1 Category Partition Method

1.1 Prerequisites Consideration

1.1.1 Type of data structure

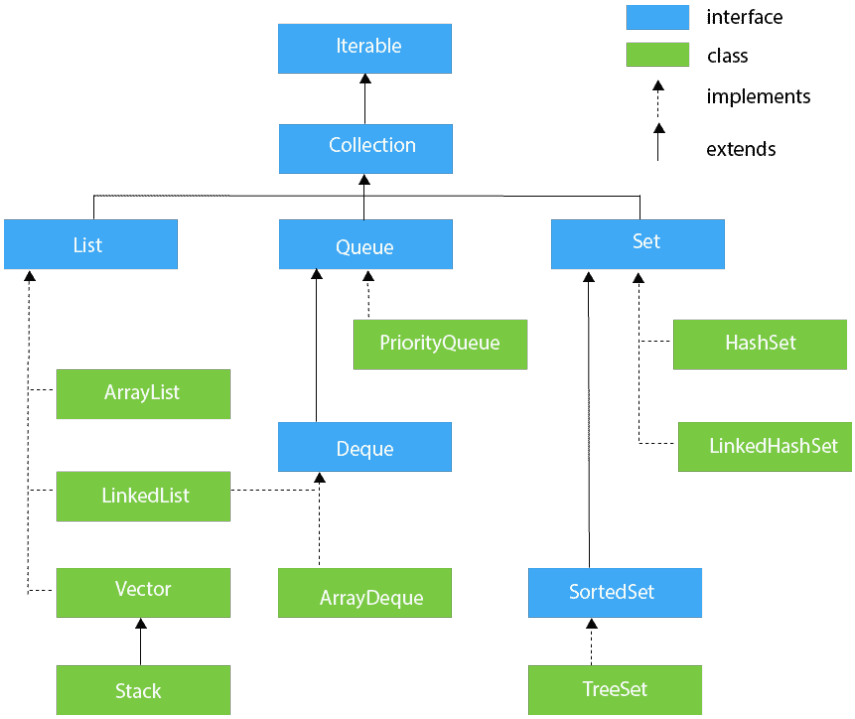

Figure 1: Java Collections Framework hierarchy [9].

As shown in **Figure 1**, `Collection` is an interface that is implemented by three different sub-interfaces, and each sub-interface is extended by different data structure classes. Therefore, it is worth to investigate the relationship between the type of data structure used and the output of the testing implementation.

1. `List`:

   (a) `ArrayList`, `LinkedList`, and `Vector` implement the `List` interface. The main difference between them is their implementation which causes different performance for different operations [1, 12, 13].

   (b) Both `Collections.sort` and `Collections.rotate` use interface over concrete types for the `list` parameter. This implies that both methods will only call methods that are defined by the `List` interface (so no `ArrayList` etc. specific methods).

   Hence, assume that `ArrayList`, `LinkedList`, and `Vector` have correct and complete implementation, the testing results of `Collections.sort` and `Collections.rotate` should not be affected by the type of data structure used.

2. `Collection`:

   (a) `List`, `Queue`, and `Set` are fundamentally different from each other in terms of storing and manipulating the data. One crucial difference is that `Set` does

not allow duplicate elements [14], which means that there will be certain categories that are not applicable when the input `coll` is a `Set`.

(b) `Collections.min` uses interface over concrete types for the `coll` parameter, which implies that the `min` method will only use methods that are defined by the `Collection` interface.

Assume that `List`, `Queue`, and `Set` have been correctly and completely implemented, implemented, the testing results of `min` should not be affected by the type of data structure used as well.

1.1.2 Type of elements

Another aspect to consider is the type of the elements stored in the `List` or `Collection`. Since, all elements must be an instance of a class that implements the `Comparable` interface [6], the correctness of `Collections.sort` and `Collections.min` is then dependent on the implementation of the `compareTo` method.

In this testing assessment, the primary objective is to test the `sort`, `rotate`, and `min` methods of the `Collections` interface. The premise for this assessment is that all classes that implemented the `Comparable` interface have defined their respective `compareTo` method correctly, otherwise it would have contradict with the expected output results set for the test cases. Hence, it is assumed that the type of elements used in the testing (e.g. `Integer`, `Double`, `Date` etc.) would not affect the test results, and it would not be taken into consideration as one of the category partition for test inputs.

1.2 Parameter: List

N.B. All elements in the list must be an instance of a class that implements the `Comparable` interface and must be mutually comparable.

Category L1 Empty list

Input: `list = []`

Reason: One of the edge cases of the `List` interface. Given this as the input, both `Collections.sort` and `Collections.rotate` should return an empty list as the output.

Category L2 Single element in the list

Input: `list =` $[x]$

Reason: One of the edge cases of the `List` interface. The output returned by both `Collections.sort` and `Collections.rotate` should be the same as the input.

Category L3 More than one element in the list

Input: `list =` $[x_0, x_1, \ldots, x_n]$ for $n > 1$, where n is the total number of elements in the list

Reason: The is the minimum viable case for testing the actual functionality of both `Collections.sort` and `Collections.rotate`. This is the base form that is used to define other variants of the input (**Category L4 - L7**).

Category L4 Repeated elements in the list

Input: `list =` $[x_0, x_1, \ldots, x_n]$ for $n > 1$. In addition, there exists some a, b for all $0 \leq a, b \leq n$, the value of x_a is equal to x_b.

Reason:

1. This is to test the stability of the `Collections.sort` sorting algorithm. The sorting algorithm should sort the repeated elements in the same order that they appear in the input.
2. This is to test that `Collections.rotate` only modifies the position of the elements. The value of each element should not affect the final output.

Category L5 List contains elements of different class

Input: `list =` $[x_0, x_1, \ldots, x_n]$ for $n > 1$, there exists some i, j such that for all $0 \leq i, j \leq n,\ i \neq j$, the type of $x_i \neq$ type of x_j.

Reason: This is a special category for `Collections.rotate`. Since the purpose of `Collections.rotate` is to modify the index of each element in the list, the actual value of each element should have no significant effect for the method. Therefore, the `rotate` method should work correspondingly when the list contains elements of different types.

Category L6 List is sorted in ascending natural order of its elements

Input: `list =` $[x_0, x_1, \ldots, x_n]$ for $n > 1$, where the list is arranged according to the natural ordering [5] of its elements such that $x_k \leq x_{k+1}$ for all $0 \leq k \leq n$. The list can contain repeated elements.

Reason: This is to test the sorting stability of `Collections.sort`. Since the input list is already sorted, then the elements in the output list should have the same order as the input list.

Category L7 List is sorted in descending natural order of its elements

Input: `list =` $[x_0, x_1, \ldots, x_n]$ for $n > 1$, where the list is arranged according to the **reverse** natural ordering [5] of its elements such that $x_k \geq x_{k+1}$ for all $0 \leq k < n$.

Reason: This is the inverse case of **Category L6**, where the input list is sorted in reverse order. The aim is to test the sorting stability of `Collections.sort` and also ensure that it can handle the cases where certain parts of the list (i.e. sub-list) are inversely sorted.

Category L8 List size is greater than or equal to `ROTATE_THRESHOLD`

Input: `list =` $[x_0, x_1, \ldots, x_n]$ for $n \geq$ `ROTATE_THRESHOLD`, in which the value of `ROTATE_THRESHOLD` is defined as 100 in `Collections.java` [11].

Reason: According to the source code of `Collections.java` [10], Java uses two different algorithm to rotate lists that are $<$ or $\geq$ than `ROTATE_THRESHOLD` respectively. The aim is to validate that both algorithms are able to correctly rotate the input list.

1.3 Parameter: distance

Category D1 Negative number

Input: `distance < 0`

Reason: To ensure that the `Collections.rotate` method will work with negative distance input by covering the negative domain of `int` data type.

Category D2 Zero

Input: `distance == 0`

Reason: 0 is the default value of `int` data type in Java, and also the starting index value of `List`. This is to ensure that the `Collections.rotate` method will work when the input distance is zero.

Category D3 Positive number

Input: `distance > 0`

Reason: To ensure that the `Collections.rotate` method will work with positive distance input by covering the positive domain of `int` data type.

Category D4 Larger than list size

Input: `distance > list.size()`

Reason: A derivative of **Category D3**. Since the `rotate` method uses modulo operation on the input distance [3], this is to test whether the behaviour of inputting a distance larger than `list.size()` is the same as inputting the value of `distance % list.size()` [3].

Category D5 Equal to list size

Input: `distance == list.size()`

Reason: This is to validate that if the distance is equal to the `list.size()`, then `Collections.rotate` will return the the same output as the input.

Category D6 Smaller than list size

Input: `distance < list.size()`

Reason: A derivative of **Category D1 - D3** and the inverse case of **Category D4**. This is to validate the assumption that there exists some d for all $d >$ `list.size()` and n for all $n <$ `list.size()` such that `rotate(list, d) == rotate(list, n)`.

Category D7 Equal to minimum boundary value of `int`

Input: `distance == Integer.MIN_VALUE`

Reason: The minimum value an `int` can have in Java is -2^{31} [8]. The aim is to validate the assumption that if `Collections.rotate` works for the minimum value of `int`, then it should work correctly for any value **larger than the minimum value**.

Category D8 Equal to maximum boundary value of `int`

Input: `distance == Integer.MAX_VALUE`

Reason: The maximum value an `int` can have in Java is $2^{31} - 1$ [7]. The aim is to validate the assumption that if `Collections.rotate` works for the maximum value of `int`, then it should work correctly for any value **smaller than the maximum value**.

1.4 Parameter: Collection

N.B. All elements in the collection must be an instance of a class that implements the `Comparable` interface and must be mutually comparable.

Category C1 Single element in collection

Input: `coll` $= \{x\}$

Reason: Edge case of the `Collections.min` method. The method should only return the single element as the minimum element of the given collection.

Category C2 More than one element in collection

Input: `coll` $= \{x_0, x_1, \ldots, x_n\}$ for $n > 1$

Reason: To test whether the `Collections.min` method is able to find and return the minimum element of the given collection.

Category C3 Repeated elements in collection

Input: `coll` $= \{x_0, x_1, \ldots, x_n\}$ for $n > 1$. In addition, there exists some a, b for all $0 \leq a, b \leq n,\ a \neq b$, the value of x_a is equal to x_b.

Reason: To test whether the `Collections.min` method is able to handle repeated elements and correctly return the minimum element of the given collection.

Category C4 Repeated minimum elements in collection

Input: `coll` $= \{x_0, x_1, \ldots, x_n\}$ for $n > 1$. In addition, there exists some a, b for all $0 \leq a, b \leq n,\ a \neq b$, the value of x_a is equal to x_b and both x_a and x_b are the minimum elements in the collection.

Reason: A derivative of **Category C3** to test the stability of `Collections.min` method. It is expected that the method will treat the repeated minimum elements as the same element and return correctly.

Category C5 Collection contains the minimum possible value of the class

Input: `coll` $= \{x_0, x_1, \ldots, x_n\}$ for $n > 1$, and there exists a k where $0 \leq k \leq n$ such that x_k is the minimum possible value of the element's class.

Reason: To cover the boundary case of `Collections.min`, and to test whether the method always return the minimum possible value of the class when the given collection contains it.

Category C6 Collection is in ascending natural order of its elements

Input: `coll` $= \{x_0, x_1, \ldots, x_n\}$ for $n > 1$, where the collection is arranged according to the natural ordering [5] of its elements such that $x_k \leq x_{k+1}$ for all $0 \leq k \leq n$. The collection can contain repeated elements.

Reason: Since the given collection is already sorted, then the minimum element returned should have the same value as first element of the collection.

Category C7 Collection is in descending natural order of its elements

Input: `coll` $= \{x_0, x_1, \ldots, x_n\}$ for $n > 1$, where the collection is arranged according to the natural ordering [5] of its elements such that $x_k \geq x_{k+1}$ for all $0 \leq k < n$. The collection can contain repeated elements.

Reason: The inverse case of **Category C6**. Since, the given collection is sorted in reverse order, then the minimum element returned should have the same value as the last element of the collection.

2 Test Cases

2.1 `Collections.sort(List<T> list)`

1. **Test Case 1**

   Categories: L2

   Input: `list = [1]`

2. **Test Case 2**

   Categories: L3 $\wedge$ L4

   Input: `list = [6, 8, 2, 4, 7, 5, 3, 2, 9]`

3. **Test Case 3**

   Categories: L3 $\wedge$ L7

   Input: `list = [9, 8, 7, 6, 5, 4, 3, 2, 1]`

2.2 `Collections.rotate(List<?> list, int distance)`

1. **Test Case 1**

   Categories: L1, D2 $\wedge$ D5

   Input: `list = [], distance = 0`

2. **Test Case 2**

   Categories: L3 $\wedge$ L5, D3 $\wedge$ D4 $\wedge$ D8

   Input: `list = [1, 2, 'a', 'b', 3.7, 4, "str"], distance =` $2^{31} - 1$

3. **Test Case 3**

   Categories: L3 ∧ L8, D1 ∧ D6 ∧ D7

   Input: `list = [1, 2, ..., 200, 201]`, `distance` = $2^{31} - 1$

2.3 `Collections.min(Collection<? extends T> coll)`

1. **Test Case 1**

   Categories: C1

   Input: `coll` = {`1`}

2. **Test Case 2**

   Categories: C2 ∧ C7

   Input: `coll` = {`1, 2, 3, 4, 5, 6, 7, 8, 9`}

3. **Test Case 3**

   Categories: C2 ∧ C3 ∧ C4 ∧ C5

   Input: `coll` = {`6, 8, 2, 4, 7, Integer.MIN_VALUE, 5, 3, 2, 9`}

3 Metamorphic Relations

3.1 `Collections.sort(List<T> list)`

The return type of `Collections.sort` is `void` [4], the input list is modified when the method is called.

```
List x = new ArrayList<>(Arrays.asList(x_0, x_1, ..., x_n));
List xTransformed = new ArrayList<>(x); // defined as x' below

... // see input transformation below

// x and xTransformed are modified directly after method call
Collections.sort(x);
Collections.sort(xTransformed);
```

Description	Input Transformation	Relation
Reverse the original input list	`Collections.reverse(x')`	`x'.equals(x)`
Double the size and content of original input by adding itself	`x'.addAll(x);`	`x'.size() == 2 * x.size`, `x'[2n] == x[n]`, `x'[2n+1] == x[n]`

3.2 `Collections.rotate(List<?> list, int distance)`

The return type of `Collections.rotate` is `void` [3], the input list is modified when the method is called.

```
List x = new ArrayList<>(Arrays.asList(x_0, x_1, ..., x_n));
int d = ...;

List xTransformed = new ArrayList<>(x); // defined as x' below
int dTransformed = ...; // defined as d' below

... // see input transformation below

// x and xTransformed are modified directly after method call
Collections.rotate(x, d);
Collections.rotate(xTransformed, dTransformed);
```

Description	Input Transformation	Relation
Reverse the original input list, convert distance to opposite sign	`Collections.reverse(x')` `dTransformed = -d`	After reversing `x'` again, `x'.equals(x)`
Add the value of `d * x.size()` to distance	`dTransformed = d * (1 + x.size())`	`x'.equals(x)`

3.3 `Collections.min(Collection<? extends T> coll)`

The return type of `Collections.min` is type of the elements in the list [2].

```
List x = new ArrayList<>(Arrays.asList(x_0, x_1, ..., x_n));
List xTransformed = new ArrayList<>(x); // defined as x' below

... // see input transformation below

// <T> is a placeholder for the type of elements in the list
<T> z = Collections.min(x);
<T> zTransformed = Collections.min(xTransformed);
```

Description	Input Transformation	Relation
Add an element smaller than the minimum element in the original collection	`x'.add(`x`)`	`z' < z`
Add an element greater than the minimum element in the original collection	`x'.add(`x`)`	`z' == z`

N.B. For the first relation, if the minimum element in the original collection is already the minimum possible elements for the class, then the new element added is equal to the minimum element in the original collection. The relation will then be `z' == z`.

4 Remarks

References

[1] *Arraylist (java platform se8)*, Oracle Corporation, 6th Dec. 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html.

[2] *Collections min (java platform se8)*, Oracle Corporation, 5th Dec. 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#min-java.util.Collection-.

[3] *Collections rotate (java platform se8)*, Oracle Corporation, 5th Dec. 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#rotate-java.util.List-int-.

[4] *Collections sort (java platform se8)*, Oracle Corporation, 5th Dec. 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#sort-java.util.List-.

[5] *Comparable (java platform se 8)*, Oracle Corporation, 5th Dec. 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html.

[6] *Comparable (java platform se8)*, Oracle Corporation, 6th Dec. 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html.

[7] *Integer max value*, Oracle Corporation, 5th Dec. 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html#MAX_VALUE.

[8] *Integer min value*, Oracle Corporation, 5th Dec. 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html#MIN_VALUE.

[9] *Java collection class and interface hierarchy*, JavaTpoint, 4th Dec. 2019. [Online]. Available: https://www.javatpoint.com/collections-in-java.

[10] *Jdk/collections.java rotate_threshold branch*, Oracle Corporation, 5th Dec. 2019. [Online]. Available: https://github.com/openjdk/jdk/blob/81ec9e3087764708/src/java.base/share/classes/java/util/Collections.java#L779.

[11] *Jdk/collections.java rotate_threshold variable*, Oracle Corporation, 5th Dec. 2019. [Online]. Available: https://github.com/openjdk/jdk/blob/81ec9e3087764708/src/java.base/share/classes/java/util/Collections.java#L109.

[12] *Linkedlist (java platform se8)*, Oracle Corporation, 6th Dec. 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html.

[13] *Vector (java platform se8)*, Oracle Corporation, 6th Dec. 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html.

[14] *Vector (java platform se8)*, Oracle Corporation, 6th Dec. 2019. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/Set.html.