Project 4

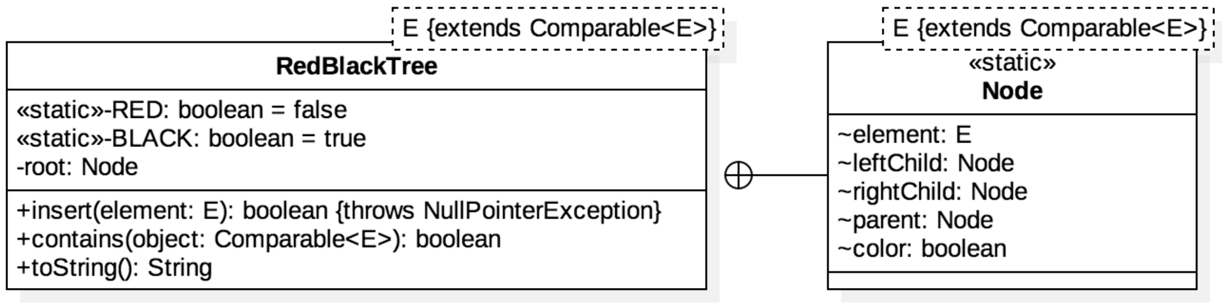
Simplified Red-Black Trees

The task of this project is to implement in Java a red-black tree data structure. However, the tree will be simplified – you only need to support insertion, not deletion.

Specification

The project must implement the following specification exactly, including all identifier names, method signatures, the presence or absence of exceptional behavior, etc. That being said, anything not clearly indicated by the UML diagram(s) or explicitly stated in the description is left up to your discretion. You may also add private helper methods or additional fields as necessary.

Structure


E {extends Comparable<E>}
RedBlackTree
«static»-RED: boolean = false
«static»-BLACK: boolean = true
-root: Node
+insert(element: E): boolean {throws NullPointerException}
+contains(object: Comparable<E>): boolean
+toString(): String
E {extends Comparable<E>}
«static»
Node
~element: E
~leftChild: Node
~rightChild: Node
~parent: Node
~color: boolean

Note that a box with a dashed border over the top right of a class entity denotes a generic type parameter. In this case, the red-black tree class has a generic type named `E` that extends `Comparable<E>` – you may choose whether or not to make `Node` generic as well. The `Comparable<T>` interface is located in the `java.lang` package, so it is not necessary to import it. Finally, for this project you should locate your code in the default package.

Behavior

`insert` should insert the given element into the tree at the correct position, and then rebalance the tree if necessary. The correct position is defined by the properties of a binary search tree, and the rebalancing procedure should enforce the properties of a red-black tree. Regarding input validation, `insert` should immediately throw a `NullPointerException` with a descriptive message if the given element is null. Alternatively, if the given element is a duplicate of an element already in the tree, then `insert` should not insert the given element. The return value should indicate whether the given element was inserted into the tree or not.

Two elements are considered duplicates iff (if and only if) they compare equal to each other using the `compareTo` method. Likewise, the ordering of any two elements for the purposes of insertion and rebalancing is given by the same method.

`contains` should return whether the tree contains any element that compares equal to the given object using the `compareTo` method of the object. This means that you should always do `object.compareTo(element)` but never do `element.compareTo(object)`.

However, if the given object is null, then `contains` should not throw an exception but rather should return false.

`toString` should override the eponymous method of `Object` and return a string representing the pre-order traversal of this tree. The returned string should be the ordered concatenation of invoking the `toString` method of each element in the traversal, where every two adjacent elements should be separated by a single space character (" "). If an element is located in a **red** node, then it should be preceded by a single asterisk character ("*") in the output string. Otherwise, an element located in a **black** node should not be preceded by an asterisk. An example of the output is as follows (assuming that the elements are of type `Integer`):

```
30 2 *5 47 *60
```

It is entirely optional, but it may make your life easier to use a `StringJoiner` and/or to implement `Node#toString()` as well.

The `color` field of the node class should be assigned and evaluated using the `RED` and `BLACK` constants of the enclosing tree class. This means that you should always do `color = BLACK` or `if(color == RED)` but never do `color = true` or `if(!color)`.

Main Class

Along with the RedBlackTree class include another class that has a main function. You can name this class anything just clearly indicate the name in Readme File. This class will take two command line arguments. The first argument will be the input file name and second will be output file name. The input file will be given to the program and the output file will be generated by the program. This main class will create an instance of RedBlackTree and do the operations specified in the input file.

To understand the format of input and output files please see the following examples.

Example 1

```
Integer
Insert:98
Insert:-68
Insert:55
Insert:45
PrintTree
Contains:45
Insert:84
Insert:32
Insert:132
Insert:45
PrintTree
Insert
hih
```

{The first line indicates the object that needs to be inserted into the Tree. It can only accept two objects String or Integer In this example we insert Integer objects. }

The corresponding correct output file should be

```
True
True
True
True
55 -68 *45 98
True
True
True
True
False
55 32 *-68 *45 98 *84 *132
Error in Line: Insert
Error in Line: hih
```

Example 2

```
String
Insert:Ana
Insert:Owen
Insert:Pete
Insert:Leo
PrintTree
Contains:Owen
Insert:Nick
Insert:Maya
Insert:Leo
PrintTree
```

The corresponding correct output file should be

```
True
True
True
True
Owen Ana *Leo Pete
True
True
True
False
Owen *Leo Ana Nick *Maya Pete
```

For all other objects (other than String and Integer), just declare can't work with the object in your output file and quit.

Example 2

```
Students
```

The corresponding correct output file should be

```
Only works for objects Integers and Strings
```

Remember your RedBlackTree class itself should be generic and should allow any object that is Comparable. We will be testing your RedBlackTree class separately using unit testing modules.

Submission

Submit the following items on eLearning:

1. **README.txt**
   This should identify who you are (name, NetID, etc.), which project you are submitting, what files comprise your project, how you developed and compiled your project (e.g. what IDE or text editor, which version of Java, what compiler options, etc.), and any other information you believe the grader should know or you want the grader to know.
2. **RedBlackTree.java**
   This should be the only source code file that you submit. It should not include a `main` method, since it is intended to be used like library code. (that is also how it will be tested) The source code should include an appropriate Javadoc for every public method and class.
3. **Main Class**
   Give an appropriate name and indicate it in the readme file. Should take two command line argument and process the input file and write the output file accordingly. Have the file inside default package.

Both items should be submitted as a single zipped file named with your lowercase NetID. The file structure should resemble the following example:

```
*-- abc123789.zip
    |-- README.txt
    |-- RedBlackTree.java
    |-- Main.java
```

Evaluation

This project will be evaluated primarily according to the correctness of your implementation of the red-black tree insertion and rebalancing algorithm, and secondarily according to the stability of your implementation and its faithfulness to the specification. It is of utmost importance that you exactly adhere to the structural specifications since your code must be compatible with a standardized test suite. Finally, bonus points may be awarded for exceptional code quality, which comprises engineering and design (e.g. modularity), documentation and comments (e.g. Javadoc), and style and layout (e.g. visually grouping lines into logical "paragraphs").

The rubric is as follows:

Category	Weight
`insert`	70%
`contains`	20%
`toString`	10%

Commentary

The specification for this project emulates several conventions used in the actual Java standard library (with appropriate simplifications). For example, the `TreeSet` class is a red-black tree that uses `RED` and `BLACK` boolean constants exactly as specified here in order to more efficiently simulate a binary enum without sacrificing the expressive power of an enum.