1.Describe the Java Collections Framework. List the interfaces, convenience

abstract classes, and concrete classes under the Collection interface.

(use online resoure)

The Java Collections Framework defines the Java API for handling common data structures tasks in Java. It defines classes and interfaces for storing and manipulating data in sets, lists, and maps.

A convenience class is an abstract class that partially implements an interface. The Java Collections Framework defines interfaces, convenience abstract classes, and concrete classes.

2. What method do you use to obtain an element in the collection from an iterator?

Use the next() method.

3. What are the differences between ArrayList and LinkedList?

Which list should you use to insert and delete elements at the beginning of a list?

ArrayList and LinkedList can be operated similarly. The critical differences between them are their internal implementation, which impacts the performance. ArrayList is efficient for retrieving elements, and for adding and removing elements from the end of the list. LinkedList is efficient for adding and removing elements anywhere in the list.

4. How do you create a list from an array of objects?

new LinkedList(Arrays.asList(arrayObject))

5. What are the differences between the Comparable interface and the Comparator interface?

In which package is Comparable, and in which package is Comparator?

The Comparable interface contains the compareTo method and Comparator interface contains the compare method and equals method. Normally, if the objects of a class have natural order (e.g., String, Date), let the class implement the Comparable interface. The Comparator interface is more flexible in the sense that it enables you to define a new class that contains the compare(Object, Object) method to compare two objects of other classes.

The Comparable interface is in the java.lang package, and the Comparator interface is in the java.util package.

6. Write a statement that sorts an ArrayList of strings named list in increasing order of their last character.

list.sort((e1, e2) -> {

if (e1.length() == 0)

return -1;

else (e2.length() == 0)

return 1;

else

return charAt(e1.size() - 1) - charAt(e2.size() - 1);

}

7. Write a statement to find the largest element in an array of comparable objects.

Collections.max(Arrays.asList(arrayObject))

8. How do you create an instance of Stack?

How do you add a new element to a stack?

How do you remove an element from a stack?

How do you find the size of a stack?

Stack is a subclass of Vector. The Stack class represents a last-in-first-out stack of objects. The elements are accessed only from the top of the stack. You can retrieve, insert, or remove an element from the top of the stack. To add a new element to a stack, use the push method. To remove an element from the top of the stack, use the method pop. To find a stack size, use the size() method.

9. How do you create a priority queue that reverses the natural order of the elements?

new PriorityQueue(initialCapacity, Collections.reverseOrder()).

Programming Question:

10. (Perform set operations on priority queues) Create two priority queues,

{"George", "Jim", "John", "Blake", "Kevin", "Michael"} and

{"George", "Katie", "Kevin", "Michelle", "Ryan"}

and find their union, difference, and intersection.

import java.util.\*;

public class UninDiffInter

{

public static void main(String[] args) {

PriorityQueue<String> queue1 = new PriorityQueue<>(Arrays.asList(

"George", "Jim", "John", "Blake", "Kevin", "Michael"));

PriorityQueue<String> queue2 = new PriorityQueue<>(Arrays.asList(

"George", "Katie", "Kevin", "Michelle", "Ryan"));

System.out.println("Set1: " + queue1);

System.out.println("Set2: " + queue2);

System.out.println("Union of sets: " + union(queue1, queue2));

System.out.println("Difference of sets: " + difference(queue1, queue2));

System.out.println("Intersection of sets: " + intersection(queue1, queue2));

}

private static <T> PriorityQueue<T> union(

PriorityQueue<T> set1, PriorityQueue<T> set2) {

PriorityQueue<T> union = new PriorityQueue<>(set1);

union.addAll(set2);

return union;

}

private static <T> PriorityQueue<T> difference(

PriorityQueue<T> set1, PriorityQueue<T> set2) {

PriorityQueue<T> difference = new PriorityQueue<>(set1);

difference.removeAll(set2);

return difference;

}

private static <T> PriorityQueue<T> intersection(

PriorityQueue<T> set1, PriorityQueue<T> set2) {

PriorityQueue<T> intersection = new PriorityQueue<>(set1);

intersection.retainAll(set2);

return intersection;

}

}