ACS-2947-002/050
Assignment 3
Due by Friday, March 22, 11:59 PM

Instructions

- Submit your `.java` files (together in an `Assign3.zip` file) via Nexus.
- Include your name and student number as a comment in every file.
- Document the classes using Javadoc notation.
  Include comments as needed and use exception handling where necessary

PART A – Trees (40 marks)

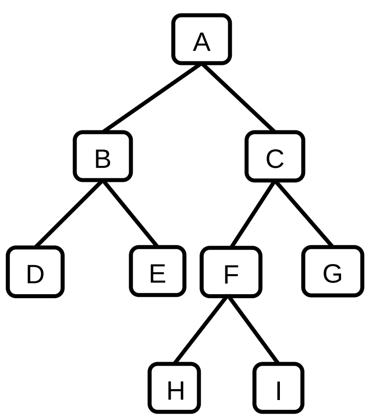
1. Using your Lab 6 `LinkedBinaryTree` implementation, add a position and element iterator.
   a. Have the `Tree` interface extend `Iterable`. Add the abstract methods `iterator()` and `positions()` that return `Iterator<E>` and `Iterable<Position<E>>`, respectively.

   b. Add the nested class and methods from your notes/text.
      - `ElementIterator`, `iterator()`, `positions()` and methods required for your tree traversals.

      * Note that your classes require the `java.util.Iterator` package

   c. Override the toString method to display a tree in the following format given the structure:

Structure:



Output:

```
- A
    - B
        - D
        - E
    - C
        - F
            - H
            - I
        - G
```

2. Create an interactive program that asks the user to work through a Yes/No decision tree with **a height of at least 3 and at least 10 nodes**. **You must come up with a meaningful decision tree of your own.** Name your driver class: `PartA_Driver`. Also, display the content of the tree using the toString method.

An example with the decision tree from your notes is given below.

Sample output:

```
Tree
-----
. . .  (Print tree as described in 1c)

Tree Interaction
-----
Are you nervous? (yes/no)
no
Will you need to access most of the money within the next 5 years?
(yes/no)
no
Are you willing to accept risks in exchange for higher expected returns?
(yes/no)
yes
Final Decision: Stock portfolio
```

3. In the same driver class, create a binary tree for the following arithmetic expression:
   `(((5 – 2) + (4 x (8 - (3 + 1)))) + (9 x ((3 x 6) – (7 + 2)))) x 10;`
   then evaluate the value of the arithmetic expression tree. **Your evaluation must be based on the expression tree**. Your implementation should:
   a. Display the inorder representation of the tree
   b. Display the postorder representation of the tree
   c. Use a **stack** to evaluate the value of the tree based on the postorder representation of the expression. *You may assume that the tree has only integer numbers.*
      - Add our `LinkedStack` implementation (from class notes) for the provided `Stack` interface in your package.
      - Note that the operator **x** or * may be used interchangeably for multiplication.

Sample output:

```
Arithmetic Tree
---------------
Inorder: [5, -, 2, +, ..., 10]
Postorder: [5, 2, -, 4, ..., x]
Tree value: ...
```

Notes:

- Iterator implementation:

a. In your tree interface:
   - Extend Iterable and add 2 abstract methods: `iterator()` that returns `Iterator<E>` and `positions()` that returns `Iterable<Position<E>>`

b. In the AbstractTree class:
   - add the nested `ElementIterator` class from your notes. Implement the `iterator()` method by returning a new instance of `ElementIterator`

c. Add the code for 4 traversal algorithms in the appropriate class:
   - `preorder()` and its associated recursive private method
   - `postorder()` and its associated recursive private method
   - `breadthfirst()`
   - `inorder()` and its associated recursive private method
     - import `List` and `ArrayList` from Java Class Libraries. *Note that the provided List interface is intended for PartB of this assignment only.*
     - For Queue you can use one from `java.util`, or add one of our implementations from class in your package e.g. `ArrayQueue` from Lecture 5.

d. Implement the `positions()` method by returning an `Iterable<Position<E>>` from one of the above. Select your preferred default, and override `toString()` to return a simple list view of your tree in the default traversal order.

   - toString:
     - "indentation" of each item depends on its position's depth in the tree
     - your algorithm should work with any tree
     - test this with your tree from Lab 5

   - decision tree:
     - Build the tree by assigning/re-assigning positions as you go along.
     - Map your yes/no to left/right child, and work through the decisions until an answer is reached (external node)
     - Your code should work for any linked binary decision tree: starts at the root as the first question and advances to left/right depending on the user input i.e. do not hardcode questions/answers to work only with your tree

PART B (50 marks)

Implement the Priority Queue ADT using a heap. The heap will use your *ArrayList* implementation from Assignment 2 and a comparator. The List interface for your ArrayList is provided.

1. Create a class called `HeapPriorityQueue` (HPQ) that implements the given `PriorityQueue` and `Entry` interfaces. Include the `AbstractPriorityQueue` class from your note/textbook for your `HeapPriorityQueue` to extend.

2. Create a driver program to show you working with your priority queue in a simple simulation of an airline standby list implemented as follows:

a. Create a `Passenger` class that stores a Passport number, a Fare code, a Flyer status code, and the Timestamp when the Passenger object was created.
b. A fare code can be one of three values *`Full, Disc, Buddy`* and priority is given to passengers in this order. Create an enum for the fare code and include a method called `randomValue` that randomly chooses one of these values.
c. A flyer status code can be one of four values *`Gold, Silver, Bronze, None`* and priority is given to passengers in this order. Create an `enum` for the fare code and include a method called `randomValue` that randomly chooses one of these values.
d. Given two passengers, they are ordered first by the fare code in the order *`Full, Disc, Buddy.`* If two passengers have the same fare code, they are ordered by the their flyer status in the order *`Gold, Silver, Bronze, None.`* If the fare codes and flyer statuses are the same, the first passenger (based on the timestamp) is ordered first.
e. Your simulation of the standby list should do the following:
    - Add 10 new passengers, whose passport numbers are provided by the user but the fare code and flyer status are randomly chosen.
    - Remove 5 passengers based on their priority.
    - Add 5 more passengers, whose passport numbers are provided by the user but the fare code and flyer status are randomly chosen.
    - Remove all passengers from the standlist based on their priority.

Notes:

- Before starting Part B, make sure that your `ArrayList` is fully implemented. You should have your `iterator` and `toString` in place and make your `ArrayList` dynamic.
- Make sure that a default comparator is included in your package.
- You may include a `toString` method in the `PQEntry` class that returns the value associated with an entry.
- Your priority queue will use `Passenger` details as key and their passport numbers as values.
- Have the Passenger class implement the `Comparable` interface for your default comparator to work. The default comparison is to order two Passengers by their passport numbers.
- For simplicity, use passport numbers with only 2 to 4 characters in your simulation.
- Import `Date` and `Timestamp` from `java.util` and `java.sql`, respectively.
- Both `Enum` and `Timestamp` already implement the `Comparable` interface.
- See sample output below.

3. In an `assign3.pdf` file, draw the resulting heaps for the first 10 Passengers' additions and the 5 removals **of the provided sample output below**. Explain the execution at each step by briefly describing all swaps that happen in the removal or addition of passengers and why they happen.
    - You only need to show the values (passport numbers) in your heaps.
    - You may draw one heap for each addition/removal but a simple explanation of all swaps that occur MUST be provided to match the sample output below.

- You may **clearly** draw the heaps on a plain paper (with no lines) and attach scans of your pages instead of typesetting the diagrams. **Unclear or unreadable diagrams will not be graded**.
- **Heaps with no explanations provided will not be graded**.
- **Note that this question and question 4 requires the diagrams of the heaps and not an array representation**.

4. Assuming that the **default comparator** is used for the same simulation in the sample output, draw the resulting heaps and explain the execution as described in question 3. *This question requires you to redo question 3 with the assumption that the default comparator is used.*

Sample Output

```
Enter passport no of new passenger: B8
Adding Passenger: (Passport: B8, Fare: Buddy, FlyerStatus: None, Time: 2024-
03-06 13:27:33.117)
Standby list: [B8]

Enter passport no of new passenger: A2
Adding Passenger: (Passport: A2, Fare: Buddy, FlyerStatus: Silver, Time:
2024-03-06 13:27:35.663)
Standby list: [A2, B8]

Enter passport no of new passenger: A4
Adding Passenger: (Passport: A4, Fare: Disc, FlyerStatus: Bronze, Time: 2024-
03-06 13:27:45.321)
Standby list: [A4, B8, A2]

Enter passport no of new passenger: B3
Adding Passenger: (Passport: B3, Fare: Full, FlyerStatus: Gold, Time: 2024-
03-06 13:27:50.302)
Standby list: [B3, A4, A2, B8]

Enter passport no of new passenger: C12
Adding Passenger: (Passport: C12, Fare: Buddy, FlyerStatus: Bronze, Time:
2024-03-06 13:28:02.971)
Standby list: [B3, A4, A2, B8, C12]

Enter passport no of new passenger: D3
Adding Passenger: (Passport: D3, Fare: Disc, FlyerStatus: Bronze, Time: 2024-
03-06 13:28:07.344)
Standby list: [B3, A4, D3, B8, C12, A2]

Enter passport no of new passenger: F4
Adding Passenger: (Passport: F4, Fare: Full, FlyerStatus: None, Time: 2024-
03-06 13:28:16.513)
Standby list: [B3, A4, F4, B8, C12, A2, D3]
```

```
Enter passport no of new passenger: G1
Adding Passenger: (Passport: G1, Fare: Buddy, FlyerStatus: Gold, Time: 2024-03-06 13:28:20.333)
Standby list: [B3, A4, F4, G1, C12, A2, D3, B8]

Enter passport no of new passenger: A9
Adding Passenger: (Passport: A9, Fare: Buddy, FlyerStatus: None, Time: 2024-03-06 13:28:27.818)
Standby list: [B3, A4, F4, G1, C12, A2, D3, B8, A9]

Enter passport no of new passenger: D7
Adding Passenger: (Passport: D7, Fare: Buddy, FlyerStatus: Bronze, Time: 2024-03-06 13:28:38.587)
Standby list: [B3, A4, F4, G1, C12, A2, D3, B8, A9, D7]

Passenger (Passport: B3, Fare: Full, FlyerStatus: Gold, Time: 2024-03-06 13:27:50.302) gets seated.
Standby list: [F4, A4, D3, G1, C12, A2, D7, B8, A9]

Passenger (Passport: F4, Fare: Full, FlyerStatus: None, Time: 2024-03-06 13:28:16.513) gets seated.
Standby list: [A4, G1, D3, B8, C12, A2, D7, A9]

Passenger (Passport: A4, Fare: Disc, FlyerStatus: Bronze, Time: 2024-03-06 13:27:45.321) gets seated.
Standby list: [D3, G1, A2, B8, C12, A9, D7]

Passenger (Passport: D3, Fare: Disc, FlyerStatus: Bronze, Time: 2024-03-06 13:28:07.344) gets seated.
Standby list: [G1, C12, A2, B8, D7, A9]

Passenger (Passport: G1, Fare: Buddy, FlyerStatus: Gold, Time: 2024-03-06 13:28:20.333) gets seated.
Standby list: [A2, C12, A9, B8, D7]

Enter passport no of new passenger:  F2
Adding Passenger: (Passport: F2, Fare: Buddy, FlyerStatus: Bronze, Time: 2024-03-06 13:28:45.606)
Standby list: [A2, C12, F2, B8, D7, A9]

Enter passport no of new passenger:  K2
Adding Passenger: (Passport: K2, Fare: Disc, FlyerStatus: Gold, Time: 2024-03-06 13:28:49.52)
Standby list: [K2, C12, A2, B8, D7, A9, F2]

Enter passport no of new passenger:  M1
Adding Passenger: (Passport: M1, Fare: Full, FlyerStatus: None, Time: 2024-03-06 13:29:13.133)
Standby list: [M1, K2, A2, C12, D7, A9, F2, B8]
```

```
Enter passport no of new passenger:  W01
Adding Passenger: (Passport: W01, Fare: Full, FlyerStatus: None, Time: 2024-03-06 13:29:30.116)
Standby list: [M1, W01, A2, K2, D7, A9, F2, B8, C12]

Enter passport no of new passenger:  A7
Adding Passenger: (Passport: A7, Fare: Buddy, FlyerStatus: None, Time: 2024-03-06 13:29:39.568)
Standby list: [M1, W01, A2, K2, D7, A9, F2, B8, C12, A7]

Passenger (Passport: M1, Fare: Full, FlyerStatus: None, Time: 2024-03-06 13:29:13.133) gets seated.
Standby list: [W01, K2, A2, C12, D7, A9, F2, B8, A7]

Passenger (Passport: W01, Fare: Full, FlyerStatus: None, Time: 2024-03-06 13:29:30.116) gets seated.
Standby list: [K2, C12, A2, B8, D7, A9, F2, A7]

Passenger (Passport: K2, Fare: Disc, FlyerStatus: Gold, Time: 2024-03-06 13:28:49.52) gets seated.
Standby list: [A2, C12, F2, B8, D7, A9, A7]

Passenger (Passport: A2, Fare: Buddy, FlyerStatus: Silver, Time: 2024-03-06 13:27:35.663) gets seated.
Standby list: [C12, D7, F2, B8, A7, A9]

Passenger (Passport: C12, Fare: Buddy, FlyerStatus: Bronze, Time: 2024-03-06 13:28:02.971) gets seated.
Standby list: [D7, B8, F2, A9, A7]

Passenger (Passport: D7, Fare: Buddy, FlyerStatus: Bronze, Time: 2024-03-06 13:28:38.587) gets seated.
Standby list: [F2, B8, A7, A9]

Passenger (Passport: F2, Fare: Buddy, FlyerStatus: Bronze, Time: 2024-03-06 13:28:45.606) gets seated.
Standby list: [B8, A9, A7]

Passenger (Passport: B8, Fare: Buddy, FlyerStatus: None, Time: 2024-03-06 13:27:33.117) gets seated.
Standby list: [A9, A7]

Passenger (Passport: A9, Fare: Buddy, FlyerStatus: None, Time: 2024-03-06 13:28:27.818) gets seated.
Standby list: [A7]

Passenger (Passport: A7, Fare: Buddy, FlyerStatus: None, Time: 2024-03-06 13:29:39.568) gets seated.
Standby list: []
```

Submission

Submit your **`Assign3.zip`** file that includes all the assignment files (`assign3.pdf`, `Position.java`, `Tree.java`, `AbstractTree.java`, `BinaryTree.java`, `AbstractBinaryTree.java`, `LinkedBinaryTree.java`, `Stack.java`, `LinkedStack.java`, `SinglyLinkedList.java`, `PartA_Driver.java`, `Entry.java`, `PriorityQueue.java`, `AbstractPriorityQueue.java`, `DefaultComparator.java`, `HeapPriorityQueue.java`, `Fare.java`, `FlyerStatus.java`, `Passenger.java`, `PassengerComparator.java`, `List.java`, `ArrayList.java`, `PartB_Driver.java`, any other class required for your implementation) via **Nexus**.

EXTRA WORK: Do not submit

Implement the Priority Queue ADT using a heap. The heap will use your ***LinkedBinaryTree*** (LBT) from PART A and a comparator.

1. Create a class called `LinkedHeapPriorityQueue` (LHPQ) that implements the given `PriorityQueue` and `Entry` interfaces. Include the `AbstractPriorityQueue` class from your textbook for your Linked Priority Queue to extend.

   Notes:
   - Make sure that your `LinkedBinaryTree` is fully implemented. You should have your tree traversal algorithms set and `toString()` in place.
   - First, have a good understanding of the array-based `HeapPriorityQueue` from your Part B. Here, the parameters are indices that represent the level number of each entry. With a linked tree-based PQ the parameters will be `Position` objects. Instead of using indices to access entries in the tree, we will determine the positions of these elements relatively.
   - Start building your LHPQ. Declare a `LinkedBinaryTree` called `heap` that holds `Entry` objects as its elements. Add the constructors in the same manner as your textbook `HeapPriorityQueue` (HPQ), and make sure that a `DefaultComparator` is included in your package. The next 5 protected utilities of HPQ are not required in the LinkedHeap version because all of this information can be either directly accessed or quickly determined via the LBT methods.
   - Next, look at the protected `swap` utility: instead of indices (int), you will have `Position` objects as parameters. In an ArrayList, you swap the *elements* in the given array indices. How would you swap the *elements* in given positions? Use this to form a basis of how to convert from array-based to LBT-based.
   - Override the `toString()` method to help with debugging
     - Should be quick if the `toString()` in your LBT is in place
     - Using the breadth-first traversal algorithm can be handy with PQs.

- The first method that you need to get running is `insert()` (which needs to have `upheap` and `size` in place): use simple sample data for your driver as you are building/testing e.g., use K-V pairs: 8-8, 6-6, 7-7, 5-5, 3-3, 0-0, 9-9.
    - This way you will insert entries that may or may not need upheaping, and values outputted are easier to understand and map
    - Jot down what the heap should look like and compare when debugging
- You will need to find a way to insert the next node to satisfy the complete binary tree property: think of how a *binary* tree works:
    - How can we insert a new entry in the next position? i.e. how do we find the parent to add this new position to, and whether we add to left or right?
    - **Think of how you may use a stack for this.**
- Once your `LinkedHeapPriorityQueue` is in place and change your HPQ instance in Part B to a LHPQ instance and test that your driver still works correctly.