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 Collections. Each method of `Selector` is very 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 for practicing what we've been discussing in lecture – generalized programming 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)
}
```

The sections that follow provide details of each method's behavior as well as specific examples.

The `min` method.

This method selects the minimum value from a given collection, as defined by a given comparator. If either the collection or comparator is null, this method throws an IllegalArgumentException. If the collection is empty, the method throws a NoSuchElementException. The collection must not be changed by this method.

Examples:

T	c	comp	<T>min(c, comp)
Integer	[2, 8, 7, 3, 4]	*ascending order*	2
Integer	[5, 9, 1, 7, 3]	*descending order*	9
Integer	[8, 7, 6, 5, 4]	*ascending order*	4
(String, Integer)	[(A,5), (B,4), (C,3), (D,2), (E, 1)]	*ascending by String field*	(A,5)
(String, Integer)	[(A,5), (B,4), (C,3), (D,2), (E, 1)]	*ascending by Integer field*	(E,1)

The `max` method.

This method selects the maximum value from a given collection, as defined by a given comparator. If either the collection or comparator is null, this method throws an IllegalArgumentException. If the collection is empty, the method throws a NoSuchElementException. The collection must not be changed by this method.

Examples:

T	c	comp	<T>max(c, comp)
Integer	[2, 8, 7, 3, 4]	*ascending order*	8
Integer	[5, 9, 1, 7, 3]	*descending order*	1
Integer	[8, 7, 6, 5, 4]	*ascending order*	8
(String, Integer)	[(A,5), (B,4), (C,3), (D,2), (E, 1)]	*ascending by String field*	(E,1)
(String, Integer)	[(A,5), (B,4), (C,3), (D,2), (E, 1)]	*ascending by Integer field*	(A,5)

The `kmin` method.

This method selects the k^{th} minimum value from a given collection, as defined by a given comparator. A value is the k^{th} minimum if there are exactly $k-1$ values less than it in the collection. If either the collection or comparator is null, this method throws an IllegalArgumentException. If the collection is empty or if there is no k^{th} minimum value, this method throws a NoSuchElementException. Note that there is no k^{th} minimum value if k is less than 1, k is greater than the number of elements in the collection, or if k is greater than the number of distinct values in the collection. The collection must not be changed by this method.

Examples:

T	c	k	comp	<T>kmin(c, k, comp)
Integer	[2, 8, 7, 3, 4]	1	*ascending order*	2
Integer	[5, 9, 1, 7, 3]	2	*descending order*	7
Integer	[8, 7, 6, 5, 4]	3	*ascending order*	6
(String, Integer)	[(A,5), (B,4), (C,3), (D,2), (E, 1)]	4	*ascending by String field*	(D,2)
(String, Integer)	[(A,5), (B,4), (C,3), (D,2), (E, 1)]	2	*ascending by Integer field*	(D,2)

The `kmax` method.

This method selects the k^{th} maximum value from a given collection, as defined by a given comparator. A value is the k^{th} maximum if there are exactly $k-1$ values greater than it in the collection. If either the collection or comparator is null, this method throws an IllegalArgumentException. If the collection is empty or if there is no k^{th} minimum value, this method throws a NoSuchElementException. Note that there is no k^{th} minimum value if k is less than 1, k is greater than the number of elements in the collection, or if k is greater than the number of distinct values in the collection. The collection must not be changed by this method.

Examples:

T	c	k	comp	<T>kmax(c, k, comp)
Integer	[2, 8, 7, 3, 4]	1	*ascending order*	8
Integer	[5, 9, 1, 7, 3]	2	*descending order*	3
Integer	[8, 7, 6, 5, 4]	3	*ascending order*	6
(String, Integer)	[(A,5), (B,4), (C,3), (D,2), (E, 1)]	4	*ascending by String field*	(B,4)
(String, Integer)	[(A,5), (B,4), (C,3), (D,2), (E, 1)]	2	*ascending by Integer field*	(B,4)

The `range` method.

This method returns a collection of all values i from a given collection such that $low \leq i \leq high$, as defined by the given comparator, including duplicate values. (Note that *low* and *high* do not have to be actual values in the given collection.) The returned collection contains only values in the range [low..high], and no others. If there are no qualifying values, including the case where c is empty, this method throws a NoSuchElementException. This method throws an IllegalArgumentException if either the collection or comparator is null. The collection is not changed by this method.

Examples:

T	c	low	high	comp	<T>range(c, low, high, comp)
Integer	[2, 8, 7, 3, 4]	1	5	*ascending order*	[2,3,4]
Integer	[5, 9, 1, 7, 3]	3	5	*ascending order*	[5,3]
Integer	[5, 9, 1, 7, 3]	5	3	*descending order*	[5,3]
Integer	[8, 7, 6, 5, 4]	4	8	*ascending order*	[8, 7, 6, 5, 4]
(String, Integer)	[(A,5), (B,4), (C,3)]	(B,3)	(C,5)	*asc. by String field*	[(B,4),(C,3)]
(String, Integer)	[(A,5), (B,4), (C,3)]	(F,4)	(G,7)	*asc. by Integer field*	[(A,5),(B,4)]

The `ceiling` method.

This method returns the smallest value i in a given collection such that $i \geq key$, as defined by the given comparator. (Note that *key* does not have to be an actual value in the given collection.) If the given collection or comparator is null, this method throws an IllegalArgumentException. If the collection is empty or if there is no qualifying value i, this method throws a NoSuchElementException.

Examples:

T	c	key	comp	<T>ceiling(c, key, comp)
Integer	[2, 8, 7, 3, 4]	1	*ascending order*	2
Integer	[5, 9, 1, 7, 3]	7	*descending order*	7
Integer	[8, 7, 6, 5, 4]	0	*ascending order*	4
(String, Integer)	[(A,5), (B,4), (C,3), (D,2), (E, 1)]	(B,9)	*ascending by String field*	(B,4)
(String, Integer)	[(A,5), (B,4), (C,3), (D,2), (E, 1)]	(F,0)	*ascending by Integer field*	(E,1)

The `floor` method.

This method returns the largest value i in a given collection such that $i \leq key$, as defined by the given comparator. (Note that *key* does not have to be an actual value in the given collection.) If the given collection or comparator is null, this method throws an IllegalArgumentException. If the collection is empty or if there is no qualifying value i, this method throws a NoSuchElementException.

Examples:

T	c	key	comp	<T>floor(c, key, comp)
Integer	[2, 8, 7, 3, 4]	6	*ascending order*	4
Integer	[5, 9, 1, 7, 3]	1	*descending order*	1
Integer	[8, 7, 6, 5, 4]	9	*ascending order*	8
(String, Integer)	[(A,5), (B,4), (C,3), (D,2), (E, 1)]	(F,0)	*ascending by String field*	(E,1)
(String, Integer)	[(A,5), (B,4), (C,3), (D,2), (E, 1)]	(B,9)	*ascending by Integer field*	(A,5)

Notes and other requirements

Here are other specific requirements, notes, and suggestions.

- Read this handout carefully. Read the provided source code carefully. Ask questions on Piazza. Start early and be proactive.
- The constructor has been completed for you and 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 any import statements to those already in the Selector class.
- You may not change or delete any import statement in the Selector class.
- You may not use fully-qualified names to circumvent the restrictions on imports above, except in the one instance noted below.
- None of the methods in the Selector class can modify the Collection parameter in any way. More generally, the methods in the Selector class should have no side-effects.
- You are only allowed to use sorting as part of your solution to `kmin` and `kmax`. You are not required to use sorting, but you are allowed to do so for these two methods only. If you use sorting, 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`.
- The use of the `ArrayList` class is allowed only in `kmin`, `kmax`, and `range`. No other method is allowed to use any implementing class of `Collection`.
- The declaration or use of an array in any method is strictly prohibited.
- Your code must be type safe; that is, your code must compile cleanly with no compiler warnings.