CSE 595 : Programming Abstractions – Homework III

Dr. Ritwik Banerjee
Computer Science, Stony Brook University

This homework document consists of two sections. Both sections are asking you to implement the same basic idea, but once in Java, and then in Python. In this assignment, the objective is to learn how iteration can be programmed as an abstraction (in a statically typed object-oriented language, as well as in a dynamically typed one). Plus, you will also learn about how to use encapsulation using decorators in Python, which does not provide the traditional mechanisms of encapsulation (as Java does).

Environment: It is highly recommended that you use IntelliJ IDEA for the Java code, and PyCharm for your Python code. You can avail them from `https://www.jetbrains.com/idea/` and `https://www.jetbrains.com/pycharm/`, respectively. You can, if you really want, use a different IDE, but if things go wrong with your development environment there, you may be on your own.

Programming Language: Starting with this homework, and for the remainder of this course, all Java code *must* be JDK 1.8 compliant. That is, you may have a higher version of Java installed, but the "language level" must be set to Java 8. This can be easily done in IntelliJ IDEA by going to "Project Structure" and selecting the appropriate "Project language level". *This is a very important requirement, since Java 9 and beyond all have additional language features that will not compile with a Java 8 compiler.*

For Python, you must use Python version 3.4 or above. In particular, make sure that you are NOT using Python version older than 3.x.

1 Iteration abstraction in Java

In Java, iteration can be performed on any collection of objects that is deemed "iterable". This is specified through the implementation of the `Iterable` interface. This interface mandates the implementation of the following method:

```
public Iterator<T> iterator();
```

All the standard collections you have used so far (*e.g.*, lists, sets) implement this interface. As long as a class implements iterable, we can iterate over an instance of that class using the standard foreach construct:

```
Iterable<T> iterableOfTs = new ...;
for (T t : iterableOfTs) {
    // body of the loop
}
```

Now, we are going to combine your knowledge of basic structures with this abstraction, so that the "client" (i.e., someone using your data structure) can iterate over the data without bothering about the details of the structure itself.

For the standard collections, this abstraction is already implemented as part of the core Java library. So, we will implement it over binary trees.

1. Implement a class called `BinaryTree`. The class should have a node-and-pointer implementation similar to how a linked list is typically implemented: every node should have an instance field to store its own data, and two instance fields pointing to its left and right children. (8)

2. Your `BinaryTree` class must have the following constructors to allow for the creation of an empty tree, and the creation of a tree with a single node containing a specific value: (8)

```
BinaryTree<String> t1 = new BinaryTree<>();
BinaryTree<String> t2 = new BinaryTree<>("value at the root node of t2");
```

   Here, I am using the `String` parameter simply as an example (it could just as well have been another parameter).

3. Your `BinaryTree` class must have two methods, `addLeftChild` and `addRightChild`. This method must be designed such that the following code, which adds a subtree, works properly: (10)

```
    BinaryTree<String> t1 = new BinaryTree<>("root value");
    BinaryTree<String> t2 = new BinaryTree<>("left child of root");
    BinaryTree<String> t3 = new BinaryTree<>(); // leaf node with null value
    t2.addRightChild(t3);
    t1.addLeftChild(t2);
```

4. Implement a private nested class called `BinaryTreeIterator`, which implements the `Iterator` interface. An *iter**able*** needs an *itera**tor***, and this is the one you will use for your binary tree implementation. Think carefully about whether or not this nested class should be static or non-static. Also implement the `iterator()` method in your binary tree class, which is required by the `Iterable` interface. There are, of course, many different ways of iterating over a binary tree. For this homework, we will use breadth-first traversal as our iteration strategy. That is, the iteration will start at the root node, and move left-to-right at each depth before moving to the next level one level deeper. This is shown in Figure 1 below. You will realize that the `iterator()` method may become remarkably simple, once the hard work has been done by the iterator object you have created. (24)

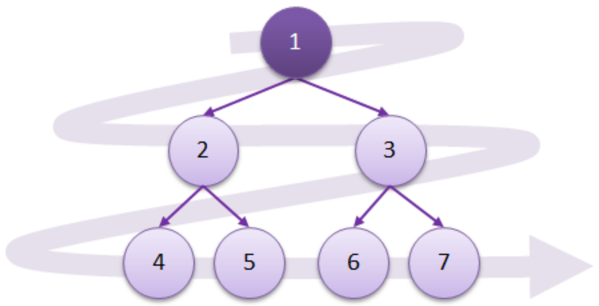

Figure 1: Breadth-first traversal on a binary tree parameterized by integers.

2 Iteration abstraction in Python

Just like with Java code, a collection of objects in a data structure is not automatically iterable in Python. We need to make the collection `Iterable` and also create an `Iterator` for it. We will use the same binary tree idea from the first section of this homework, and work through how to make your own object iterable.

5. Implement a class called `BinaryTree` in a file called `binarytree.py`. This class should be able to construct an empty binary tree or a binary tree with a value. For example, (6)

```
    t1 = BinaryTree()
    t2 = BinaryTree("value")
```

should both be valid.

6. Your binary tree class must have two methods `add_leftchild` and `add_rightchild`, to add left and right children to a tree. For example, the following should work: (8)

```
t1 = BinaryTree(0)
t1.add_leftchild(BinaryTree(5))
t2 = BinaryTree(10)
t2.add_rightchild(BinaryTree(23))
t1.add_rightchild(t2)
```

7. Unlike Java, Python is dynamically typed. So, the value stored in one node may not be of the same type as the value stored in another node in the same tree. In other words, we can create a tree without a single parameter. Often, this is an advantage. But, our goal is to iterate over the tree . . . and typically, when we iterate over a collection, we want to repeatedly do similar actions with each item (think of the `map()` function, for example). Thus, we would like to impose the condition that every element in the tree must be the same type. If not, your code should raise a `TypeError`. For example: (8)

```
t1 = BinaryTree(0)
t1.add_leftchild(BinaryTree(5))
t2 = BinaryTree(10)
t2.add_rightchild(BinaryTree("not an int"))
t1.add_rightchild(t2)
```

should raise crash the code with the following error message:

```
TypeError: Type mismatch between int and str
```

Pay careful attention to the error message. Your error message must be the exact same format. In particular, the order of the `int` and `str` in the error message above is important.

8. The value in the tree must be a private variable. Write your code using `@property` and `@data.setter` decorations properly, such that the instance variable is private but the following piece of code works and prints `5 0`: (8)

```
t1 = BinaryTree(0)
t1.add_leftchild(BinaryTree(5))
print(t1.left.data, t1.data)
```

9. Next, let us ensure that we can use a for loop on our tree, or iterate in other similar ways. The first step is to make our class iterable. For this, we need to implement the `__iter__()` function. Implement this function such that iterating over a binary tree performs the preorder traversal. For example, the following code should print `[0, 5, 10, 22]`. (20)

```
t1 = BinaryTree(0)
t1.add_leftchild(BinaryTree(5))
t2 = BinaryTree(10)
t2.add_rightchild(BinaryTree(22))
t1.add_rightchild(t2)
print(list(iter(t1)))

# we could have also used a for-loop as follows
# for item in t1:
#     print(item)
```

It is extremely important to note one difference between your Python implement here, as opposed to what you did in Java. In Python, you should use the `yield` statement instead of creating a separate iterator object. Python allows you to create such an object, but using the `yield` statement is a strict requirement for this assignment.

NOTES:

- **Uncompilable code** will not be accepted.
- Please remember to verify what you are submitting. Make sure you are, indeed, submitting what you think you are submitting!
- What to submit?
  A single `.zip` file consisting of the single Java file `BinaryTree.java` and your Python file `binarytree.py`.

Submission Deadline: May 4, 2021, 11:59 pm