CS1812 Assignment 1

Dr Reuben Rowe, Dr Matteo Sammartino

Submission Deadline: 16:00, Monday, 7th February 2022

Feedback Provided By: Monday, 28th February 2022

Assessment Weighting: 10% of overall course mark

Learning Outcomes: In this assignment, you will practice using recursive data structures and algorithms, file access, exception handling, and your general Java programming skills

Marking Criteria

Submissions are assessed on functionality (i.e. whether the submitted code performs the requested tasks) and coding style. You should aim to write well-formatted, well-commented code, making use of informative variable names. You should also aim to make the code as readable as possible, i.e. do not clutter the code by providing too much information.

You should try all parts and hand in what you have done, even if you are unable to complete some of the questions. Some marks are given for a correct approach even when the overall solution is incomplete or incorrect.

A `jar` file for testing is provided. Please use this testing tool as it will give an indication of your progress. This tool will be used by the markers in assessing your work. Markers will perform additional tests independently, and so passing all provided tests does not guarantee full marks.

NOTE: All the work you submit have to be solely your own work. Coursework submissions are routinely checked for this.

Overview

In this assignment, you will solve the problem of deciding which song should come next in the soundtrack of your life.

You will write a program that asks the user a series of Yes/No questions until a potential option is reached. If the program makes a suggestion that satisfies the user, the interaction ends. If the user is still not satisfied, the program asks for a song that they would prefer instead, plus a question that distinguishes it from the suggested option. This option and question are then added to a database. That is, the program learns new suggestions and questions and its ability to recommend good options grows.

One program interaction should look as follows (user input is marked by an initial “<” and program output is marked by an initial “>”):

```
> Welcome! Let's help you find a song.
> Do you like classical music? [y/n]
< n
> Perhaps you would like "Take Five" by Dave Brubeck.
> Is this satisfactory? [y/n]
< n
> What would you prefer instead?
< "Easy On Me" by Adele
> Tell me a question that distinguishes "Take Five" by Dave Brubeck from
  "Easy On Me" by Adele
< Do you dislike jazz?
> Do you want to play again? [y/n]
< y
```

The interaction would then continue to a second round, and this time the program will give the good suggestion:

```
> Do you like classical music? [y/n]
< n
> Do you dislike jazz? [y/n]
< y
> Perhaps you would like "Easy On Me" by Adele.
> Is this satisfactory? [y/n]
< y
> Do you want to play again? [y/n]
< n
```

1 The Question Tree [15 marks]

Your program will store the questions and suggestions using a binary tree. Questions are regular nodes with two children (one child representing what to do if the user answers “yes”, and one representing what to do if the user answers “no”), and song suggestions are leaves in the tree (i.e. nodes without children).

Write a class `TreeNode` for representing binary trees of strings. The class should contain the following:

- A constructor `public TreeNode(String value)` for constructing a leaf.
- A constructor `public TreeNode(String value, TreeNode left, TreeNode right)` for constructing a tree node with left and right children.
- Methods `public TreeNode getLeft()` and `public TreeNode getRight()` for getting the left and right children.
- A method `public boolean isLeaf()` to check whether a `TreeNode` is a leaf.

You can use the testing tool to test your implementation. In the same directory as your class `TreeNode`, run

```
$ java -jar assignment1Tester.jar
```

At this point you should pass the first test, but not the subsequent tests for the later questions.

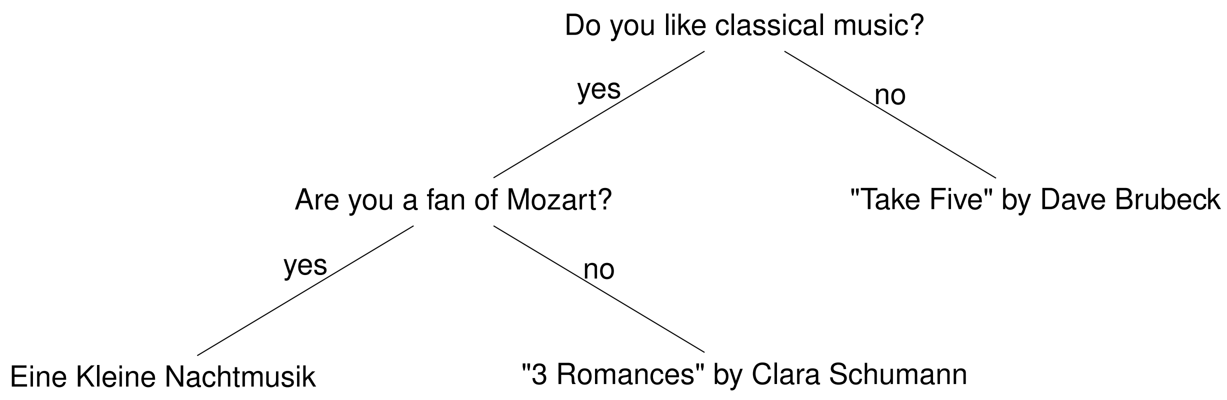
Hint. Review the lecture on recursive data types.

Help My Tests Are Failing. Some possible causes of error are listed below.

1. Ensure that you compile your code before running the tool.
2. If the testing tool cannot find your `TreeNode` class, make sure that there is no `package` line in `TreeNode.java`. The tool looks for the class `TreeNode`. It is not smart enough to look for `my.package.name.TreeNode`.

2 The Game Logic [30 marks]

Create a class `SongSuggester` that will contain most of the game logic. Add a main method to it that constructs the basic initial tree shown below.


Do you like classical music?
yes
no
Are you a fan of Mozart?
"Take Five" by Dave Brubeck
yes
no
Eine Kleine Nachtmusik
"3 Romances" by Clara Schumann

Reading User Input

You will need to read input from the user. You can use the `nextLine()` method in the `Scanner` class for reading a single string from the keyboard.

The program frequently needs to ask Yes/No questions, so it makes sense to encapsulate this functionality in a method. Lines containing questions should end with `[y/n]`. For example

```
    Do you like classical music? [y/n]
```

In the `SongSuggester` class, create a method `boolean askYesNo(String question)` that asks a Yes/No question and reads the user's answer. If the user answers "y", the method should return `true`, otherwise it should return `false`.

The Interaction

The interaction is controlled by the data in the tree. Example interactions were given in the overview.

Each round of the game begins at the root, which holds the first question to be asked. The game then executes a loop that does the following.

- Asks the current question.
- If the player answers "yes", makes the left subtree the next current question.
- If the player answers "no", makes the right subtree the next current question.

The next iteration of the loop then asks this new question.

Thus the game moves down the tree until it encounters a leaf, which will contain a suggestion instead of a further question. The program should present it to the user. For now, the computer will then just ask the user whether she would like to play again, in which case the interaction starts from the root again. You may find it convenient to have an outer `do while` loop for repeated interactions.

Your game should now already be playable, using the questions and answers that you explicitly added when constructing the tree.

Use the testing tool to make sure you have followed the correct design.

Hint. Review the lectures on recursion and recursive data types.

Help My Tests Are Failing. Some possible causes of error are listed below.

1. The tool tries to run your program with the command

   ```
   $ java SongSuggester
   ```

   In the same place where you are running the testing tool, make sure that this command succeeds. If you have a `package` line in `SongSuggester.java` you will need to run your code with `java my.package.name.SongSuggester`. The testing tool is not smart enough to know your package name, so will fail!

2. Blank or extraneous lines in your output will confuse the tool. Make sure your output exactly matches the examples in the overview.

3. You are using `System.out.print` instead of `System.out.println` when asking a question.

3 Learning New Questions [30 marks]

Extend the game such that when the computer makes a suggestion, it asks the user whether it was satisfactory. If it was not, then the program asks for:

1. the name of a preferred song; and
2. a question that is true for the preferred song, and false for the rejected suggestion.

In subsequent rounds, the program will now ask this new question once it reaches the same position in the tree and then, depending on the user's answer, suggest the new or existing option.

To implement this, you will need to update the tree of questions after a bad suggestion in the follwing way.

- The new question must replace the old leaf.
- The new suggestion must become the left child of the new question.
- The previous suggestion (i.e. the old leaf) must become the right child of the new question.

Before writing any code, draw yourself a picture of the tree and how you plan to update it.

Test your code by playing multiple rounds of the game and using the provided testing tool.

4 Saving a Tree in Preorder [25 marks]

In order not to lose the new knowledge, your program must save the tree between games. When the user wants to quit at the end of a round, your program should ask whether the questions should be saved. An example interaction is shown below.

```
> Welcome! Let's help you find a song.
> Do you like classical music? [y/n]
< n
> Perhaps you would like "Take Five" by Dave Brubeck.
> Is this satisfactory? [y/n]
< n
> What would you prefer instead?
< "Easy On Me" by Adele
> Tell me a question that distinguishes "Take Five" by Dave Brubeck from
  "Easy On Me" by Adele
< Do you dislike jazz?
> Do you want to play again? [y/n]
< n
> Do you want to save the current tree? [y/n]
< y
```

Preorder

To save the questions, your program should write the binary tree into a text file with the name `suggestions.txt`. Implement an algorithm that writes out the questions and answers to this file in preorder (see the lecture on recursive data structures). This is most easily done by adding a recursive method to the `TreeNode` class. Test your preorder algorithm first by just printing the questions and answers to the screen instead of to the file. Once you have confirmed that it works as expected, you can implement the actual file access.

Note that to be able to load the tree from the file again, you will need to distinguish a line with a question from a line with a suggestion. For example, you could start each line containing a suggestion with a special character sequence.

File Output

Use `new BufferedWriter(new FileWriter(filename))` to create a `BufferedWriter` for writing to a file with name `filename`, and then use its `write` method to output strings. After writing a question or answer, you will have to explicitly insert an end of line character using the `newLine()` method.

You can use the testing tool to verify that the file `suggestions.txt` is created.

Hint. You have to use a single instance of `BufferedWriter` to write the file, so you should pass it as an argument to your recursive preorder method.

Exceptions

All methods of BufferedWriter may throw exceptions of type IOException (from the java.io package). The entire process of writing the file should be surrounded by a try-catch-finally or a try-with-resources construction that makes sure your program does not crash in case of an error, and that the close() method of the BufferedWriter is called whether or not an exception is thrown.

Hint Review the lectures on I/O streams and exceptions.

Loading a Tree from Preorder

When your program starts, it should reconstruct the suggestions tree from the saved questions in suggestions.txt. To read from the file whose name is stored in a variable filename, use

```
        new BufferedReader(new FileReader(filename))
```

and the readLine method of BufferedReader that reads an entire line into a string.

The best way to do this is to implement a recursive static method fromPreorder, which reads a line in the file and then recursively calls itself to create the left and right subtrees. The base case of the recursion is reached when the line read from the file contains an answer instead of a question. For detecting this case, you can use the special character sequence inserted when writing the file.

Again, the methods may throw exceptions of type IOException, which you will need to handle appropriately. When loading the tree fails (e.g. because the file was not found), you should instead create the default tree above.

You can use the testing tool to verify that the file is being correctly saved and loaded.