1. Overall Object-oriented design:
   - QuadTreeCommands: where to process all the commands from inputs

   - QuadTree<T>: the QuadTree class that has 3 types of nodes: EmptyNode, LeafNode and InternalNode
     - Inner interface and inner classes that implement it (Node-centric approach, Polymorphism and Recursion):

       Interface QuadNode<E>: insert, find, delete, tree, rfind
       - EmptyNode<E>: use Flyweight design patter (create only one instance of it and whenever we need to add an empty node somewhere, simply point to the existing empty node instance from the QuadTree)
       - LeafNode<E>
       - InternalNode<E>

   - Point2D: stores locations of the cities

   - Use Strategic design pattern:
     Interface Extractor2D<T>: getPoint2D(T elem)
     - CoordinatesExtractor2D<T> implements Extractor2D<T>

     (Extractor is used to get locations of the cities while keeping everything generics)

2.
   - find: Every time moving down the tree, I checked for within which quadrant the needed city is located and went down that way.

   - rfind: I used something similar to the full traversal except before moving deeper, I checked to see whether the current quadrant overlaps with the given bounding box, if not, I won't go.