University of New Brunswick
CS1083 – Open Access Version

Module 11 Assignment

The purpose of this assignment is to give you practice:

- Coding and using a binary search tree; and
- Throwing and catching an exception.

For all assignments:

- Follow the instructions in the document "Java Coding Guidelines.pdf" available within the "Start Here" module, under the "Assignments" topic.
- For each .java file you create, include a javadoc comment (one that begins with /**) at the beginning of the file that describes that Java class. This comment block should include a line that begins with @author followed by your name and student number on the same line. (Note: inclusion of this comment is part of the instructions given in "Java Coding Guidelines.pdf")
- Include comments throughout your programs to explain any non-obvious portions of your code.
- It is recommended that you create a separate folder on your computer for each assignment. You might even wish to create separate sub-folders within an assignment folder if the assignment has multiple parts. Keeping your work organized makes it easier to find things later when you want to review what you have done.
- Few things in life are more frustrating than losing your work while working on an assignment. Get in the habit of saving frequently when working on an assignment. Also, regularly make backup copies of any files you create as part of your work for this course.
- Each module has a link to the discussion forum where you can ask questions and seek help in completing your assignments.
- Submitting your assignment involves creating a pdf file and uploading that single file to the assignment drop box on D2L. In general, that file will tend to include all Java code you wrote for the assignment, plus any output from running your programs that the assignment instructions specify you should capture. Specific submission instructions are included at the end of each assignment.
- To create the pdf file for each assignment, begin by opening a new document using the word processing program of your choice. Save the document using the name “Your Name CS1083 Module x Assignment Submission.docx” Replace x

with the correct module number. “docx” may be different depending on which word processing software you choose to use.

- At the beginning of your submission document enter your name, your student number, CS1083, the assignment name (this one is “Module 2 Assignment”), and the date. It doesn’t matter if the date is when you started working on the assignment or when you submit it – either will do.
- You can add content to your submission document as you work on the various questions in the assignment. Clearly label each part of the assignment (“Part A” etc.) in your document. When your document is complete, save / export as a pdf file. Always make sure your pdf file opens properly before uploading it.
- To include Java code in your submission document, copy all the text in your .java file and then paste that text into your submission document. Use a monospaced font (e.g.: Consolas, Courier) for your code to maintain proper indentation.
- To include output from running your program in your submission document, you have two choices. You can either (a) copy and paste the text from your command prompt window, or (b) capture a screen shot of the output and include that as a picture / image in your submission document. Either approach is fine.
- To copy text from your command prompt window, try selecting the desired text and then pressing either command-c (Mac) or control-c (Windows or Linux). If you have issues, you can always use Google to see how to do this on your specific type of computer.
- To capture a screen shot of a selected portion of your command prompt window, try command-shift-4 (Mac), WindowsKey-shift-s (Windows), or shift-PrtScrn (Linux).
- Once you have pasted your output text into your submission document, make sure it is in a monospaced font (e.g.: Consolas, Courier) so you retain the alignment of output.

Finding Duplicate Numbers

Many types of real-world computer applications involve records with unique identifiers. Often numbers are used as identifiers. Examples include your social insurance number, social security number, driver's license number, or employee id.

The Module 2 Assignment provides yet another example; each LotteryTicket object includes a unique ticket id.

Often with such an application there is a need to ensure that no duplicate id numbers have been assigned. Your task for this assignment is to simulate checking that there are no duplicates within a set of id numbers.

Binary search trees provide a fast and efficient way to accomplish this task, so begin by modifying the **BinaryTreeNode** class provided for download earlier in this module.

Modify this class to store an id number of type int in each node (replacing the 'name' instance variable of type String that is currently there). Update the constructor and all existing methods to be consistent with this change.

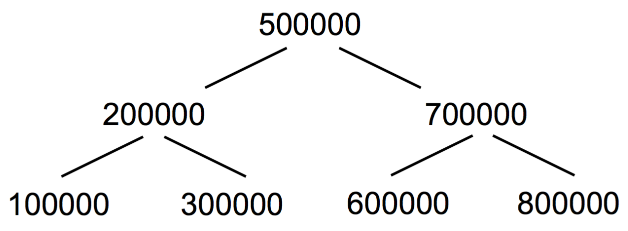
Change the displayInOrder() method to displayPreOrder(), which displays the tree rooted at "this" node using a recursive preorder traversal. The output is in the format of the following example:

```
500000
    200000
        100000
        300000
    700000
        600000
        800000
```

In the above example:

- 500000 is the root of the tree (hence preorder – the root of each subtree is displayed first),
- 200000 is the left child of the 500000 node,
- 100000 is the left child of the 200000 node,
- 300000 is the right child of the 200000 node,
- 700000 is the right child of the 500000 node,
- etc.

To clarify, this example could also be drawn as:



Note that displayPreOrder() indents each line of output depending on the level of the tree where that number resides. Accomplish this the same way we did in the Module 7 Assignment – by using a String parameter like this:

```
public void displayPreOrder(String indent)
```

For every number displayed, include the current value of `indent` in front of the number.

Increase the length of `indent` with each recursive call by calling:

```
somevariable.displayPreOrder(indent+"    ");
```

As the final change to the BinaryTreeNode class, modify the recursive insert() method. The supplied version of the insert() method permits duplicate names to be added, but this is not appropriate in the case of id numbers, which are supposed to have no duplicates.

Change the insert() method so it will only insert unique id numbers into the tree. Any attempt to insert a number that is already in the tree should be detected and the duplicate number should not be inserted.

To accomplish this, before checking to see where a given id number should be inserted (that is, left versus right), the insert() method must check to see if the given id number matches the number in "this" node. If so, throw an IllegalArgumentException with a message indicating that a duplicate number has been found, for example:

```
Duplicate found: Number 779653 is already in the tree.
```

Modify the **BinaryTree** class (provided for download earlier in this module) as follows:

- Make any updates required for the change from 'name' to 'idNumber';
- Add the clause '`throws IllegalArgumentException`' to the first line of the insert() method. This tells Java that exceptions may arise from calling `root.insert(...)`, and that such exceptions should be passed back to the main() method to be handled there;
- Change the displayInOrder() method to displayPreOrder(). Inside this method, call `root.displayPreOrder("")`. In other words, supply a string of length zero as the initial value for `indent`. Modify the "Beginning of" and "End of" messages appropriately.

Write a **DuplicateNumbers** class with a main method that tests your modified BinaryTree and BinaryTreeNode classes as follows.

First, create a BinaryTree instance and loop to insert each number from the following array:

```
int[] array = {500000, 200000, 100000, 300000, 700000, 600000, 800000};
```

Include the call to the insert() method within a try-catch block so you can catch any instances of IllegalArgumentException (which should not happen with this test, since there are no duplicates).

Call displayPreOrder() with the resultant tree and make sure the display looks exactly like this:

```
500000
    200000
        100000
        300000
    700000
        600000
        800000
```

This test is to confirm that your insert() and displayPreOrder() methods work properly.

Then randomly generate a series of twenty id number values. Store these numbers in an array. These numbers are NOT to be sorted. Your program will generate a different set of numbers each time you run it.

Once that is done, have your software randomly pick one position within the array (that is, a number between 0 to 19). Copy (not swap) the number in that position into three other randomly selected positions in the array. The effect of this is to ensure that the array includes a few duplicate numbers, in randomly chosen positions.

Create a new BinaryTree instance. Loop through your twenty test values, using the insert() method to attempt to add each number to the tree.

Once all the insertion attempts are completed, display a count of how many numbers were added to the tree and use the displayPreOrder() method to display the contents of the tree.

Executing the DuplicateNumbers class should produce output in the format shown on the next page.

NOTE: You will notice that my example lists the twenty test values in four rows of five numbers. This is not strictly necessary; you can simply list one number per line. If you wish to emulate my approach, you can do so in a couple of different ways:

- You could do this with nested loops; or
- Within a single loop, display each number with a 'print' statement. Include an 'if' statement in the loop to detect when to execute a 'println()' statement to move to a new line. Use the % operator to help identify every fifth index value.

```
*** Beginning of preorder display ***
500000
    200000
        100000
        300000
    700000
        600000
        800000
****** End of preorder display ******

***** Array of 20 numbers (100000-999999) *****

347281    108456    611311    805785    779653
311154    652552    779653    779653    852375
219748    428022    281663    209832    712051
221893    692554    193631    580914    341877

**************** End of Array ****************

Inserting Values in the Tree:
Duplicate found: Number 779653 is already in the tree.
Duplicate found: Number 779653 is already in the tree.
A total of 18 numbers were added.

*** Beginning of preorder display ***
347281
    108456
        311154
            219748
                209832
                    193631
                281663
                    221893
            341877
    611311
        428022
            580914
        805785
            779653
                652552
                    712051
                        692554
            852375
****** End of preorder display ******
```

NOTE: Notice that the first number in the array is shown in the preorder display as the root of the entire tree. The second number is a direct descendant of the root, and so on.

Submission Instructions

Include the following in your submission document:

- Your name, student number, CS1083, Module 11 Assignment, and the date
- Complete code for your modified BinaryTreeNode and BinaryTree classes, plus your DuplicateNumbers class
- Two sets of output from executing DuplicateNumbers

D2L DROPBOX SUBMISSION INSTRUCTIONS

Remember to save / export your submission document as a pdf file, and then upload that one pdf file to D2L as follows:

1. In the top-navigation bar on the course screen, select 'Assessments' and then 'Assignments'.
2. Select the assignment title and follow the instructions to upload your submission document.