Assignment 1 of CSCB07
Due date: 9th October @ 11:59pm

1 Introduction

This assignment has two parts. Part one involves writing some basic code involving binary trees that you should find familiar from your CSCA48, albeit we will attempt this problem using Java and not Python. The purpose of this part is to provide you with some more practice of writing Java code and get familiar with the syntax and other nuances of Java. Part two of this assignment involves you designing a set of classes for maintaining a mock file system and interacting with it in a program that operates like a Unix shell. We will study CRC cards in lecture on 1st October. This assignment must be worked in teams of two. We recommend that each one of you work on both parts of the assignment and that we should see commits by both the team members towards each of the section of the assignment. **You must work with a partner. Submitting this alone is not an option; if you do, we will give you a mark of 0.** Your Assignment1 SVN repo is of the following form `http://markus.utsc.utoronto.ca/svn/cscb07f18/group_0ZZZ` where ZZZ is your group number and you can find this by logging into MarkUs here `https://markus.utsc.utoronto.ca/cscb07f18`.

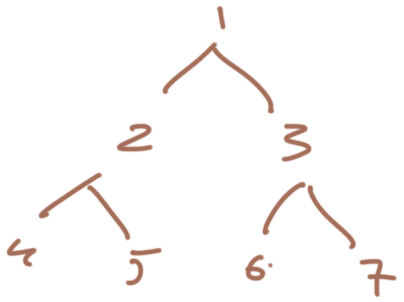
2 Part one of assignment

2.1 Binary Trees

If you were to open up JavaDoc for most collections in Java for example, you can find here `https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#toList--` that there is a method called `toList` that simply converts the items in the collection to a list. In this spirit, we will try and attempt to provide a similar functionality to a binary tree such that you can convert the items in a binary tree to a list. In the starter code `BinaryTree.java` and `Node.java` that we provide, you are first asked to understand the starter code and make sure that you understand what the `Node` class and the method `addData` inside the BinaryTree does. The `addData` method is already implemented for you, and you are not asked to change this function. The `addData` method adds the integers in the binary tree level by level and starting from left to right. So for example, when the following code is called in the provided `main` method of `BinaryTree`

```
BinaryTree bt=new BinaryTree();
bt.addData(1);
bt.addData(2);
bt.addData(3);
bt.addData(4);
bt.addData(5);
bt.addData(6);
bt.addData(7);
```

The binary tree that gets created will look like as shown in the next figure:



The instance methods that you are asked to complete for this assignment are the following:

- `public ArrayList toList()`

  The ArrayList that you return back from this method is an ArrayList that contains ints (i.e., the data of each of the Node of the BinaryTree). The toList method will in turn call the method addSubTree. You must ensure that the ArrayList that is returned back from this method contains ints of the BinaryTree in Inorder traversal.

- `private void addSubTree(Node temp, ArrayList values)`

  This method must use recursion to populate the ArrayList using inOrder traversal of the tree.

- `public String toString()`

  You must complete this method so that you return back a textual representation of the binary tree. The textual representation of the binary tree is returning back a string such that the string contains all the ints of the binary tree level by level and moving from left to right. You will find the expected output of this method in the main function.

Do not change the signature of the above methods and do not turn them into static method. Doing so will result in zero towards this part of the assignment. We remind you again to first get familiar witht the `addData` method inside the `BinaryTree` class so that it may also assist you in completing the above methods.

3 Part two of assignment

3.1 What's a shell?

Before the GUI, there was the command line. All interactions with the file system were done using commands, much like what we have been seeing in lectures (Lecture1, your instructor using the svn client via the terminal) and in your future labs. Rather than double-clicking on windows containing pictures of the contents of a hard drive, people would type commands:

```
mv oldfile newfile to rename a file or move it to a different directory
cd to change directories
cat filename to display the contents of a file called filename
```

A command consists of the command name followed (optionally) by a space and then arguments for the command. A shell is just another program. Most are written in the programming language C. Shells interact with the operating system. Java's virtual machine is like an operating system. Your program needs to support the equivalents of these bash commands (italics indicate the names of files and directories or something else that the user chooses). Everything is case sensitive.

1. exit

   Quit the program.

2. mkdir DIR

   Create a directory DIR, which may be relative to the current directory or may be a full path.

3. cd DIR

   Change directory to DIR, which may be relative to the current directory or may be a full path. As with Unix, .. means a parent directory and . means the current directory. The directory separator must be /, the forward slash. The root of the file system is a single slash: /.

4. ls

   Print the names of files and directories in the current directory, with a new line following each of them.

5. pwd

   Print the current working directory (including the whole path).

6. mv OLDFILE NEWFILE

   Move file OLDFILE to NEWFILE. Both OLDFILE and NEWFILE may be relative to the current directory or may be full paths.

7. cp OLDFILE NEWFILE

   Like mv, but don't remove OLDFILE.

8. cat FILE

   Display the contents of FILE in the shell.

9. get URL

   URL is a web address. This command retrieves the file at that URL and adds it to the current directory.

10. echo STRING > OUTFILE

    Put STRING in file OUTFILE. STRING is a string of characters surrounded by quotation marks. This creates a new file if OUTFILE does not exist, and erases the old contents if OUTFILE already exists. In either case, the only thing in OUTFILE should be STRING.

11. echo STRING >> OUTFILE

    Like the previous command, but appends instead of overwrites.

3.2 Design using CRC cards

Create a set of CRC cards that could be used to implement the shell described. Instead of handing in your index cards, there is a directory named crcCards in your subversion repository. In crcCards, for each card that create, create a text file (with a .txt suffix) with the name of the class. Each CRC Card (or each .txt file) contains the following format:

```
Class name: Classname
Parent class (if any): Classname
Subclasses (if any): List all the subclasses separated by a comma.

Responsibilities:
*g
*h
*i

Collaborators:
*j
*k
*l
```

Do a thorough job on the CRC analysis. There may be hidden responsibilities that are not listed, but are necessary in order to write the program. Make sure your responsibilities are clearly written: the grader (one of the TAs) should easily be able to understand each of them.