```java
package br.anhembi.aps.structure;

import br.anhembi.aps.exceptions.InsertNullException;

/*
 * Felipe Castro Marques - 21259039
 * João Pedro Caires dos Santos Dante - 21321117
 * Nícolas Ribeiro Marques - 21279441
 * Gabriel Santana Mascena - 21265810
 */

public class MBBSTree<Type extends MonsterComparable<Type>> implements
MBBSTreeADT<Type> {

    private Node<Type> powerTree;
    private Node<Type> charismaTree;

    public MBBSTree() {
        this.powerTree = null;
        this.charismaTree = null;
    }

    @Override
    public void insertItem(Type data) {
        if (data == null) throw new InsertNullException();
        this.powerTree = this.insertPowerItem(this.powerTree, data);
        this.charismaTree = this.insertCharismaItem(this.charismaTree, data);
    }

    @Override
    public void traversingPowerPreOrder() {
        if (this.powerTree != null) {
            System.out.println(this.powerTree.data);
            this.traversingPreOrder(this.powerTree.left);
            this.traversingPreOrder(this.powerTree.right);
        }
    }

    @Override
    public void traversingPowerInOrder() {
        if (this.powerTree != null) {
            this.traversingInOrder(this.powerTree.left);
            System.out.println(this.powerTree.data);
            this.traversingInOrder(this.powerTree.right);
        }
    }

    @Override
```

```
public void traversingPowerPostOrder() {
   if (this.powerTree != null) {
      traversingPostOrder(this.powerTree.left);
      traversingPostOrder(this.powerTree.right);
      System.out.println(this.powerTree.data);
   }
}

// ----------

@Override
public void traversingCharismaPreOrder() {
   if (this.charismaTree != null) {
      System.out.println(this.charismaTree.data);
      this.traversingPreOrder(this.charismaTree.left);
      this.traversingPreOrder(this.charismaTree.right);
   }
}

@Override
public void traversingCharismaInOrder() {
   if (this.charismaTree != null) {
      this.traversingInOrder(this.charismaTree.left);
      System.out.println(this.charismaTree.data);
      this.traversingInOrder(this.charismaTree.right);
   }
}

@Override
public void traversingCharismaPostOrder() {
   if (this.charismaTree != null) {
      traversingPostOrder(this.charismaTree.left);
      traversingPostOrder(this.charismaTree.right);
      System.out.println(this.charismaTree.data);
   }
}

private Node<Type> insertPowerItem(Node<Type> root, Type data) {
   if (root == null) {
      return new Node(data);
   } else {
      if (data.comparePowerTo(root.data) < 0) {
         root.left = insertPowerItem(root.left, data);
      } else if (data.comparePowerTo(root.data) > 0) {
         root.right = insertPowerItem(root.right, data);
      } else {
         return root;
      }
```

```
    }
    return root;
}

private Node<Type> insertCharismaItem(Node<Type> root, Type data) {
    if (root == null) {
        return new Node(data);
    } else {
        if (data.compareCharismaTo(root.data) < 0) {
            root.left = insertCharismaItem(root.left, data);
        } else if (data.compareCharismaTo(root.data) > 0) {
            root.right = insertCharismaItem(root.right, data);
        } else {
            return root;
        }
    }
    return root;
}

private void traversingPreOrder(Node<Type> root) {
    if (root != null) {
        System.out.println(root.data);
        this.traversingPreOrder(root.left);
        this.traversingPreOrder(root.right);
    }
}

private void traversingInOrder(Node<Type> root) {
    if (root != null) {
        this.traversingInOrder(root.left);
        System.out.println(root.data);
        this.traversingInOrder(root.right);
    }
}

private void traversingPostOrder(Node<Type> root) {
    if (root != null) {
        traversingPostOrder(root.left);
        traversingPostOrder(root.right);
        System.out.println(root.data);
    }
}

@Override
public boolean isEmpty() {
    return this.powerTree == null || this.charismaTree == null;
}
```

```
  @Override
  public boolean contains(Type data) {
    if ((this.charismaTree == null && this.powerTree == null) || data == null) return false;
    return this.containsPowerItem(this.powerTree, data) ||
containsCharismaItem(this.charismaTree, data);
  }

  private boolean containsPowerItem(Node<Type> child, Type data) {
    if (child == null) return false;
    int result = data.comparePowerTo(child.data);
    if (result == 0) return true;
    if (result < 0 && child.left != null) return this.containsPowerItem(child.left, data);
    if (result > 0 && child.right != null) return this.containsPowerItem(child.right, data);
    return false;
  }

  private boolean containsCharismaItem(Node<Type> child, Type data) {
    if (child == null) return false;
    int result = data.compareCharismaTo(child.data);
    if (result == 0) return true;
    if (result < 0 && child.left != null) return this.containsCharismaItem(child.left, data);
    if (result > 0 && child.right != null) return this.containsCharismaItem(child.right, data);
    return false;
  }

  @Override
  public void delete(Type data) {
    if (data != null && this.powerTree != null) this.powerTree = deletePowerItem(this.powerTree,
data);
    if (data != null && this.charismaTree != null) this.charismaTree =
deleteCharismaItem(this.charismaTree, data);
  }

  private Node<Type> deletePowerItem(Node<Type> root, Type data) {
    if (root != null) {
      int result = data.comparePowerTo(root.data);
      if (result < 0) root.left = deletePowerItem(root.left, data);
      else if (result > 0) root.right = deletePowerItem(root.right, data);
      else if (root.left == null) root = root.right;
      else if (root.right == null) root = root.left;
      else root.right = this.getMinOfRightSubTree(root, root.right);
    }
    return root;
  }

  private Node<Type> deleteCharismaItem(Node<Type> root, Type data) {
    if (root != null) {
      int result = data.compareCharismaTo(root.data);
```

```
            if (result < 0) root.left = deleteCharismaItem(root.left, data);
            else if (result > 0) root.right = deleteCharismaItem(root.right, data);
            else if (root.left == null) root = root.right;
            else if (root.right == null) root = root.left;
            else root.right = this.getMinOfRightSubTree(root, root.right);
        }
        return root;
    }

    private Node<Type> getMinOfRightSubTree(Node<Type> root, Node<Type> subTreeRoot) {
        if (subTreeRoot.left == null) {
            root.data = subTreeRoot.data;
            return subTreeRoot.right;
        } else {
            subTreeRoot.left = this.getMinOfRightSubTree(root, subTreeRoot.left);
        }
        return subTreeRoot;
    }
}
```