Project 3: COVID Health Check

DUE: Sunday, October 18 at 11:59pm

Extra Credit Available for Early Submissions!

Basic Procedures

You must:

- Fill out a readme.txt file with your information (goes in your user folder, an example readme.txt file is provided)
- Have a style (indentation, good variable names, etc.)
- Comment your code well in JavaDoc style (no need to overdo it, just do it well)
- Have code that compiles with the command: **`javac *.java`** in your user directory
- Have code that runs with the command: `java` **Driver data2.txt**

You may:

- Add additional methods and variables, however these methods **must be private**.
- Use built in Java classes ArrayList, Scanner, and java.io package for file processing.

You may NOT:

- Make your program part of a package.
- Add additional public methods or variables
- Alter any method signatures defined in this document of the template code (if any). Note: "throws" is part of the method signature in Java, don't add/remove these.
- Add any additional libraries/packages which require downloading from the internet.

Setup

- Download the `project3.zip` and unzip it. This will create a folder `section-yourGMUUserName-p3`;
- Rename the folder replacing `section` with the `001`, `002`, `003`, `005` based on the lecture section you are in;
- Rename the folder replacing `yourGMUUserName` with the first part of your GMU email address;
- After renaming, your folder should be named something like: `001-tmengis-p3`.
- Complete the `readme.txt` file (an example file is included: `exampleReadmeFile.txt`)

Submission Instructions

- Make a backup copy of your user folder!
- Remove all test files, jar files, class files, etc.
- You should just submit your java files and your readme.txt
- Zip your user folder (not just the files) and name the zip `section-username-p1.zip` (no other type of archive) following the same rules for `section` and `username` as described above.
    - The submitted file should look something like this:

```
001-tmengis-p1.zip --> 001-tmengis-p1 -->     JavaFile1.java
                                              JavaFile2.java
                                              JavaFile3.java
                                              ...
```

- Submit to blackboard.

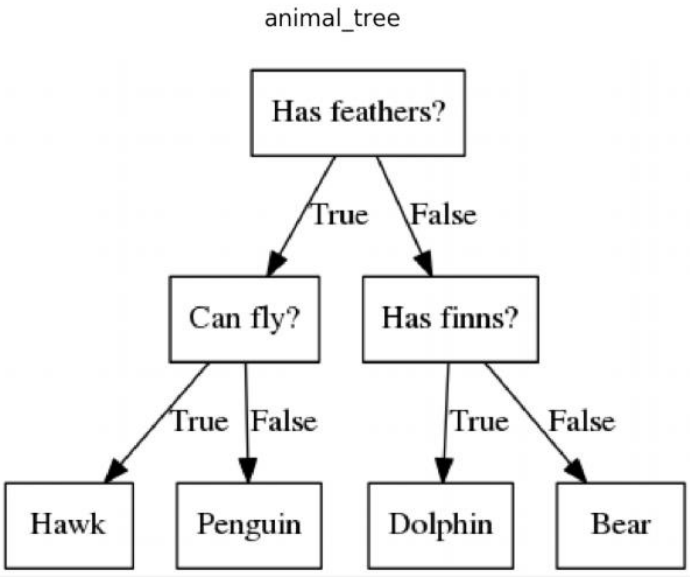
Grading Rubric

Due to the complexity of this assignment, an accompanying grading rubric pdf has been included with this assignment. Please refer to this document for a complete explanation of the grading.

Overview

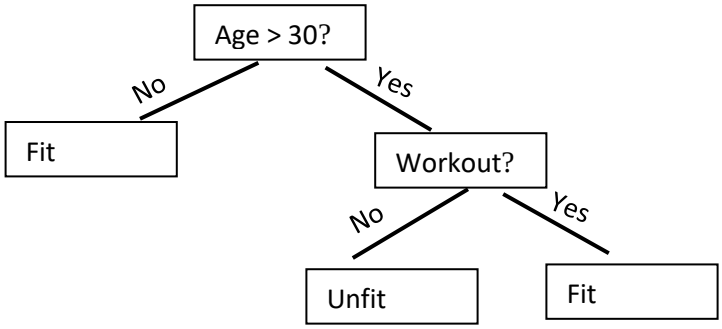
In this project, we are going to work on COVID-19, again 😊. This time we are going to help GMU to have an intelligent system that decides if a patriot has COVID-19 or not after analyzing the responses of the individual for a set of questions. We will use a decision tree data structure and recursion.

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules. The following diagram depicts an example decision tree.



In this project, you will build a decision tree by reading a coma separated text file from a hard disk and use that tree to give recommendation if the user can visit campus, get tested, or seek emergency medical service. When the program runs, it will ask the user a series of questions and determines if the individual potentially has COVID-19 or not. This decision will be based on the tree built from the txt file, which contains the level-order traversal of a decision tree. Due to the unprecedented nature of the virus, there are still things we must learn about it. To accommodate new information, our system is smart enough to learn from its environment by asking suggestion from its user. Based on the accepted information, your program will update the tree and use that new information in the next round of questions to the user.

The following is the demo of the system using a simple decision tree depicted as follows. The corresponding text file (data2.txt) is included in the project folder.



Example Run (Command Line)

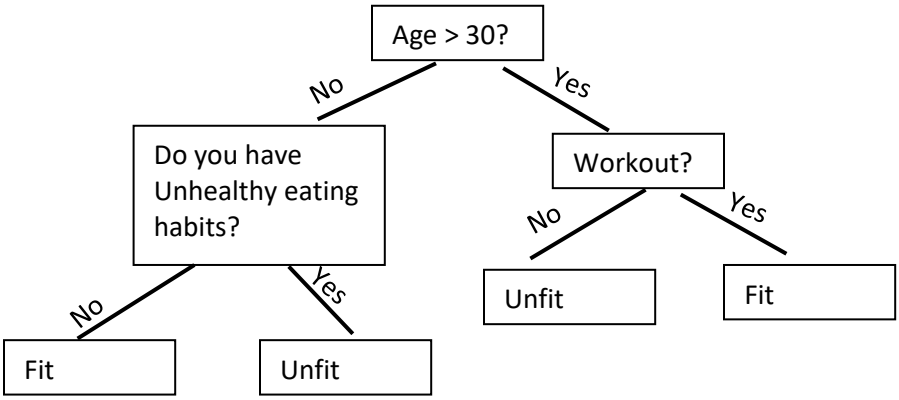
- java Driver data2.txt

```
Age >30?
yes
workout?
yes
fit
Satisfied by my intelligence ?
yes
Try again? yes
Age >30?
no
fit
Satisfied by my intelligence ?
yes
Try again? no
Have a nice day!
```

If the user is not satisfied:

```
Age >30?
no
fit
Satisfied by my intelligence ?
no
What should be the answer?
Unfit
Give me a question whose answer is yes for Unfit  but no for  fit
Do you have unhealthy eating habits?
Try again? yes
Age >30?
no
Do you have unhealthy eating habits?
yes
Unfit
Satisfied by my intelligence ?
```

The tree after the update:



Implementation/Classes

This project will be built using a number of classes representing the component pieces of the Covid Health Check for GMU. Here we provide a description of these classes.

INTERFACE (BinaryTreeInterface): **`BinaryTreeInterface.java`**

This interface represents all the functionalities a generic binary tree.

Operations

+ getRootData(): T - This method returns the data stored in the root node of a tree. Note that you cannot return the data of an empty tree.

+ getRootNode(): BinaryNode<T> - This method returns the reference to the root node of a tree.

+ setRootNode(BinaryNode<T>):void - This method sets the root node of a tree.

+ getHeight():int - This method returns the height of a tree.

+ getNumberOfNodes(): int - This method returns the number of nodes in a tree.

+ isEmpty():boolean - This method returns true, if a tree is empty, false otherwise.

+ clear():void – clears a tree.

INTERFACE (DecisionTreeInterface): **`DecisonTreeInterface.java`**

This interface represents all the functionalities the Decision tree that is going to be used by our COVID-19 health check. It extends the BinaryTreeInterface

Operations

+ isAnswer(): boolean - This method returns true if the current node contains the final decision(answer), false otherwise. Note which node contains the final answer in decision trees.

+ moveToNo():void - Set the current node to its left child. Note that the current node should not be null.

+ moveToYes(): void - Set the current node to its right child. Note that the current node should not be null.

+ resetCurrentNode():void - Set the current node to the root.

+ getCurrentNode():BinaryNode<T> - returns the reference to the current node.

+ getCurrentData():T – returns the data portion of the current node.

+ setResponses(T responseForNo, T responseForYes):void – Sets the data in the children (left and right) of the current node.

Note that this interface should be generic, i.e., it should be able to accommodate any type of object. The Covid testing is just an example.

CLASS (BinaryNode): **BinaryNode.java**

This generic class represents the nodes in our tree data structure. It has three attributes:

- data – a generic element that represents the data
- leftChild – a BinaryNode that represents the left child of the node
- rightChild - a BinaryNode that represents the right child of the node

It also has the following behaviors:

+BinaryNode(): constructor - initializes all the attributes to null.

+ BinaryNode(data): constructor - initializes the data with the accepted value and the rest attributes to null.

+ BinaryNode(data, leftNode, rightNode): constructor - initializes the attributes to the given values.

+ setData() and getData(): - the setter and getter methods for the data attribute.

+ setLeftChild() and getLeftChild(): - the setter and getter methods for the leftChild attribute.

+ setRightChild() and getRightChild(): - the setter and getter methods for the rightChild attribute.

+ hasLeftChild(): boolean - returns true, if a node has a left child, false otherwise.

+ hasRightChild(): boolean - returns true, if a node has a right child, false otherwise.

+ isLeaf(): boolean - returns true, if a node is a leaf, false otherwise.

+ getHeight():int - This method returns the height of a tree rooted at the node.

+ getNumberOfNodes(): int - This method returns the number of nodes in a tree rooted at the node.

+ copy(): BinaryNode - copy the subtree rooted at a node and returns the root of the copied subtree.

Note that you can use the implementation given in the textbook for BinaryNode class as you see fit.

CLASS (BinaryTree): **BinaryTree.java**

This generic class implements the BinaryTreeInterface. The class has one attribute root, which represents the root node of a tree. In addition, it has the following behaviors:

+BinaryTree(): constructor - initializes root to null.

+ BinaryTree(data): constructor - initializes the root node's data.

+ BinaryTree(data, leftTree, rightTree): constructor - initializes the root node with the given values.

+ setTree(data, leftTree, rightTree): void- set the tree to the root with data and the leftTree and rightTree subtrees. Make sure that the new created tree has only one point of entry, which is the root.

+inorderTraversal(): void – displays all the nodes in tree using inorder traversal. It displays the content of the nodes separated by a space in a single line. For example, if the nodes' content are 1, 2, 3, and 4, the method displays "1 2 3 4". We have to have a non empty tree for traversals.

Note that you can use the implementation given in the textbook for BinaryTree class as you see fit.

CLASS (DecisionTree): **DecisionTree.java**

This generic class extends BinaryTree and implements the DecisionTreeInterface. It has an attribute currentNode that keeps track of where we are in the tree. It also has the following behaviors:

+ DecisionTree(): constructor - initializes the attributes to null.

+ DecisionTree(data): constructor - initializes the root node's data and the other attribute to null.

CLASS (COVIDHealthBuilder): **`CovidHealthBuilder.java`**

This class builds the decision tree that we will use for our application. It has the following attribute

- healthTree - a decision tree data structure that will contain the data. It has a datatype of DecisionTreeInterface. This tree will be initialized by reading the content of a file recursively.

The class has the following important methods:

+ CovidHealthBuilder(String fileName):constructor – This constructor accepts the file name that contains the content of a tree (as specified in readData()), builds, and initializes the healthTree using a recursive method buildTree().

+ readData(String):ArrayList<String> - this method accepts the file name that contains contents of a tree, where each node in a given level will be separated by a comma. Each line in the file contains the nodes in each level (left to right). The first line contains the root of a tree. If a node does not have a child (left or right), the file uses "null" as a place holder. The method returns an ArrayList that contains all the contents of the file.

+ buildTree(ArrayList<String>, BinaryNode<String>, int): BinaryNode – this method builds a binary decision tree recursively. It accepts the arraylist returned by the readData method, a binary node that represents a node in a tree and the index of the elements in the arraylist. This method should be implemented **recursively**. It will be used by the constructor to initialize the healthTree. **There will be a manual check for this method during grading.**

+ decide():void – This method asks a series of questions based on the tree initialized (healthTree) during the instantiation of the class. It will accept either "no" or "yes" from the user. After displaying the decision, it will ask the user if the decision is to the user satisfaction by displaying "Satisfied by my intelligence?". Based on the answer of the user for this question, the method either do nothing (for yes response), or update (using the learn() method below) the tree by asking another set of questions(for no answer).Please take a look at the second screenshot in the example above.

+ learn():void – This method is going to handle if the user is not satisfied with the output of the decide() method, i.e., if the user enters "no" for the "Satisfied by my intelligence?" question. Please, take a look at the second screenshot in the example Run above. Based on the answer for the two questions: "What should be the answer" and "Give me a question whose answer is yes for **newAnswer** but no for **OldAnswer**" , respectively, it will update the tree.

+ updateTree(String question, String noAnswer, String yesAnswer): void
- this method updates the healthTree with a new node (question) together with its left child(noAnswer) and right child (yesAnswer). Please take a look at the second screenshot in the example above.

+getHealthTree():this method returns the healthTree of the class.

CLASSNAME (EmptyTreeException): EmptyTreeException.java

Some of the operations in the classes throw exception. For example, getRootData() cannot return a value if the tree is empty. EmptyTreeException handles this error by returning a string. The class extends Java's RuntimeException class.

Requirements

An overview of the requirements are listed below, please see the grading rubric for more details.

- **Implementing the classes** - You will need to implement required classes.
- **JavaDocs** - You are required to write JavaDoc comments for all the required classes and methods. Check provided classes for example JavaDoc comments.

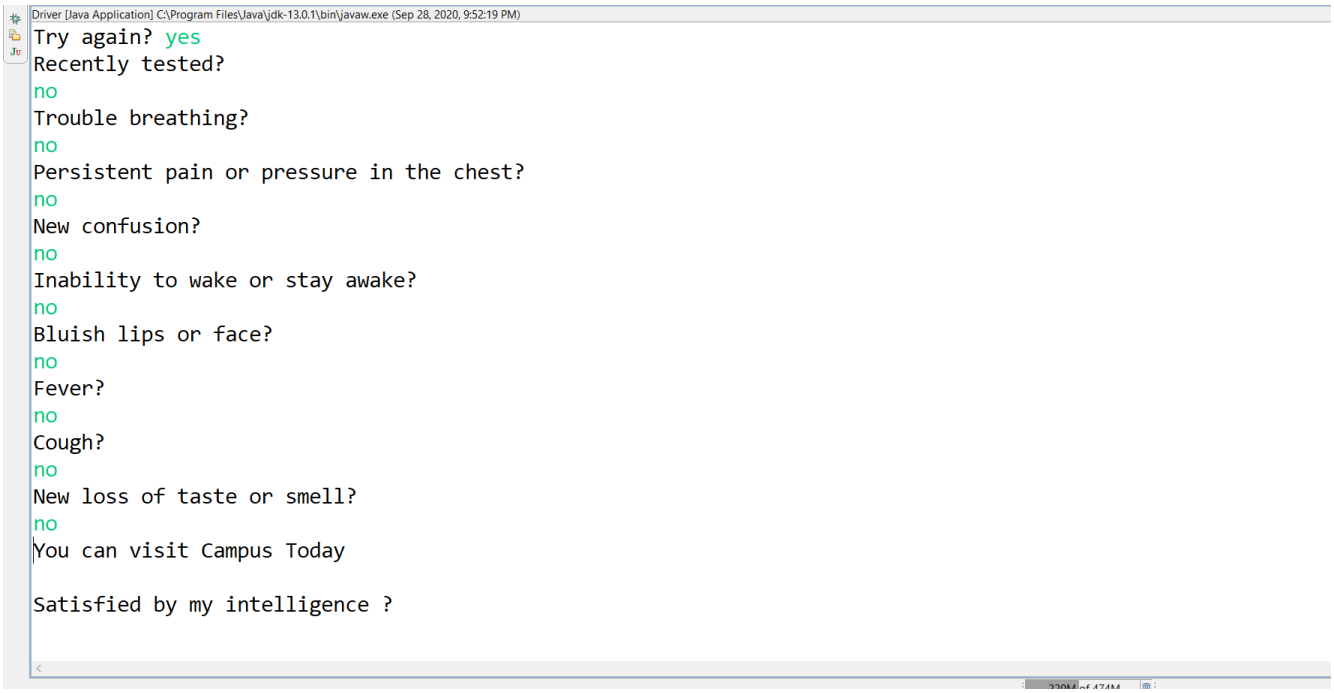
Testing

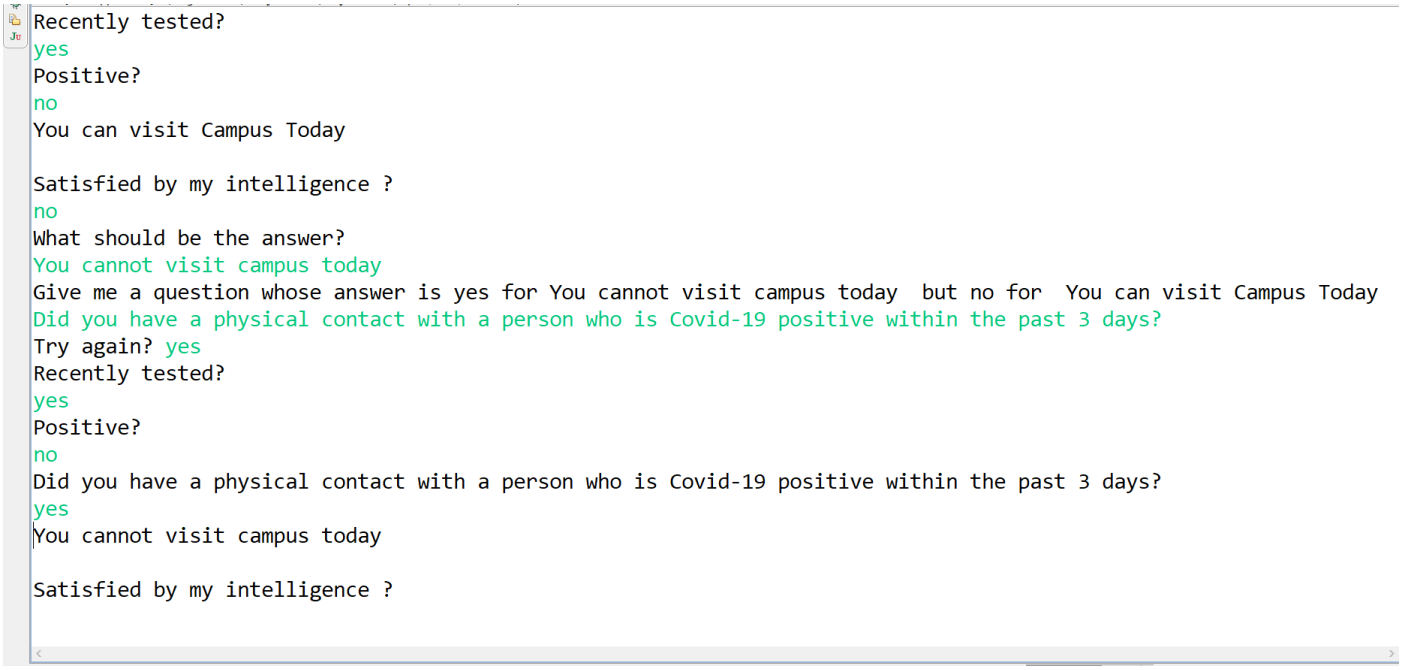
The main method provided in the template files contain useful code to test your project as you work. You can use command like "**java Driver data.txt**" to run the testing defined in **main()**. JUnit test cases will not be provided for this project, but feel free to create JUnit tests for yourself. A part of your grade *will* be based on automatic grading using test cases made from the specifications provided. The following are some of the screenshots of the expected output for the command **java Driver data.txt and** some user inputs (green ones).

```
Driver [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (Sep 28, 2020, 9:52:19 PM)
Recently tested?
no
Trouble breathing?
no
Persistent pain or pressure in the chest?
no
New confusion?
no
Inability to wake or stay awake?
no
Bluish lips or face?
no
Fever?
yes
Sore throat?
yes
Shortness of breath or difficulty breathing?
no
Headace?
yes
Please get tested

Satisfied by my intelligence ?
```

```
Bluish lips or face?
no
Fever?
yes
Sore throat?
yes
Shortness of breath or difficulty breathing?
no
Headace?
yes
Please get tested

Satisfied by my intelligence ?
yes
Try again? yes
Recently tested?
yes
Positive?
yes
Please quarantine yourself for 10 days

Satisfied by my intelligence ?
```

```
Recently tested?
yes
Positive?
yes
Please quarantine yourself for 10 days

Satisfied by my intelligence ?
yes
Try again? yes
Recently tested?
no
Trouble breathing?
no
Persistent pain or pressure in the chest?
yes
seek emergency medical care immediately

Satisfied by my intelligence ?
```


Driver [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (Sep 28, 2020, 9:52:19 PM)
Try again? yes
Recently tested?
no
Trouble breathing?
no
Persistent pain or pressure in the chest?
no
New confusion?
no
Inability to wake or stay awake?
no
Bluish lips or face?
no
Fever?
no
Cough?
no
New loss of taste or smell?
no
You can visit Campus Today
Satisfied by my intelligence ?


Recently tested?
yes
Positive?
no
You can visit Campus Today
Satisfied by my intelligence ?
no
What should be the answer?
You cannot visit campus today
Give me a question whose answer is yes for You cannot visit campus today but no for You can visit Campus Today
Did you have a physical contact with a person who is Covid-19 positive within the past 3 days?
Try again? yes
Recently tested?
yes
Positive?
no
Did you have a physical contact with a person who is Covid-19 positive within the past 3 days?
yes
You cannot visit campus today
Satisfied by my intelligence ?