Collection Selector

Problem Overview

This assignment focuses on implementing the methods of a class much like `java.util.Collections`. The `Selector.java` file defines a class with static methods that implement polymorphic algorithms that operate on and/or return objects that implement the `Collection` interface. Each method of `Selector` is clearly specified, is independent of the other methods in the class, and is designed to provide relatively simple functionality. So, this is a great context in which to practice systematic, disciplined development and test-based verification.

The `Selector` class

You must correctly implement all the method bodies of the provided `Selector` class. Your implementation must adhere **exactly** to the API of the `Selector` class, as described in the provided source code comments and as described below.

```
public class Selector {
    public static <T> T min(Collection<T> c, Comparator<T> comp)
    public static <T> T max(Collection<T> c, Comparator<T> comp)
    public static <T> T kmin(Collection<T> c, int k, Comparator<T> comp)
    public static <T> T kmax(Collection<T> c, int k, Comparator<T> comp)
    public static <T> Collection<T> range(Collection<T> c, T low, T high,
                                          Comparator<T> comp)
    public static <T> T ceiling(Collection<T> c, T key, Comparator<T> comp)
    public static <T> T floor(Collection<T> c, T key, Comparator<T> comp)
}
```

Refer to the `Selector.java` source code file for the complete specification of each method's required behavior. For an overview of each method, brief descriptions and example calls are provided below. While the generic type variable makes these methods independent of any specific data type, in each example the type variable `T` is bound to `Integer`. The

definitions of “less than”, “greater than,” and “equal to” for values of type `T` are defined by the `Comparator` parameter `comp`. This parameter in each of the examples is the `ascendingInteger` object defined below.

```
/**
 * Defines a total order on integers as ascending natural order.
 */
static Comparator<Integer> ascendingInteger =
    new Comparator<Integer>() {
        public int compare(Integer i1, Integer i2) {
            return i1.compareTo(i2);
        }
    };
```

The `min` method

This method selects the minimum value from a given collection. Examples:

coll	min(coll, comp)
[2, 8, 7, 3, 4]	2
[5, 9, 1, 7, 3]	1
[8, 7, 6, 5, 4]	4
[8, 2, 8, 7, 3, 3, 4]	2

The `max` method

This method selects the maximum value from a given collection. Examples:

coll	max(coll, comp)
[2, 8, 7, 3, 4]	8
[5, 9, 1, 7, 3]	9
[8, 7, 6, 5, 4]	8

[8, 2, 8, 7, 3, 3, 4]	8

The `kmin` method

This method selects the k-th minimum (smallest) value from a given collection. A value is the k-th minimum if and only if there are exactly k - 1 distinct values strictly less than it in the collection. Note that `kmin(coll, 1, comp) == min(coll, comp)` and `kmin(coll, coll.size(), comp) == max(coll, comp)`. Examples:

coll	k	kmin(coll, k, comp)
[2, 8, 7, 3, 4]	1	2
[5, 9, 1, 7, 3]	3	5
[8, 7, 6, 5, 4]	5	8
[8, 2, 8, 7, 3, 3, 4]	3	4

The `kmax` method

This method selects the k-th maximum (largest) value from a given collection. A value is the k-th maximum if and only if there are exactly k - 1 distinct values strictly greater than it in the collection. Note that `kmax(coll, 1, comp) == max(coll, comp)` and `kmax(coll, coll.size()) == min(coll, comp)`. Examples:

coll	k	kmax(coll, k, comp)
[2, 8, 7, 3, 4]	1	8
[5, 9, 1, 7, 3]	3	5
[8, 7, 6, 5, 4]	5	4
[8, 2, 8, 7, 3, 3, 4]	3	4

The `range` method

This method selects all values from a given collection that are greater than or equal to `low`

and less than or equal to `high`.

coll	low	high	range(coll, low, high, comp)
[2, 8, 7, 3, 4]	1	5	[2, 3, 4]
[5, 9, 1, 7, 3]	3	5	[5, 3]
[8, 7, 6, 5, 4]	4	8	[8, 7, 6, 5, 4]
[8, 2, 8, 7, 3, 3, 4]	3	7	[7, 3, 3, 4]

The `floor` method

This method selects from a given collection the largest value that is less than or equal to `key`. Examples:

coll	key	floor(coll, key, comp)
[2, 8, 7, 3, 4]	6	4
[5, 9, 1, 7, 3]	1	1
[8, 7, 6, 5, 4]	9	8
[8, 2, 8, 7, 3, 3, 4]	5	4

The `ceiling` method

This method selects from a given collection the smallest value that is greater than or equal to `key`. Examples:

coll	key	ceiling(coll, key, comp)
[2, 8, 7, 3, 4]	1	2
[5, 9, 1, 7, 3]	7	7
[8, 7, 6, 5, 4]	0	4
[8, 2, 8, 7, 3, 3, 4]	5	7

Notes and Other Requirements

- The `Selector.java` source code file is provided in the `startercode` folder in Vocareum. You can download `Selector.java` from Vocareum and work on your local machine.
- The comments provided in `Selector.java` describe the required behavior of each method.
- The constructor of the `Selector` class has been written for you and it must not be changed in any way.
- You may add any number of private methods that you like, but you may not add any public method or constructor, nor may you change the signature of any public method or constructor.
- You must not add any fields, either public or private, to the `Selector` class.
- You must not add, remove, or change in any way the import statements that are already provided. You may not use fully-qualified names to circumvent this restriction, except in the instance noted below.
- You may not use any of the `toArrray()` methods in the `Collection` interface and then solve the problem in terms of arrays. More generally, you may not convert the `Collection` parameter to any other type (with the exception necessary for sorting in `kmin` and `kmax` mentioned below). The penalty for violating this constraint will be a deduction of up to 50% of the total points available on the assignment.
- You may not use sorting in any method, except for `kmin` and `kmax`. The penalty for violating this constraint will be a deduction of points up to 50% of the total points available on the assignment.
- You do not have to use sorting in `kmin` and `kmax`, but doing so makes the solution more straightforward. If you choose to use sorting in these two methods, you must do so by calling the `java.util.Collections.sort(List, Comparator)` method. You must use the fully-qualified name (no importing `Collections`) and you are allowed at most two calls to this method - at most one in `kmin` and at most one in `kmax`. You are also allowed to use a call to `java.util.Collections.reverse(List)` in `kmax`.