Creative Project 3: ZiaoFeed Quiz

Specification

Background

The website BuzzFeed, now a media and news outlet, came to popularity on social media in part due to its quizzes. On these simple, interactive websites, users are presented with a series of choices or questions to respond to, after which they are given some sort of result-- a score, categorization, or recommendation, among other options. This format has been emulated on many other entertainment and social media platforms. (Of note, BuzzFeed did not invent this format, but is likely responsible for popularizing the types of quizzes currently prevalent on social media.)

In this assignment, you will implement a version of a BuzzFeed-style quiz that we have named "ZiaoFeed"*.

**This name was more apt for when the instructor had a one-syllable name that started with 'B'. However, in keeping with the tradition of naming the assignment after the instructor, we will still call this assignment "ZiaoFeed".*

Learning Objectives

By completing this assignment, students will demonstrate their ability to:

- Define data structures to represent compound and complex data
- Structure data appropriately to efficiently solve a problem
- Write functionally correct Java classes to represent binary trees
- Produce clear and effective documentation to improve comprehension and maintainability of a method
- Write methods that are readable and maintainable, and that conform to provided guidelines for style and implementation

System Structure

In our quizzes, users will be asked repeatedly to choose which of two options they prefer until they are presented with a final result. We will represent a quiz using a binary tree, where leaf nodes represent possible results, and non-leaf nodes (branches) represent choices the user will make. When a user takes a quiz, they will be presented with the choice from the root node of the tree. Based on their response, the system will traverse to either the left or right child of the root. If the node found is a leaf, the user will be shown their result. Otherwise, the process will repeat from the new node until

a leaf is reached. Each node will also contain a number "score" which will be utilized in the creative extensions. You can assume that all scores are non-negative. See below for a full sample quiz and execution.

Quiz File Format

In addition to representing quizzes as a binary tree in our program, we will also read quizzes from and store quizzes to text files in a standard file format. In a quiz file, each node will be represented by a single line in the file containing the text for that node.

- "Choice" nodes (i.e., nodes that represent a choice between two options) will be written with the two choices separated by a single slash (`/`) character. When taking the quiz, choosing the option before the slash will move to the left child of the node, whereas choosing the option after the slash will move the right child of the node.
  - For example, `red/blue` represents a choice between red and blue, where red is the "left" choice and blue is the "right" choice.
  - You may assume that no choice will contain a slash.
- "Result" nodes will be written as the result option, prefixed with the text `END:`.
  - For example, `END:froot loops` represents a result node for the result "froot loops"
  - You may assume that no result will contain the exact text `END:`.
- Both kinds of nodes will have a dash (`-`) followed by an integer representing the score for that node.
  - You may assume that none of the questions/answers will ever contain `-` as a character.

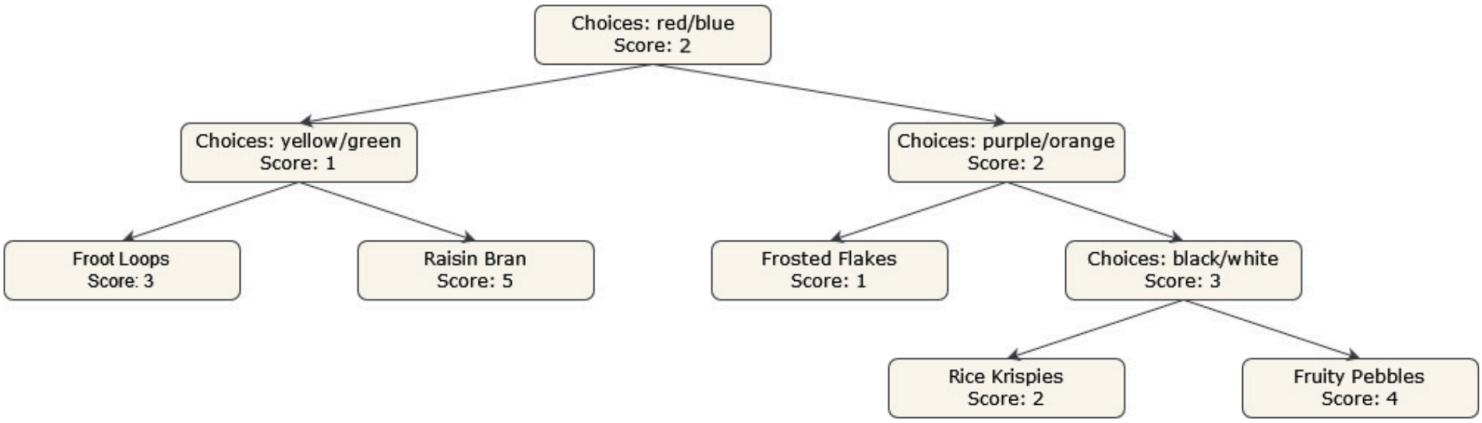
Sample Quiz File

Here is a sample quiz file representing a quiz that asks users to choose between colors to find their preferred breakfast cereal. Notice that the file represents a ***pre-order traversal*** of the resulting tree. So, for example, "red/blue" is the root of the tree, and "yellow/green" is the left child of the root. (This sample quiz has been provided to you as the file `colors-cereals.txt`.

```
red/blue-2
yellow/green-1
END:Froot Loops-3
END:Raisin Bran-5
purple/orange-2
END:Frosted Flakes-1
black/white-3
END:Rice Krispies-2
END:Fruity Pebbles-4
```

Tree Representation of Sample Quiz

Below is a visual representation of the tree (click to expand).



You may assume that each choice and result in a quiz is unique, and that no text appears as both a choice and a result. (For example, a quiz that asks the user to choose between "red" and "blue" will not have either "red" or "blue" as a possible result.)

Required Class

You will implement a class called `QuizTree` to represent a quiz as a binary tree. To earn an S on this assignment, you must implement the following methods in your `QuizTree` class:

```
public QuizTree(Scanner inputFile)
```

- Constructs a new quiz based on the provided input. See the above "Sample Quiz File" section for the expected file format.
- You may assume the provided input is in the correct format.

```
public void takeQuiz(Scanner console)
```

- Allows the user to take the current quiz using the provided `Scanner`. This method should prompt the user to choose between the options at each node and traverse the tree until a leaf node is reached, keeping track of the score as they go. When a leaf is reached, the user's result and total score should be printed. See the below "Sample Executions - Taking Quiz" section for example output.
    - Note that we will refer to the total score of a path as the sum of the scores of every single node visited on that path.

```
public void export(PrintStream outputFile)
```

- Print the current quiz to the provided output file. See the above "Sample Quiz File" section for the expected file format.

```
public void addQuestion(String toReplace, String choices,
                        String leftResult, String rightResult)
```

- Replace the node for the result `toReplace` with a new node representing a choice between the

choices in `choices` leading to `leftResult` and `rightResult` respectively.
- If `toReplace` is not a possible result in the quiz, including if it is a choice rather than a result, do *nothing*.
- `toReplace` should be treated *case-insensitively.*
    - For example, say you want to replace the result 'Froot Loops', but the tree has a result with 'fRoOT LoOps', the result should still be replaced.
- You do not need to support replacing choice nodes; only results.
- `choices` will be in the same format as in the standard file. Namely, it will look like `"<left choice>/<right choice>-<score>"` where the choice that leads to the left result is in place of `<left choice>`, the choice that leads to the right result is in place of `<right choice>`, and the score of the new choice node is in place of `<score>`.
- `leftResult` and `rightResult` will be in the format `"<result>-<score>"`.

`QuizTreeNode` class

As part of writing your `QuizTree` class, you should also create a **public static inner class** called `QuizTreeNode` to represent the nodes of the tree. The contents of this class are up to you, but must meet the following requirements:

- The fields of the `QuizTreeNode` class must be `public`.
    - All data fields should be declared `final` as well. This does not include fields representing the children of a node.
    - You may have fields within a `QuizTreeNode` that are not used depending on the type of node (question vs. answer)
- The `QuizTreeNode` class must not contain any constructors or methods that are not used by the `QuizTree` class.
- The `QuizTreeNode` class must *not* contain any logic necessary to take a quiz-- it should purely represent a node in the tree.
- You must have a single `QuizTreeNode` class that can represent both choices and results-- you should *not* create separate classes for the different types of nodes.
    - You may also not create a single base class or interface that two separate classes extend or implement. All nodes in the tree must be instances of the same class.

Sample Executions - Taking Quiz

Here are a few sample executions of taking the sample quiz above. User input is **bold and underlined**.

```
Do you prefer red or blue? red
Do you prefer yellow or green? green
Your result is: Raisin Bran
Your score is: 8
```

```
Do you prefer red or blue? blue
```

```
Do you prefer purple or orange? purple
Your result is: Frosted Flakes
Your score is: 5
```

```
Do you prefer red or blue? blue
Do you prefer purple or orange? orange
Do you prefer black or white? black
Your result is: Rice Krispies
Your score is: 9
```

```
Do you prefer red or blue? green
  Invalid response; try again.
Do you prefer red or blue? white
  Invalid response; try again.
Do you prefer red or blue? neither!!!
  Invalid response; try again.
Do you prefer red or blue? Red
Do you prefer yellow or green? YELLOW
Your result is: Froot Loops
Your score is: 6
```

Notice in the last example that when the user types an invalid option, they should be informed their response was not valid and prompted again. Notice also that options should be *case-insensitive*. (For example, the program accepted `Red` instead of `red`).

Sample Output - Modifying Quiz

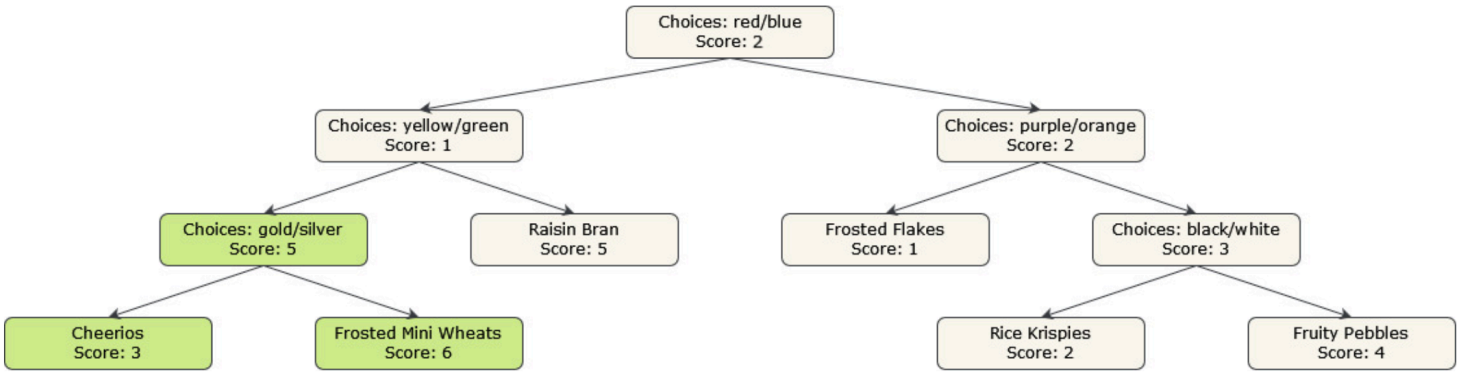
Suppose the sample quiz from the above "Sample Quiz File" section were stored in a `QuizTree` called `cereals`, and the following call were made:

```
cereals.addQuestion("Froot Loops", "gold/silver-5", "Cheerios-3", "Frosted Mini Wheats-6");
```

Then, calling `cereals.export("new-colors-cereals.txt")` would result in the following quiz in a file named `new-colors-cereals.txt`:

```
red/blue-2
yellow/green-1
gold/silver-5
END:Cheerios-3
END:Frosted Mini Wheats-6
END:Raisin Bran-5
purple/orange-2
END:Frosted Flakes-1
black/white-3
END:Rice Krispies-2
END:Fruity Pebbles-4
```

Below is a visual representation of the new tree (click to expand).



Notice that the result `Froot Loops` is no longer available in the quiz. In its place is now a new choice node choosing between `gold` and `silver`, which produce the results `Cheerios` and `Frosted Mini Wheats` respectively. These new nodes are colored green. Note that all three of these nodes should be **newly constructed nodes**, you should not modify the data of the existing result node to update the tree.

Implementation Requirements

To earn a grade higher than U on the Behavior and Concepts dimensions of this assignment, **your *core* algorithms for each method must be implemented *recursively*. You will want to utilize the *public-private pair* technique discussed in class.** You are free to create any helper methods you like, but the core of your implementations must be recursive.

Implementation Tips

- Use `Integer.parseInt(String s)` to convert a string representing a number into an integer.
  - You will likely want to represent the `score` of a `QuizTreeNode` using an `int` to make mathematical computations easier.
- Utilize the `split` method of the `String` class to split up a `String` based on a passed-in string.
- You should represent every set of choices as a a single choice node. For example, for the choices `red/blue`, you should only have a **single node for these choices**, not a node for the choice `red` and another node for the choice `blue`.
- The `writeTree` and `readTree` problems from sections 9 and 10 will be particularly helpful resources for implementing the `export` method and constructor.
- The `limitPathSum` activity from section 13 will be helpful for understanding how to track the score of the path taken through the tree.

Code Quality Requirements

- Similar to with linked lists, do not "morph" a node by directly modifying fields (especially when replacing a choice node with a result node or vice versa). Existing nodes can be rearranged in the tree, but adding a new value should always be done by creating and inserting a new node, not by modifying an existing one.

    - When modifying your `QuizTree` you should be using the `x=change(x)` pattern discussed in class to reduce redundancy.
- Look out for including additional base or recursive cases when writing recursive code. While multiple calls may be necessary, you should avoid having more cases than you need. Try to see if there are any redundant checks that can be combined!
- Include helper methods as necessary to implement your program, but all extra methods should be `private` (so outside code cannot call them).
- Limit redundancy across constructors with the `this` keyword.
- Make sure that all parameters within a method are used and necessary.
- Comment your code following the Commenting Guide. You should write comments with basic info (a header comment at the top of your file), a class comment for your `QuizTree` class, and a comment for every method.
    - Make sure to avoid including *implementation details* in your comments. In particular, for your object class, a *client* should be able to understand how to use your object effectively by only reading your class and method comments, but your comments should maintain *abstraction* by avoiding implementation details.
- Make all fields in `QuizTreeNode` as public, all fields in `QuizTree` private, and avoid fields that are not necessary for solving the problem.

Creative Extension

To earn an E on this assignment, you must implement a `void` method called `creativeExtension` in your `QuizTree` class. The method can take whatever parameters you require. Your `creativeExtension` method must implement the functionality described in one of the following options:

Option 1: Cutoff

Your method should allow a user to take the quiz just like in the `takeQuiz` method. However, the method should prompt the user for a "cutoff" before beginning its traversal. Then, when the score of the path being traversed is greater than or equal to the inputted cutoff, it should stop prompting the user for input. Instead, the method should randomly select a result based on the remaining options from the path it has traversed and print it out as the result of the quiz. If a result is reached before the cutoff is reached, then simply print out the result like normal.

Below is a sample execution using the tree represented by `colors-cereals.txt`. User input is **bold and underlined**.

▶ Expand

Option 2: Trim

Your method should prompt the user for a "cutoff". Then, it should explore all possible paths in the

tree and whenever a path's total score exceeds `cutoff`, it should replace that choice node with a random result further down in the path. If a result is reached before the `cutoff` is reached, then the path will remain unchanged.

Below is a sample execution using the tree represented by `colors-cereals.txt`. User input is **bold and underlined**.

▶ Expand

Option 3: Percentages

This method should determine the percentage of the "full score" of the tree that each result of the quiz contains and print them out. In other words, for every possible result in the quiz, compute its score (as the total score of the path to reach that node). Then, divide that score by the sum of all scores in the tree to get the percentage of the full score of the tree that the result covers.

Below is a sample execution using the tree represented by `colors-cereals.txt`.

▶ Expand

Custom Extension

If you would like, you may propose a different extension of your choice. Your proposed extension must be roughly similar in complexity to the above options and must materially change the basic assignment. If you would like to propose a custom extension, you must post it in this Ed thread and receive approval.

Optional: Create Your Own Quiz!

As part of testing your program, you may want to create your own ZiaoFeed quiz in the file format specified above. If you do, we encourage you to share your creations with your classmates and the course staff in this Ed thread!