Project 2(Due date: 07/04/2016)

Problem 1. (Autocomplete Term) Implement an immutable comparable data type Term in Term.java that represents an autocomplete term: a string query and an associated real-valued weight. You must implement the following API, which supports comparing terms by three different orders: lexicographic order by query string (the natural order); in descending order by weight (an alternate order); and lexicographic order by query string but using only the first r characters (a family of alternate orderings). The last order may seem a bit odd, but you will use it in Problem 3 to second all terms that start with a given prefix (of length r).
method description

method	description
`Term(String query)`	initializes a term with the given query string and zero weight
`Term(String query, long weight)`	initializes a term with the given query string and weight
`static Comparator<Term> byReverseWeightOrder()`	compares the terms in descending order by weight
`static Comparator<Term> byPrefixOrder(int r)`	compares the terms in lexicographic order but using only the first r characters of each query
`int compareTo(Term that)`	compares the terms in lexicographic order by query
`String toString()`	returns a string representation of the term

Corner cases. The constructor should throw a java.lang.NullPointerException if query is null and a java.lang.IllegalArgumentException if weight is negative. The byPrefixOrder() method should throw a java.lang.IllegalArgumentException if r is negative.

Output:

```
$ java Term cities.txt 5
Top 5 by lexicographic order:
2200      's Gravenmoer, Netherlands
19190     's-Gravenzande, Netherlands
134520    's-Hertogenbosch, Netherlands
3628      't Hofke, Netherlands
246056    A Corua, Spain
Top 5 by reverse-weight order:
14608512          Shanghai, China
13076300          Buenos Aires, Argentina
12691836          Mumbai, India
12294193          Mexico City, Distrito Federal, Mexico
11624219          Karachi, Pakistan
```

Problem 2. Complete BinaryTree.java with methods inorder, preorder and postorder traversal. And add rotate left and rotate right methods.

Output:

```
 Inorder
10 20 30 40 50 60 70
 Preorder
40 20 10 30 60 50 70
 Postorder
10 30 20 50 70 60 40
 Preorder after rotateLeft
60 40 20 10 30 50 70
 Preorder after rotateRight
40 20 10 30 60 50 70
```