOND_ID`) ranging from 1-151. These IDs cannot be the same. The combination of `FIRST_ID` and `SECOND_ID` determine the following features of each Chugimon:

- `private final String NAME` - The name of the Chugimon
- `private final ChugiType PRIMARY_TYPE` - The primary type of the Chugimon; cannot be null; cannot be the same as the secondary type
- `private final ChugiType SECONDARY_TYPE` - The secondary type of the Chugimon; may be null; cannot be the same as the primary type
- `private final double HEIGHT` - The height of the Chugimon in meters
- `private final double WEIGHT` - The weight of the Chugimon in kilograms

Implement the `Chugimon` class as per these javadocs. You will need to use the `ChugidexUtility` class to get name, primary type, secondary type, height, and weight of the Chugimon from a pair of IDs.

Notes:

- A Chugimon `equals` another Chugimon if their respective FIRST_ID and SECOND_ID are identical.
- Chugimon names are not unique! Some Chugimon may have the same name, but different IDs.
- A Chugimon cannot have the same `FIRST_ID` and `SECOND_ID`.
- Each Chugimon has a primary type and *may* have a secondary type. If the Chugimon does not have a secondary type, `SECONDARY_TYPE` is set to `null`.
- The enum `ChugiType` in the `p09utility.jar` provides possible types.

3.1 Chugimon comparison

In order to power the BST, we have to provide a way to compare two Chugimon and determine which is greater and which is lesser. The `Chugimon` class implements the `Comparable` interface. According to the `Comparable` interface, a `compareTo(T o)` method must return:

- a positive int if `this` is greater than the specified object
- a negative int if `this` is less than the specified object

- 0 if the `this` is equal to specified object

For this assignment, we will consider a Chugimon *less than* another Chugimon if:

1. The Chugimon's name comes before the other alphabetically
2. (If the names are equal) the Chugimon's `FIRST_ID` is less than the other's `FIRST_ID`
3. (If the names and `FIRST_ID` are equal) this Chugimon's `SECOND_ID` is less than the other's `SECOND_ID`

3.2 Test your Chugimon

Before you continue, implement the following tester methods in `ChugiTreeTester.java`.

- `testChugimonCompareToEquals()`
- `testChugimonToString()`

To check whether you are testing enough cases, upload `ChugidexTester.java` to gradescope.

4 Implement a Binary Search Tree

4.1 Checking for a valid BST

We will use the formal definition of a BST to write a method to determine whether your `ChugiTree` contains a valid tree.

> For every internal (non-leaf) node of a binary tree, all the values in a node's left subtree are less than the value in the node, and all the values in a node's right subtree are greater than the value in the node.

`isValidBST` should return true if the above conditions are satisfied and false otherwise.

Notes:

- You will implement this method in the `ChugiTree` **recursively** (for what it's worth, the the recursive strategy is *easier* than the iterative strategy).
- The BST you are writing cannot contain any duplicates. In other words, if one node equals another node, it is not a valid BST.

- DO NOT catch any exceptions in this method. You want to be able to distinguish between 1) `isValidBST` returning `false` because of incorrect node ordering, and 2) other errors deeper than the node ordering.
- You might wondering how to test this method without having implemented the `add` method. Instead of testing `isValidBST` you will be testing `isValidBSTHelper` in `testIsValidBSTHelper`. In order to do this, you can manually build a tree by creating new `BSTNodes` and using the setter methods `setLeft` and `setRight`.

4.2 Draw the rest of the owl

Suggested order:

1. `getRoot()`, `isEmpty()`, `size()`
2. `toString()`, `toStringHelper()`
3. `add()`, `addHelper()`, `testAddToStringSize()`
4. `lookup()`, `lookupHelper()`, and `testLookup()`
5. `countType` and `testCountType()`
6. `height()`, `heightHelper()`, and `testHeight()`
7. `getFirst()`, `getFirstHelper()`, and `testGetFirst()`
8. `getLast()`, `getLastHelper()`, and `testGetLast()`
9. `next()`, `nextHelper()`, and `testNext()`
10. `previous()`, `previousHelper()`, `testPrevious()`
11. `delete()`, `deleteChugimonHelper()`, and `testDelete()`

Notes:

- There are a lot of methods to implement so we have provided method signatures and documentation for helpers along with the occasional skeleton code and hints.
- Test each method as you go. If you try implement the BST all at once and save testing for later, you will be sad.
- `testAdd()` and `testDelete()` should make ample use of `isValidBST()`
- Paper and pencil are your friends. Draw a diagram of your BST to help visualize what needs to happen during each method.

5 Final Thoughts

Once your ChugiTree is implemented, you can use the provided GUI. In order to do so, you will need to download the additional files. You'll need to change the code in the constructor of `Chugidex.java` as indicated by `// TODO` tags.

6 Assignment Submission

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the CS300 Course Style Guide, you should submit your final work through Gradescope. The only THREE files that you must submit include:

- `Chugimon.java`
- `ChugidexTree.java`
- `ChugidexTester.java`

Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline. The second portion of your grade for this assignment will be determined by running that same submission against additional offline automated grading tests after the submission deadline. Finally, the third portion of your grade for your submission will be determined by humans looking for organization, clarity, commenting, and adherence to the CS300 Course Style Guide.

Extra Challenges

Here are some suggestions for interesting ways to extend this project, **after** you have completed, backed up, and submitted the graded portion of this assignment. **No extra credit will be awarded for implementing these features**, but they will provide valuable experience so that you can be the very best, like no one ever was. DO NOT submit such extensions via gradescope.

1. **Suggestion 1** – Using the `Storage` interface, implement a storage class using a linked list instead of a BST.
2. **Suggestion 2** – Improve the GUI. Here are some ideas. You are welcome to share customized `ChugidexGUI.java` and `Chugidex.java` code with your classmates on Piazza.