Lab 15: Trees

Implement a Tree

Specifications

- Read all of these instructions carefully. Name things exactly as described.
- Do all your work in a public repository (matching the example provided by your instructor) called `data-structures-and-algorithms`, with a well-formatted, detailed top-level README.md
- Create a branch in your repository called `tree`
- On your branch, create...
  - *C#*: Create a class named `Tree.cs`
  - *JavaScript*: a folder named `tree` which contains a file called `tree.js`
  - *Python*:a folder named `tree` which contains a file called `tree.py`
  - *Java*: a package named `tree` which contains a file called `Tree.java`

- Include any language-specific configuration files required for this challenge to become an individual component, module, library, etc.
  - *NOTE: You can find an example of this configuration for your course in your class lecture repository.*

Features

- Create a Node class that has properties for the value stored in the node, the left child node, and the right child node.
- Create a BinaryTree class
  - Define a method for each of the depth first traversals called `preOrder`, `inOrder`, and `postOrder` which returns an array of the values, ordered appropriately.

- At no time should an exception or stack trace be shown to the end user. Catch and handle any such exceptions and return a printed value or operation which cleanly represents the state and either stops execution cleanly, or provides the user with clear direction and output.

- Create a BinarySearchTree class
  - Define a method named `add` that accepts a value, and adds a new node with that value in the correct location in the binary search tree.
  - Define a method named `contains` that accepts a value, and returns a boolean indicating whether or not the value is in the tree at least once.

Structure and Testing

Utilize the Single-responsibility principle: any methods you write should be clean, reusable, abstract component parts to the whole challenge. You will be given feedback and marked down if you attempt to define a large, complex algorithm in one function definition.

Write tests to prove the following functionality:

1. Can successfully instantiate an empty tree
2. Can successfully instantiate a tree with a single root node
3. Can successfully add a left child and right child to a single root node
4. Can successfully return a collection from a preorder traversal
5. Can successfully return a collection from an inorder traversal
6. Can successfully return a collection from a postorder traversal

Ensure your tests are passing before you submit your solution.

Stretch Goal

Create a new branch called `k-ary tree`, and, using the resources available to you online, implement a k-ary tree, where each node can have any number of children.

Documentation: Your README.md

```
# Trees
<!-- Short summary or background information -->

## Challenge
<!-- Description of the challenge -->

## Approach & Efficiency
<!-- What approach did you take? Why? What is the Big O space/time for this

## API
<!-- Description of each method publicly available in each of your trees --
```

Submission Instructions

1. Create a pull request from your branch to your `master` branch
2. In your open pull request, leave as a comment a checklist of the specifications and tasks above, with the actual steps that you completed checked off
3. Submitting your completed work to Canvas:

1. Copy the link to your open pull request and paste it into the corresponding Canvas assignment
2. Leave a description of how long this assignment took you in the comments box
3. Add any additional comments you like about your process or any difficulties you may have had with the assignment

4. Merge your branch into `master`, and delete your branch (don't worry, the PR link will still work)