GUAVA - RANGE CLASS

Range represents an interval or a sequence. It is used to get a set of numbers/ strings lying in a particular range.

Class Declaration

Following is the declaration for **com.google.common.collect.Range<C>** class:

```
@GwtCompatible
public final class Range<C extends Comparable>
   extends Object
      implements Predicate<C>, Serializable
```

Methods

Sr.No	Method & Description
1	**static <C extends Comparable<?>> Range<C> all** Returns a range that contains every value of type C.
2	**boolean apply*Cinput*Deprecated.** Provided only to satisfy the Predicate interface; use contains*C* instead.
3	**static <C extends Comparable<?>> Range<C> atLeast*Cendpoint*** Returns a range that contains all values greater than or equal to endpoint.
4	**static <C extends Comparable<?>> Range<C> atMost*Cendpoint*** Returns a range that contains all values less than or equal to endpoint.
5	**Range<C> canonical*DiscreteDomain* < *C* > *domain*** Returns the canonical form of this range in the given domain.
6	**static <C extends Comparable<?>> Range<C> closed*Clower, Cupper*** Returns a range that contains all values greater than or equal to lower and less than or equal to upper.
7	**static <C extends Comparable<?>> Range<C> closedOpen*Clower, Cupper*** Returns a range that contains all values greater than or equal to lower and strictly less than upper.
8	**boolean contains*Cvalue*** Returns true if value is within the bounds of this range.
9	**boolean containsAll*Iterable* < ?*extends*C > *values*** Returns true if every element in values is contained in this range.

10 **static <C extends Comparable<?>> Range<C> downTo**
Cendpoint, BoundTypeboundType

Returns a range from the given endpoint, which may be either inclusive *closed* or exclusive *open*, with no upper bound.

11 **static <C extends Comparable<?>> Range<C> encloseAll***Iterable < C > values*

Returns the minimal range that contains all of the given values.

12 **boolean encloses***Range < C > other*

Returns true if the bounds of other do not extend outside the bounds of this range.

13 **boolean equals***Objectobject*

Returns true if object is a range having the same endpoints and bound types as this range.

14 **static <C extends Comparable<?>> Range<C> greaterThan***Cendpoint*

Returns a range that contains all values strictly greater than endpoint.

15 **int hashCode**

Returns a hash code for this range.

16 **boolean hasLowerBound**

Returns true if this range has a lower endpoint.

17 **boolean hasUpperBound**

Returns true if this range has an upper endpoint.

18 **Range<C> intersection***Range < C > connectedRange*

Returns the maximal range enclosed by both this range and connectedRange, if such a range exists.

19 **boolean isConnected***Range < C > other*

Returns true if there exists a *possiblyempty* range which is enclosed by both this range and other.

20 **boolean isEmpty**

Returns true if this range is of the form [v..v) or (v..v].

21 **static <C extends Comparable<?>> Range<C> lessThan***Cendpoint*

Returns a range that contains all values strictly less than endpoint.

22 **BoundType lowerBoundType**

Returns the type of this range's lower bound: BoundType.CLOSED if the range includes

its lower endpoint, BoundType.OPEN if it does not.

23 **C lowerEndpoint**

Returns the lower endpoint of this range.

24 **static <C extends Comparable<?>> Range<C> open***Clower, Cupper*

Returns a range that contains all values strictly greater than lower and strictly less than upper.

25 **static <C extends Comparable<?>> Range<C> openClosed***Clower, Cupper*

Returns a range that contains all values strictly greater than lower and less than or equal to upper.

26 **static <C extends Comparable<?>> Range<C> range**
Clower, BoundTypelowerType, Cupper, BoundTypeupperType

Returns a range that contains any value from lower to upper, where each endpoint may be either inclusive *closed* or exclusive *open*.

27 **static <C extends Comparable<?>> Range<C> singleton***Cvalue*

Returns a range that contains only the given value.

28 **Range<C> span***Range < C > other*

Returns the minimal range that encloses both this range and other.

29 **String toString**

Returns a string representation of this range, such as "[3..5)"
otherexamplesarelistedintheclassdocumentation.

30 **BoundType upperBoundType**

Returns the type of this range's upper bound: BoundType.CLOSED if the range includes its upper endpoint, BoundType.OPEN if it does not.

31 **C upperEndpoint**

Returns the upper endpoint of this range.

32 **static <C extends Comparable<?>> Range<C> upTo***Cendpoint, BoundTypeboundType*

Returns a range with no lower bound up to the given endpoint, which may be either inclusive *closed* or exclusive *open*.

Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

Example of Range class

Create the following java program using any editor of your choice in say **C:/> Guava.**

GuavaTester.java

```
import com.google.common.collect.ContiguousSet;
import com.google.common.collect.DiscreteDomain;
import com.google.common.collect.Range;
import com.google.common.primitives.Ints;

public class GuavaTester {

   public static void main(String args[]) {
      GuavaTester tester = new GuavaTester();
      tester.testRange();
   }

   private void testRange() {

      //create a range [a,b] = { x | a <= x <= b}
      Range<Integer> range1 = Range.closed(0, 9);
      System.out.print("[0,9] : ");
      printRange(range1);

      System.out.println("5 is present: " + range1.contains(5));
      System.out.println("(1,2,3) is present: " + range1.containsAll(Ints.asList(1, 2,
3)));
      System.out.println("Lower Bound: " + range1.lowerEndpoint());
      System.out.println("Upper Bound: " + range1.upperEndpoint());

      //create a range (a,b) = { x | a < x < b}
      Range<Integer> range2 = Range.open(0, 9);
      System.out.print("(0,9) : ");
      printRange(range2);

      //create a range (a,b] = { x | a < x <= b}
      Range<Integer> range3 = Range.openClosed(0, 9);
      System.out.print("(0,9] : ");
      printRange(range3);

      //create a range [a,b) = { x | a <= x < b}
      Range<Integer> range4 = Range.closedOpen(0, 9);
      System.out.print("[0,9) : ");
      printRange(range4);

      //create an open ended range (9, infinity
      Range<Integer> range5 = Range.greaterThan(9);
      System.out.println("(9,infinity) : ");
      System.out.println("Lower Bound: " + range5.lowerEndpoint());
      System.out.println("Upper Bound present: " + range5.hasUpperBound());

      Range<Integer> range6 = Range.closed(3, 5);
      printRange(range6);

      //check a subrange [3,5] in [0,9]
      System.out.println("[0,9] encloses [3,5]:" + range1.encloses(range6));

      Range<Integer> range7 = Range.closed(9, 20);
      printRange(range7);

      //check ranges to be connected
      System.out.println("[0,9] is connected [9,20]:" + range1.isConnected(range7));

      Range<Integer> range8 = Range.closed(5, 15);

      //intersection
      printRange(range1.intersection(range8));

      //span
      printRange(range1.span(range8));
   }
```

```
   private void printRange(Range<Integer> range) {

      System.out.print("[ ");

      for(int grade : ContiguousSet.create(range, DiscreteDomain.integers())) {
         System.out.print(grade +" ");
      }
      System.out.println("]");
   }
}
```

Verify the Result

Compile the class using **javac** compiler as follows:

```
C:\Guava>javac GuavaTester.java
```

Now run the GuavaTester to see the result.

```
C:\Guava>java GuavaTester
```

See the result.

```
[0,9] : [ 0 1 2 3 4 5 6 7 8 9 ]
5 is present: true
(1,2,3) is present: true
Lower Bound: 0
Upper Bound: 9
(0,9) : [ 1 2 3 4 5 6 7 8 ]
(0,9] : [ 1 2 3 4 5 6 7 8 9 ]
[0,9) : [ 0 1 2 3 4 5 6 7 8 ]
(9,infinity) :
Lower Bound: 9
Upper Bound present: false
[ 3 4 5 ]
[0,9] encloses [3,5]:true
[ 9 10 11 12 13 14 15 16 17 18 19 20 ]
[0,9] is connected [9,20]:true
[ 5 6 7 8 9 ]
[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ]
```