```
from tree import Tree

class BinaryTree(Tree):
  """Abstract base class representing a binary tree structure."""

  # --------------------- additional abstract methods ---------------------

  def left(self, p):
    """Return a Position representing p's left child.

    Return None if p does not have a left child.
    """
    raise NotImplementedError('must be implemented by subclass')

  def right(self, p):
    """Return a Position representing p's right child.

    Return None if p does not have a right child.
    """
    raise NotImplementedError('must be implemented by subclass')

  # ---------- concrete methods implemented in this class ----------

  def sibling(self, p):
    """Return a Position representing p's sibling (or None if no sibling)."""
    parent = self.parent(p)
    if parent is None:                    # p must be the root
      return None                         # root has no sibling
    else:
      if p == self.left(parent):
        return self.right(parent)         # possibly None
      else:
        return self.left(parent)          # possibly None

  def children(self, p):
    """Generate an iteration of Positions representing p's children."""
    if self.left(p) is not None:
      yield self.left(p)
    if self.right(p) is not None:
      yield self.right(p)

  def inorder(self):
    """Generate an inorder iteration of positions in the tree."""
    if not self.is_empty():
      for p in self._subtree_inorder(self.root()):
        yield p

  def _subtree_inorder(self, p):
    """Generate an inorder iteration of positions in subtree rooted at p."""
    if self.left(p) is not None:          # if left child exists, traverse its
subtree
      for other in self._subtree_inorder(self.left(p)):
        yield other
    yield p                               # visit p between its subtrees
```

```
    if self.right(p) is not None:         # if right child exists, traverse its
subtree
      for other in self._subtree_inorder(self.right(p)):
        yield other

  # override inherited version to make inorder the default

  def positions(self):
    """Generate an iteration of the tree's positions."""
    return self.inorder()                 # make inorder the default
```