JAVA.UTIL.TREEMAP.SUBMAP METHOD

Description

The **subMap***KfromKey, booleanfromInclusive, KtoKey, booleantoInclusive* method is used to return a view of the portion of this map whose keys range from fromKey to toKey. If fromKey and toKey are equal, the returned map is empty unless fromExclusive and toExclusive are both true. The returned map is backed by this map, so changes in the returned map are reflected in this map, and vice-versa.

Declaration

Following is the declaration for **java.util.TreeMap.subMap** method.

```
public NavigableMap<K,V> subMap(K fromKey,
                                boolean fromInclusive,
                                K toKey,
                                boolean toInclusive)
```

Parameters

- **fromKey** -- This is the low endpoint of the keys in the returned map.
- **fromInclusive** -- This is true if the low endpoint is to be included in the returned view.
- **toKey** -- This is the high endpoint of the keys in the returned map.
- **toInclusive** -- This is true if the high endpoint is to be included in the returned view.

Return Value

The method call returns a view of the portion of this map whose keys range from fromKey to toKey.

Exception

- **ClassCastException** -- is exception is thrown if fromKey and toKey cannot be compared to one another using this map's comparator.
- **NullPointerException** -- This exception is thrown if fromKey or toKey is null and this map uses natural ordering, or its comparator does not permit null keys.
- **IllegalArgumentException** -- This exception is thrown if fromKey is greater than toKey; or if this map itself has a restricted range, and fromKey or toKey lies outside the bounds of the range.

Example

The following example shows the usage of java.util.TreeMap.subMap

```
package com.tutorialspoint;

import java.util.*;

public class TreeMapDemo {
   public static void main(String[] args) {
   // creating maps
   TreeMap<Integer, String> treemap = new TreeMap<Integer, String>();
   NavigableMap<Integer, String> treemapincl = new TreeMap<Integer, String>();

   // populating tree map
   treemap.put(2, "two");
   treemap.put(1, "one");
   treemap.put(3, "three");
   treemap.put(6, "six");
```

```
   treemap.put(5, "five");

   System.out.println("Getting a portion of the map");
   treemapincl=treemap.subMap(1, true, 3, true);
   System.out.println("Sub map values: "+treemapincl);
   }
}
```

Let us compile and run the above program, this will produce the following result.

```
Getting a portion of the map
Sub map values: {1=one, 2=two, 3=three}
```