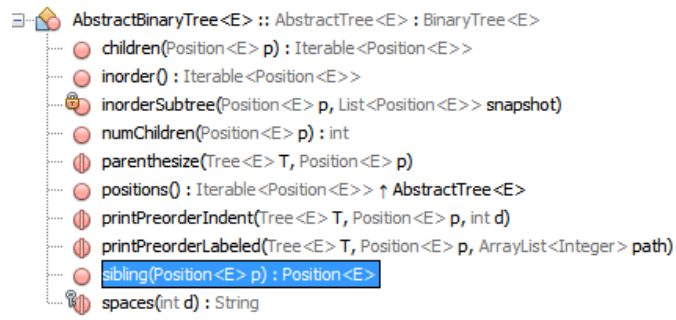
C:\Users\Rich\Documents\NetBeansProjects\Lab08\src\AbstractBinaryTree.java

```
import java.util.ArrayList;
import java.util.List;

/**
 * An abstract base class providing some functionality
 * of the Binary Tree Interface.
 * @author Goodrich, Tamassia, Goldwasser
 */
public abstract class AbstractBinaryTree<E> extends AbstractTree<E> implements BinaryTree<E> {

  /** return the position of p's sibling ( or null if no sibling exists).*/
  public Position<E> sibling(Position<E> p)
  {
    Position<E> parent = parent(p);
    if(parent == null) return null;
    if (p == left(parent))
      return right(parent);
    else
      return left(parent);
  }

  public int numChildren(Position<E> p)
  {
    int count = 0;
    if(left(p)!= null)
      count++;
    if(right(p) != null)
      count++;
    return count;
  }
  /** Returns an iterable collection of the Positions representing p's children.*/
  public Iterable<Position<E>> children(Position<E> p)
  {
    List<Position<E>> snapshot = new ArrayList<> (2);
    if(left(p) != null)
      snapshot.add(left(p));
    if(right(p)!= null)
      snapshot.add(right(p));
    return snapshot;
  }


  // code fragement 8.22
  /** Adds positions of the subtree rooted at Position p to the given snapshot. */
  private void inorderSubtree(Position<E> p, List<Position<E>> snapshot)
  {
    if(left(p)!= null)
      inorderSubtree(left(p), snapshot);
    snapshot.add(p);
    if(right(p)!= null)
      inorderSubtree(right(p), snapshot);
  }
  /** Returns an iterable collection of positions of the tree, reported in inorder. */
  public Iterable<Position<E>> inorder()
  {
```

get the parent of argument p

// that's mean we're @ the root

in subclass linkedBinaryTree



```
    List<Position<E>> snapshot = new ArrayList<>();
    if(!isEmpty())
        inorderSubtree(root(), snapshot);
    return snapshot;
  }
  /** Overrides positions to make inorder the default order for binary trees. */
  public Iterable<Position<E>> positions()
  {
    return inorder();
  }

  //code fragment 8.23
  public static<E> void printPreorderIndent(Tree<E> T, Position<E> p,int d)
  {
    System.out.println(spaces(2*d) + p.getElement());      //Indent based on d
    for(Position<E> c : T.children(p))
      printPreorderIndent(T,c, d+1);       //child depth is d+1
  }
  public static<E> void printPreorderLabeled(Tree<E>T, Position<E>p, ArrayList<Integer> path)
  {
    int d = path.size();
    System.out.print(spaces(2*d));
    for(int j =0; j<d; j++)
        System.out.print(path.get(j) +(j==d-1? " " : "."));
    System.out.println(p.getElement());
    path.add(1);
    for(Position<E> c: T.children(p))
    {
        printPreorderLabeled(T,c,path);
        path.set(d,1+path.get(d));         //increment last entry of path
    }
    path.remove(d);                        //restore path to its incoming state

  }

  public static <E> void parenthesize(Tree<E> T, Position<E> p)
  {
    System.out.print(p.getElement());
    if(T.isInternal(p))
    {
        boolean firstTime = true;
        for(Position<E> c: T.children(p))
        {
            System.out.print((firstTime ? " (" : ", "));
            firstTime = false;
            parenthesize(T,c);
        }
        System.out.print(")");
    }
  }

  // utility method
  protected static String spaces(int d)
  {
    String spaceWidth = " ";
    for(int i = 0; i < d; i++ )
        spaceWidth += " ";
    return spaceWidth;
  }
```

```
}
```