

Class List

[Java AbstractCollection](#)
[Java AbstractList](#)
[Java AbstractMap](#)
[Java AbstractQueue](#)
[Java AbstractSequentialList](#)
[Java AbstractSet](#)
[Java ArrayDeque](#)
[Java ArrayList](#)
[Java ArrayList Util](#)
[Java Arrays](#)
[Java Base64](#)
[Java BitSet](#)
[Java Calendar](#)
[Java Collection](#)
[Java Collection Util](#)
[Java Collections](#)
[Java Comparator](#)
[Java ConcurrentModificationException](#)
[Java Currency](#)
[Java Date](#)
[Java Date Duration](#)
[Java Date Parse](#)
[Java Date Time](#)
[Java Date Week](#)
[Java Deque](#)
[Java Dictionary](#)
[Java DoubleSummaryStatistics](#)
[Java DuplicateFormatFlagsException](#)
[Java EmptyStackException](#)
[Java Enumeration](#)
[Java EnumMap](#)
[Java EnumSet](#)
[Java EventListener](#)
[Java EventObject](#)
[Java Formattable](#)
[Java FormattableFlags](#)
[Java Formatter](#)
[Java GregorianCalendar](#)
[Java GUID](#)
[Java HashMap](#)
[Java HashSet](#)
[Java Hashtable](#)
[Java IdentityHashMap](#)
[Java IllegalFormatConversionException toString\(\). Returns a short description of this throwable.](#)
[Java IllegalFormatException](#)
[Java IllegalFormatPrecisionException](#)

Java List Computes Jaccard coefficient between the sets of the keyword-attribute combinations

[Previous](#)

[Next](#)

Computes Jaccard coefficient between the sets of the keyword-attribute combinations contained in the both queries keywords not occurring in the query are bound to a zero-attribute

Return:

0 for dissimilar queries

1 for the highest possible similarity

[Java](#)
[IllegalFormatWidthException](#)
[Java InputMismatchException](#)
[Java IntSummaryStatistics](#)
[Java](#)
[InvalidPropertiesFormatException](#)
[Java Iterator](#)
[Java LinkedHashMap](#)
[Java LinkedHashSet](#)
[Java LinkedList](#)
[Java List](#)
[Java List Util](#)
[Java ListIterator](#)
[Java ListResourceBundle](#)
[Java Locale](#)
[Java LongSummaryStatistics](#)
[Java Map](#)
[Java](#)
[MissingFormatArgumentException](#)
[Java MissingResourceException](#)
[Java NavigableMap](#)
[Java NavigableSet](#)
[Java NoSuchElementException](#)
[Java Objects](#)
[Java Observable](#)
[Java Observer](#)
[Java Optional](#)
[Java OptionalDouble](#)
[Java OptionalInt](#)
[Java OptionalLong](#)
[Java PrimitivesIterator](#)
[Java PriorityQueue](#)
[Java Properties](#)
[Java PropertyPermission](#)
[Java PropertyResourceBundle](#)
[Java Queue](#)
[Java Random](#)
[Java RandomAccess tutorial with examples](#)
[Java ResourceBundle](#)
[Java Scanner](#)
[Java ServiceConfigurationError](#)
[Java ServiceLoader](#)
[Java Set](#)
[Java SimpleTimeZone](#)
[Java SortedMap](#)
[Java SortedSet](#)
[Java Spliterator](#)
[Java Spliterators](#)
[Java Stack](#)
[Java StringJoiner](#)
[Java StringTokenizer](#)
[Java Timer](#)
[Java TimerTask](#)
[Java TimeZone](#)

[Copy](#)

```
//package com.java2s;

import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class Main {
    /**/* w w w . d e m o 2 s . c o m */
    * Computes Jaccard coefficient between the sets of the keyword-
attribute
    * combinations contained in the both queries keywords not occurring
in the
    * query are bound to a zero-attribute
    *
    * @return 0 for dissimilar queries
    * @return 1 for the highest possible similarity
    */
    public static double getJaccardSimilarity(List<String> text_a,
List<String> text_b) {

        // determine the query with the bigger number of interpretations
        List<String> combi_big = text_a.size() >= text_b.size() ? text_a
: text_b;

        // determine the query with the smaller number of interpretations
        List<String> combi_small = text_a.size() >= text_b.size() ?
text_b : text_a;

        // build intersection
        Set<String> intersection = new HashSet<String>();
        // go through the smaller set of interpretations
        for (String keyword : combi_small) {
            // interpretation of the keyword in smaller set
            if (combi_big.contains(keyword)) {
                intersection.add(keyword);
            }
        }
        // build disjunction
        Set<String> disjunction = new HashSet<String>();
        disjunction.addAll(combi_small);
        disjunction.addAll(combi_big);

        // Build an intersection A AND B, compute the size
        // Build a disjunction A ODER B, compute the size
        // JC= (A AND B) / (A ODER B)
        double intersection_size = intersection.size();
        double disjunction_size = disjunction.size();

        // String s1 = q1.toFullString();
        // String s2 = q2.toFullString();

        double JC = intersection_size / disjunction_size;

        /*
```

[Java](#)
[TooManyListenersException](#)
[Java TreeMap](#)
[Java TreeSet](#)
[Java](#)
[UnknownFormatConversionException](#)
[Java UUID](#)
[Java Vector](#)
[Java WeakHashMap](#)

```
* System.out.println("query 1: " + q1.toFullString());
* System.out.println("query 2: " + q2.toFullString());
* System.out.println("AND Size: " +
intersection.keySet().size());
* System.out.println("OR Size: " + disjunction.keySet().size());
* System.out.println("JC: " + JC + "\n");
*/

return JC;
}

public static double getJaccardSimilarity(Set<String> text_a,
Set<String> text_b) {

    // determine the query with the bigger number of interpretations
    Set<String> combi_big = text_a.size() >= text_b.size() ? text_a :
text_b;

    // determine the query with the smaller number of interpretations
    Set<String> combi_small = text_a.size() >= text_b.size() ? text_b
: text_a;

    // build intersection
    Set<String> intersection = new HashSet<String>();
    // go through the smaller set of interpretations
    for (String keyword : combi_small) {
        // interpretation of the keyword in smaller set
        if (combi_big.contains(keyword)) {
            intersection.add(keyword);
        }
    }

    // build disjunction
    Set<String> disjunction = new HashSet<String>();
    disjunction.addAll(combi_small);
    disjunction.addAll(combi_big);

    // Build an intersection A AND B, compute the size
    // Build a disjunction A OR B, compute the size
    // JC= (A AND B) / (A OR B)
    double intersection_size = intersection.size();
    double disjunction_size = disjunction.size();

    // String s1 = q1.toFullString();
    // String s2 = q2.toFullString();

    double JC = intersection_size / disjunction_size;

    /*
    * System.out.println("query 1: " + q1.toFullString());
    * System.out.println("query 2: " + q2.toFullString());
    * System.out.println("AND Size: " +
intersection.keySet().size());
    * System.out.println("OR Size: " + disjunction.keySet().size());
    * System.out.println("JC: " + JC + "\n");
    */
}
```

```
        */  
  
        return JC;  
    }  
}
```

[Previous](#)[Next](#)

Related

- [Java List write a list of lines to a byte\[\] as UTF-8 encoded chars](#)
- [Java List Checks whether a given list of strings consists of integers.](#)
- [Java List closest\(int of, List<Integer> in\).](#)
- [Java List Computes Jaccard coefficient between the sets of the keyword-attribute combinations](#)
- [Java List Converts a list of strings into a long string separated by glue.](#)
- [Java List Create a string formulated by inserting a delimiter in between consecutive array elements.](#)
- [Java List Creates a comma-separated string from a list of strings.](#)