P09 Pokémon Catalog

Overview

For this assignment, you will implement a simple Pokémon Catalog using a Binary Search Tree (BST). If you're unfamiliar with Pokémon from several decades of popular video games and trading card games and a Japanese animated series with over 1100 episodes (as of this writing), there's a basic introduction to them here.

Our Pokémon Catalog is a tree-based database that contains the name of a Pokémon and a numeric value representing its Combat Power (CP). The CP value is calculated based on the following stats:

- Attack (**A**)
- Stamina (**S**)
- Defense (**D**)

There's some complicated official equation to convert these stats to CP, but we're going to simplify it as:

$$CP = 100A + 10S + D$$

With the assumption that each of **A**, **S**, and **D** are integers between 1-9. Our BST will store Pokémon by their CP values, helping you learn how BSTs facilitate insertion and retrieval operations from a collection of ordered elements, in an easy and elegant way.

Name: Pikachu
Attack: 1
Stamina: 2
Defense: 3
CP: 123

Grading Rubric

5 points	**Pre-assignment Quiz**: accessible through Canvas until 11:59PM on **11/30**.
20 points	**Immediate Automated Tests**: accessible by submission to Gradescope. You will receive feedback from these tests *before* the submission deadline and may make changes to your code in order to pass these tests. Passing all immediate automated tests does **not** guarantee full credit for the assignment.
25 points	**Additional Automated Tests**: these will also run on submission to Gradescope, but you will not receive feedback from these tests until after the submission deadline.

Learning Objectives

The goals of this assignment are:

- Implement a Binary Search Tree data structure
- Practice working with object-oriented design and exception handling
- Gain more experience developing tests for your code

Additional Assignment Requirements and Notes

Keep in mind:

- You can NOT have any additional import statements besides relevant exceptions.
- You can NOT define any additional class or instance fields beyond the ones detailed in this specification.
- You are allowed to define any **local** variables you may need to implement the methods in this specification.
- You are allowed to define additional **private** helper methods to help implement the methods in this specification.
- You are allowed to define additional public boolean class methods in your **tester class only**.
- All methods, public or private, must have their own Javadoc-style method header comments in accordance with the CS 300 Course Style Guide. Additionally, every class must also have its own Javadoc-style class header comment.
- Any source code provided in this specification may be included verbatim in your program without attribution.

CS 300 Assignment Requirements

You are responsible for following the requirements listed on both of these pages on all CS 300 assignments, whether you've read them recently or not. Take a moment to review them if it's been a while:

- Academic Conduct Expectations and Advice, which addresses such questions as:
  - How much can you talk to your classmates?
  - How much can you look up on the internet?
  - What do I do about hardware problems?
  - and more!
- Course Style Guide, which addresses such questions as:
  - What should my source code look like?
  - How much should I comment?
  - and more!

Getting Started

1. Create a new project in Eclipse, called something like **P09 Pokemon Catalog**.
   a. Ensure this project uses Java 11. Select "JavaSE-11" under "Use an execution environment JRE" in the New Java Project dialog box.
   b. Do **not** create a project-specific package; use the default package.

2. **Download** and add the following source files to that project's src folder:
   a. Pokemon.java
   b. PokemonCatalog.java
   c. PokemonTree.java
   d. PokemonTreeTester.java

3. Create one (1) additional Java source file within that project's src folder:
   a. PokemonNode.java (does NOT include a main method)

Implementation Requirements

The only class you will need to create yourself is the **PokemonNode** class; it will function as a wrapper for Pokemon objects in our data structure, like LinkedOrder from P07 did for Orders, and LinkedNode from P08 did for Messages. Be sure to implement *carefully* as specified below.

PokemonNode

Your **PokemonNode** class must contain ONLY the following instance fields:

- `private Pokemon data;` `// data field of this PokemonNode`
- `private PokemonNode leftChild;` `// reference to the left child`
- `private PokemonNode rightChild;` `// reference to the right child`

Additionally, your **PokemonNode** class must contain the following instance methods:

- `public PokemonNode (Pokemon data)`
  - A one-argument constructor which sets leftChild and rightChild to null and initializes the data field.
  - Throws an IllegalArgumentException if `data` is null
- `public PokemonNode getLeftChild()`
  - Returns a reference to the left child of this PokemonNode
- `public PokemonNode getRightChild()`
  - Returns a reference to the right child of this PokemonNode
- `public Pokemon getPokemon()`
  - Returns a reference to the Pokemon contained in this PokemonNode
- `public void setLeftChild(PokemonNode left)`
  - Sets the left child of this PokemonNode (null values allowed)
- `public void setRightChild(PokemonNode right)`
  - Sets the right child of this PokemonNode (null values allowed)

Once you have completed this class, read through the provided source code and its comments for the rest of your program.

Pokemon

The **Pokemon** class is provided in its entirety and represents an individual Pokémon in our catalog. For simplicity, we consider only two fields (name and CP). Notice the CP is final! This will be our lookup key in the Pokemon Catalog.

Notice too that the Pokemon class implements the Comparable interface -- you can call the method `compareTo()` to compare two Pokémon with respect to their CP.

You should NOT edit (or submit) the Pokemon.java file.

PokemonTree and PokemonTreeTester

Once you've completed PokemonNode and familiarized yourself with the Pokemon class, you're ready to begin the actual Binary Search Tree.

Make sure you've downloaded PokemonTree and PokemonTreeTester and added them to your project. Notice that we added default return statements to the incomplete methods to simply let the code compile. You will complete the implementation of all the methods defined in these classes; pay attention to the TODO tags, as well as the details provided in their javadoc method headers.

Remember that you MAY NOT add any additional fields (either instance or static) to PokemonTree, and NO additional public methods may be added to this class. Read all the details provided in the javadoc method headers carefully!

If a method is described to be a **recursive** helper method, you are NOT allowed to implement it using iteration (for or while loops). They MUST be designed and implemented using **recursion**.

Note that even though it is recommended to declare all helper methods to be private, we set some of them to be public so that we can call them from our automated test methods. Do not change these access modifiers, as this will cause your program to fail automated tests!

Implementation tips:

- You can compare two Pokemon objects by calling the Pokemon.compareTo() method.
- Pokemon with the same CP value are NOT allowed in this tree.
- Note also that in our PokemonTree, the increasing order of the catalog goes from the least powerful to the most powerful Pokemon. All the Pokemon *less* powerful than the Pokemon in a given node must go in the subtree rooted at its **left** child. Whereas all the Pokemon *more* powerful than the Pokemon in the current node must go in the subtree rooted at its **right** child.

You are responsible for testing your implementation of the PokemonTree public methods thoroughly. You must implement at least the FIVE unit test methods defined in the PokemonTreeTester class. Make sure to test every method with at least 3 different scenarios. Hints are provided in the provided javadocs method headers.

PokemonCatalog

The **PokemonCatalog** class is provided in its entirety and contains a driver to demonstrate how you can use our PokemonTree to build a Pokemon Catalog. Please carefully read through the code to understand how it works.

Sample output for this driver program is contained in the following section.

Sample Output

Running the provided PokemonCatalog class with a properly-implemented Binary Search Tree will yield the following output:

```
Size: 0 Height: 0
Catalog:

============================================================
Size: 2 Height: 2
Catalog:
[Pikachu CP:123 (A:1 S:2 D:3)]
[Eevee CP:224 (A:2 S:2 D:4)]

============================================================
Size: 3 Height: 3
Catalog:
[Pikachu CP:123 (A:1 S:2 D:3)]
[Eevee CP:224 (A:2 S:2 D:4)]
[Snorlax CP:448 (A:4 S:4 D:8)]

The Least Powerful Pokemon: [Pikachu CP:123 (A:1 S:2 D:3)]
The Most Powerful Pokemon: [Snorlax CP:448 (A:4 S:4 D:8)]
============================================================
Size: 5 Height: 4
Catalog:
[Pikachu CP:123 (A:1 S:2 D:3)]
[Eevee CP:224 (A:2 S:2 D:4)]
[Charmander CP:321 (A:3 S:2 D:1)]
[Snorlax CP:448 (A:4 S:4 D:8)]
[Mewtwo CP:999 (A:9 S:9 D:9)]

The Least Powerful Pokemon: [Pikachu CP:123 (A:1 S:2 D:3)]
The Most Powerful Pokemon: [Mewtwo CP:999 (A:9 S:9 D:9)]
============================================================
Size: 9 Height: 6
Catalog:
[Pikachu CP:123 (A:1 S:2 D:3)]
[Bulbasaur CP:223 (A:2 S:2 D:3)]
[Eevee CP:224 (A:2 S:2 D:4)]
[Squirtle CP:312 (A:3 S:1 D:2)]
[Charmander CP:321 (A:3 S:2 D:1)]
[Snorlax CP:448 (A:4 S:4 D:8)]
[Lapras CP:735 (A:7 S:3 D:5)]
[Rayquaza CP:823 (A:8 S:2 D:3)]
[Mewtwo CP:999 (A:9 S:9 D:9)]
```

```
The Least Powerful Pokemon: [Pikachu CP:123 (A:1 S:2 D:3)]
The Most Powerful Pokemon: [Mewtwo CP:999 (A:9 S:9 D:9)]
============================================================
Lookup query: search for the Pokemon whose CP is 123
Search Results: [Pikachu CP:123 (A:1 S:2 D:3)]
Lookup query: search for the Pokemon whose CP is 256
No search results.
```

Assignment Submission

Once you're satisfied with your work, both in terms of adherence to this specification and the academic conduct and style guide requirements, submit your source code through Gradescope.

For full credit, please submit **ONLY** the following files (source code, *not* .class files):

- PokemonNode.java
- PokemonTree.java
- PokemonTreeTester.java

Your score for this assignment will be based on the submission marked "**active**" prior to the deadline. You may select which submission to mark active at any time, but by default this will be your most recent submission.

Copyright Notice