```java
package scmAlgorithm;

import epos.model.tree.Tree;
import epos.model.tree.treetools.TreeUtilsBasic;
import gnu.trove.map.hash.TObjectDoubleHashMap;
import scmAlgorithm.treeScorer.ConsensusResolutionScorer;
import scmAlgorithm.treeSelector.TreePair;
import scmAlgorithm.treeSelector.TreeSelector;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

/**
 * Created by fleisch on 05.02.15.
 */
public abstract class AbstractSCMAlgorithm implements SupertreeAlgorithm{
    protected List<Tree> superTrees;
    public final TreeSelector selector;

    public AbstractSCMAlgorithm(TreeSelector selector) {
        this.selector = selector;
    }

    protected abstract List<TreePair> calculateSuperTrees();

    @Override
    public List<Tree> getSupertrees() {
        if (superTrees == null || superTrees.isEmpty()) {
            List<TreePair> finalPairs = calculateSuperTrees();
            superTrees = new ArrayList<>(finalPairs.size());
            TreeResolutionComparator comp =  new TreeResolutionComparator();

            for (TreePair pair : finalPairs) {
                Tree st =  pair.getConsensus();
                TreeUtilsBasic.cleanTree(st);
                comp.put(st, TreeUtilsBasic.calculateTreeResolution(pair.
                    getNumOfConsensusTaxa(), st.vertexCount()));
                superTrees.add(st);
            }
            Collections.sort(superTrees, comp);
        }
        return superTrees;
    }

    @Override
    public Tree getSupertree() {
        return getSupertrees().get(0);
    }

    //Descending comparator
    protected class TreeResolutionComparator implements Comparator<Tree> {
        //caches scores of already known trees
        private TObjectDoubleHashMap<Tree> scores =  new
            TObjectDoubleHashMap<>();
        @Override
        public int compare(Tree o1, Tree o2) {
            double s1 = scores.get(o1);
            if (s1 == scores.getNoEntryValue()){
```

```
59                  s1 =  caclulateTreeResolution(o1);
60                  scores.put(o1,s1);
61              }
62
63              double s2 =  scores.get(o2);
64              if (s2 == scores.getNoEntryValue()){
65                  s2 =  caclulateTreeResolution(o2);
66                  scores.put(o2,s2);
67              }
68
69              return Double.compare(s2,s1);//ATTENTION: wrong order to create
                    a descending comparator
70          }
71          private double caclulateTreeResolution(Tree tree) {
72              return TreeUtilsBasic.calculateTreeResolution(tree.getNumTaxa(),
                     tree.vertexCount());
73          }
74          public double put(Tree tree, double resolution){
75              return scores.put(tree,resolution);
76          }
77      }
78  }
```