**CS 1450 Data Structures and Algorithms – Fall 2021**

**Assignment #10**

Due Date: **Dec 1, 2021** at 1:40pm (MW class), **Dec 2, 2021** at 12:15pm (TR classes)

Purpose: Learning to create and manipulate a binary tree.

Effort: Individual

Points: **100**

Deliverables: Upload a **.zip** file with **ONLY** your source code (.java file) to Canvas by due date.

**Copy your code and design notes into your OneNote notebook.**

**The Situation**

The Parrots are at the airport in Japan waiting to fly to New York. While they are waiting, the parrots have perched in an ornamental tree in the departure lounge and are singing to the other travelers. They’ve been practicing a special song where each parrot knows a phrase of the song. If you solve this assignment, they will sing their special song to you.

**Assignment Description**

This assignment provides the opportunity to implement a binary search tree and to implement a level-order traversal of the tree. The tree will be filled with parrots and when a level order traversal is performed the parrots will sing a song.

The first step is to fill a binary search tree with parrots. Each parrot has an ID number, name, and one phrase that is part of a song. The parrots are placed in the binary tree using their IDs as keys. After the tree is filled, traverse the tree in level-order, and have each parrot sing its part of the song as you visit it. Finally, list the names of the parrots who are at the leaves of the tree (going from left to right).

The complete program contains the following classes:

* 1 Parrot
* 1 Binary Tree
* 1 TreeNode (inner class of the Binary Tree)

**Specifications**

1. Create a Java class called **LastNameFirstNameAssignment10**
2. Follow "CS1450 Programming Assignments Policy"
3. Include in your **design notebook** the following tasks:
   1. First, draw a binary search tree from the input given in *Figure 1* below.
   2. Second, do a **level-order** traversal of the binary search tree diagram in *Figure 2* below.
   3. Third, translate the key sequence from that traversal into words using decoder in *Figure 3.*
4. Write a test program (i.e. main) to do the following:
   1. Create an instance of your binary search tree ADT class: **BinaryTree**
   2. Create parrots and add them to the binary tree
      1. For each parrot in the file:
         * Read the parrot’s ID, name, and song word
         * Create a parrot object
         * Add the parrot object to the binary tree following the **binary search tree property using** the id as the key.
   3. Traverse the binary tree in level order. As you reach each parrot, **print its phrase** in the song.
   4. Visit the leaf nodes, **print each parrot’s name**.

**Design Notebook Binary Search Tree Traversal Exercises**

Before you start coding the parrots, here are some binary search tree exercises.

Include these in your design notebook.

1. **Draw a Binary Tree**

In your design notebook draw a binary search tree created from these keys:

**first** **last**

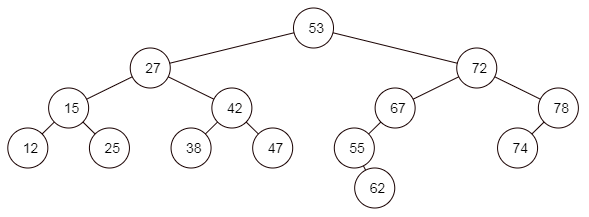
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 50 | 72 | 68 | 26 | 98 | 12 | 8 | 40 | 57 | 99 | 20 | 42 | 35 | 81 | 29 |

*Figure 1. Keys to construct a binary tree from*

Starting on the left at 50, insert each number into the tree following the binary search tree property. The tree should start with 50 at the root, and the nodes immediately below 50 will be 26 on the left and 72 on the right.

1. **Level Order Traversal of a Binary Tree**

Perform a level-order traversal of this tree.



*Figure 2. Binary tree for level-order traversal*

1. **Translate Level Order Traversal into Words**

To make it more interesting, a traversal can be decoded into words. The decoder is in *Figure 3*. Each key in the tree maps to a different letter in the decoder. For example, when you see key 53 in the tree, it will match to an ‘N’ in the decoder.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Key** | 53 | 27 | 15 | 12 | 25 | 42 | 38 | 47 | 72 | 67 | 55 | 62 | 78 | 74 |
| **Letter** | N | e | e | i | v | r | e | - | v | - | u | ! | g | p |

*Figure 3: Decoder for level-order keys*

In your design notebook:

* Traverse the tree in *level-order*, writing down the numbers as you go.
* Decode the message associated with the level order traversal.

Now, on to coding the parrots in a binary search tree!

**Classes**

**Parrot Class**

* Description
  + Represents one parrot.
* Private Data Fields
  + **id** – integer for the parrot’s ID number
  + **name** – string for the parrot’s name
  + **songPhrase** – string for the parrot’s phrase in the song
* Public Methods
  + Constructor
    - Initializes all private data fields with incoming values read from file.
  + Getters
    - For **name** field
  + Setters
    - None
  + ***public string sing()***
    - Returns string containing parrot’s songPhrase
  + ***public int compareTo(Parrot otherParrot)***
    - Returns result that indicates whether this parrot’s ID is less than, equal to, or greater than the ID of the parrot it is being compared to.
    - Used when adding parrots to the tree.

**BinaryTree Class**

* Description
  + A binary search tree constructed from nodes.
  + Each node contains a parrot. The parrot’s ID is used as the node’s key.
* Private Data Fields
  + **root** – reference to the root node of the binary search tree.
* Public Methods
  + Constructor
    - Initializes root reference to null
  + ***public boolean insert(Parrot parrotToAdd)*** 
    - Adds a parrot to tree maintaining the binary search tree property.
    - Returns true if parrot was added, false if parrot was not added.
    - See the FAQ for more details.
  + ***public void levelOrder()***
    - Traverses the tree in level-order, printing each parrot’s phrase as it is visited.
    - As you move through the levels in the tree you will need to use a queue to keep track of the nodes in the next level down.
    - See the FAQ for more details.
  + ***public void visitLeaves()***
    - This method only calls the **private** recursive method ***visitLeaves***
    - Because main doesn’t have access to root, this method is called in main.
    - Because this method is in the BinaryTree class, it has access to the root and therefore can call recursive ***visitLeaves*** method sending it the root.
* Private Methods
  + ***private void visitLeaves(TreeNode aNode)***
    - **Must be recursive** (this is the recursive helper method).
    - Visits leaf nodes in left to right order (left-most leaf first, then the one to its right, and so-on to the right-most leaf in the tree.)
    - At each leaf, print the name of the parrot at that leaf.
* Private Inner Class
  + **TreeNode**
    - Description
      * Represents **one node** in the binary search tree.
      * A node stores a parrot in the data field and two references to the nodes beneath it in the tree.
    - Private Data Fields
      * **parrot** – the data stored in a tree node will be a parrot object
      * **left** – reference to the tree node below and to the left of this tree node
      * **right** – reference to the tree node below and to the right of this tree node
    - Public Methods
      * Constructor
        + Initializes parrot to incoming parrot
        + Initializes left and right references to null

**Test File Information**

The file **parrots.txt** contains one line for each parrot. Again, this is a test file, do NOT ASSUME that the order or number of parrots will be the same in the file used for grading.

The format of each line in the file is as follows:

id name song phrase

**102 Pollymorphic Dashing through**

**Moving the file data into the tree:**

* In the file, the parrots are in the order that they must be inserted into the tree.
* The first parrot in the file is the root parrot at the top of the tree.
* The subsequent parrots must be read from the file and added to the tree one-by-one in the order they appear in the file.
* An extra test file called **parrotsTest.txt** is provided **ONLY** to test code on a smaller data set. It contains:

19 Nola Jingle bells,

15 Red jingle

22 Dora bells,

17 Pollymorphic all the

24 Tiki way

6 Rudolph jingle

Using these values:

* The level order traversal produces: ***Jingle bells, jingle bells, jingle all the way****.*
* The names of the parrots at the leaves from left to right: ***Rudolph, Pollymorphic, Tiki***
  + Use this file to test your code on a smaller data set.

**Tips**

**Must Do**

* Create your own **BinaryTree** ADT
  + Create a private inner **TreeNode** class that contains a parrot as the data field
  + You do not need to create a generic binary tree or node.

**Binary Search Tree Property**

* In this assignment, the key to compare nodes will be the parrot’s id.
* For example, if a node contains **id** = 24, all the nodes in its left sub-tree must have keys **ids** less than 24, and all nodes in its right sub-tree must have **ids** greater than 24
* You may assume that **ids** are unique: there are no duplicate **ids**.

**FAQ Document**

* I provided a Binary Tree FAQ document. This document contains several hints so if you are trying to figure things out on your own be aware that looking at the hints might give things away. 😊

**Output**

**parrotsTest.txt:**

* File to test your code with less nodes – this does not need to be turned in – it is for testing on a smaller data set.
* Use these values to draw a tree and then walk through your code when debugging.
* The parrot’s “test” song is: **Jingle bells, jingle bells, jingle all the way.**
* Output for test file:

Parrot Christmas Song

---------------------

Jingle bells, jingle bells, jingle all the way

Parrots on Leave Nodes

----------------------

Rudolph

Pollymorphic

Tiki

**parrots.txt**:

* If your code worked properly on the test file, then running your code on *parrots.txt* will reveal the parrot’s song. 😊
* When you hand in your code, this is the only file your code needs to run.