# GUAVA
java collections framework

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

## About the Tutorial

Guava is an open source, Java-based library developed by Google. It facilitates best coding practices and helps reduce coding errors. It provides utility methods for collections, caching, primitives support, concurrency, common annotations, string processing, I/O, and validations.

This tutorial adopts a simple and intuitive way to describe the basic-to-advanced concepts of Guava and how to use its APIs.

## Audience

This tutorial will be useful for most Java developers, starting from beginners to experts. After completing this tutorial, we are confident you will find it easy to use Guava in your programs.

## Prerequisites

Prior exposure to Java programming is the only requirement to make the most of this tutorial.

## Copyright & Disclaimer

© Copyright 2014 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd.  The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

## Table of Contents

tutorialspoint
SIMPLYEASYLEARNING

# 1. Guava – Overview

## What is Guava?

Guava is an open source, Java-based library and contains many core libraries of Google, which are being used in many of their projects. It facilitates best coding practices and helps reduce coding errors. It provides utility methods for collections, caching, primitives support, concurrency, common annotations, string processing, I/O, and validations.

## Benefits of Guava

- **Standardized** – The Guava library is managed by Google.
- **Efficient** - It is a reliable, fast, and efficient extension to the Java standard library.
- **Optimized** – The library is highly optimized.
- **Functional Programming** - It adds functional processing capability to Java.
- **Utilities** - It provides many utility classes which are regularly required in programming application development.
- **Validation** - It provides a standard failsafe validation mechanism.
- **Best Practices** – It emphasizes on best practices.

Consider the following code snippet.

```
public class GuavaTester {
   public static void main(String args[]){
      GuavaTester guavaTester = new GuavaTester();
      Integer a =  null;
      Integer b =  new Integer(10);
      System.out.println(guavaTester.sum(a,b));
   }

   public Integer sum(Integer a, Integer b){
      return a + b;
   }
}
```

Run the program to get the following result.

```
Exception in thread "main" java.lang.NullPointerException
     at GuavaTester.sum(GuavaTester.java:13)
     at GuavaTester.main(GuavaTester.java:9)
```

Following are the problems with the code.

- sum() is not taking care of any of the parameters to be passed as null.
- caller function is also not worried about passing a null to the sum() method accidently.
- When the program runs, NullPointerException occurs.

In order to avoid the above problems, null check is to be made in each and every place where such problems are present.

Let's see the use of Optional, a Guava provided Utility class, to solve the above problems in a standardized way.

```
import com.google.common.base.Optional;


public class GuavaTester {
   public static void main(String args[]){
      GuavaTester guavaTester = new GuavaTester();


      Integer invalidInput = null;
      Optional<Integer> a =  Optional.of(invalidInput);
      Optional<Integer> b =  Optional.of(new Integer(10));
      System.out.println(guavaTester.sum(a,b));
   }
   public Integer sum(Optional<Integer> a, Optional<Integer> b){
      return a.get() + b.get();
   }
}
```

Run the program to get the following result.

```
Exception in thread "main" java.lang.NullPointerException
	at
com.google.common.base.Preconditions.checkNotNull(Preconditions.java:210)
	at com.google.common.base.Optional.of(Optional.java:85)
	at GuavaTester.main(GuavaTester.java:8)
```

Let's understand the important concepts of the above program.

- **Optional** - A utility class, to make the code use the null properly.
- **Optional.of** - It returns the instance of Optional class to be used as a parameter. It checks the value passed, not to be 'null'.
- **Optional.get** - It gets the value of the input stored in the Optional class.

Using the Optional class, you can check whether the caller method is passing a proper parameter or not.

# 2. Guava – Environment Setup

## Try it Option Online

We have setup a Java Programming environment online, so that you can compile and execute all the available examples online at the same time when you are doing your theory work. This gives you confidence in what you are reading and to check the result with different options. Feel free to modify any example and execute it online.

Try following example using the **Try it** (http://compileonline.com/) option available at the top right corner of the code boxes in our website:

```
public class MyFirstJavaProgram {


    public static void main(String []args) {
       System.out.println("Hello World");
    }
}
```

For most of the examples given in this tutorial, you will find a **Try it** option, provided with the sole intention to make your learning more enjoyable.

## Local Environment Setup

If you are still willing to set up your environment for Java programming language, then this section guides you on how to download and set up Java on your machine. Please follow the steps mentioned below to set up the environment.

Java SE is freely available from the link http://www.oracle.com/technetwork/java/javase/downloads/index-jdk5-jsp-142662.html. So you download a version based on your operating system.

Follow the instructions to download Java and run the **.exe** to install Java on your machine. Once you have installed Java on your machine, you would need to set environment variables to point to correct installation directories:

## Setting up the Path for Windows 2000/XP

We are assuming that you have installed Java in *c:\Program Files\java\jdk* directory:

- Right-click on 'My Computer' and select 'Properties'.
- Click on the 'Environment variables' button under the 'Advanced' tab.
- Now, alter the 'Path' variable so that it also contains the path to the Java executable. Example, if the path is currently set to 'C:\WINDOWS\SYSTEM32', then change your path to read 'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'.

## Setting up the Path for Windows 95/98/ME

We are assuming that you have installed Java in *c:\Program Files\java\jdk* directory:

- Edit the 'C:\autoexec.bat' file and add the following line at the end: 'SET PATH=%PATH%;C:\Program Files\java\jdk\bin'

## Setting up the Path for Linux, UNIX, Solaris, FreeBSD

Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your shell documentation if you have trouble doing this.

Example, if you use *bash* as your shell, then you would add the following line to the end of your '.bashrc: export PATH=/path/to/java:$PATH'

## Popular Java Editors

To write your Java programs, you need a text editor. There are many sophisticated IDEs available in the market. But for now, you can consider one of the following:

- **Notepad:** On Windows machine you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.
- **Netbeans:** It is a Java IDE that is open-source and free which can be downloaded from **http://www.netbeans.org/index.html**.
- **Eclipse:** It is also a Java IDE developed by the eclipse open-source community and can be downloaded from **http://www.eclipse.org/**.

## Download Guava Archive

Download the latest version of Guava jar file from "guava-18.0.jar". At the time of writing this tutorial, we have downloaded *guava-18.0.jar* and copied it into C:\>Guava folder.

| OS | Archive name |
|---|---|
| Windows | guava-18.0.jar |
| Linux | guava-18.0.jar |
| Mac | guava-18.0.jar |

## Set Guava Environment

Set the **Guava_HOME** environment variable to point to the base directory location where Guava jar is stored on your machine. Assuming, we've extracted guava-18.0.jar in Guava folder on various Operating Systems as follows.

| OS | Output |
|---|---|
| Windows | Set the environment variable Guava_HOME to C:\Guava |
| Linux | export Guava_HOME=/usr/local/Guava |
| Mac | export Guava_HOME=/Library/Guava |

## Set CLASSPATH Variable

Set the **CLASSPATH** environment variable to point to the Guava jar location. Assuming, you have stored guava-18.0.jar in Guava folder on various Operating Systems as follows.

| OS | Output |
|---|---|
| Windows | Set the environment variable CLASSPATH to %CLASSPATH%;%Guava_HOME%\guava-18.0.jar;.; |
| Linux | export CLASSPATH=$CLASSPATH:$Guava_HOME/guava-18.0.jar:. |
| Mac | export CLASSPATH=$CLASSPATH:$Guava_HOME/guava-18.0.jar:. |

# 3. Guava – Optional Class

Optional is an immutable object used to contain a not-null object. Optional object is used to represent null with absent value. This class has various utility methods to facilitate the code to handle values as available or not available instead of checking null values.

## Class Declaration

Following is the declaration for **com.google.common.base.Optional<T>** class:

```
@GwtCompatible(serializable=true)
public abstract class Optional<T>
   extends Object
      implements Serializable
```

## Class Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static <T> Optional<T> absent()**<br>Returns an Optional instance with no contained reference. |
| 2 | **abstract Set<T> asSet()**<br>Returns an immutable singleton Set whose only element is the contained instance if it is present; an empty immutable Set otherwise. |
| 3 | **abstract boolean equals(Object object)**<br>Returns true if object is an Optional instance, and either the contained references are equal to each other or both are absent. |
| 4 | **static <T> Optional<T> fromNullable(T nullableReference)**<br>If nullableReference is non-null, returns an Optional instance containing that reference; otherwise returns absent(). |
| 5 | **abstract T get()**<br>Returns the contained instance, which must be present. |
| 6 | **abstract int hashCode()**<br>Returns a hash code for this instance. |
| 7 | **abstract boolean isPresent()**<br>Returns true if this holder contains a (non-null) instance. |
| 8 | **static <T> Optional<T> of(T reference)** |

| | |
|---|---|
| | Returns an Optional instance containing the given non-null reference. |
| 9 | **abstract Optional<T> or(Optional<? extends T> secondChoice)**<br>Returns this Optional if it has a value present; secondChoice otherwise. |
| 10 | **abstract T or(Supplier<? extends T> supplier)**<br>Returns the contained instance if it is present; supplier.get() otherwise. |
| 11 | **abstract T or(T defaultValue)**<br>Returns the contained instance if it is present; defaultValue otherwise. |
| 12 | **abstract T orNull()**<br>Returns the contained instance if it is present; null otherwise. |
| 13 | **static <T> Iterable<T> presentInstances(Iterable<? extends Optional<? extends T>> optionals)**<br>Returns the value of each present instance from the supplied optionals, in order, skipping over occurrences of absent(). |
| 14 | **abstract String toString()**<br>Returns a string representation for this instance. |
| 15 | **abstract <V> Optional<V> transform(Function<? super T,V> function)**<br>If the instance is present, it is transformed with the given Function; otherwise, absent() is returned. |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of Optional Class

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

## GuavaTester.java

```
import com.google.common.base.Optional;


public class GuavaTester {
   public static void main(String args[]){
      GuavaTester guavaTester = new GuavaTester();


      Integer value1 =  null;
      Integer value2 =  new Integer(10);
      //Optional.fromNullable - allows passed parameter to be null.
      Optional<Integer> a = Optional.fromNullable(value1);
      //Optional.of - throws NullPointerException if passed parameter is null
      Optional<Integer> b = Optional.of(value2);

      System.out.println(guavaTester.sum(a,b));
   }

   public Integer sum(Optional<Integer> a, Optional<Integer> b){
      //Optional.isPresent - checks the value is present or not
      System.out.println("First parameter is present: " + a.isPresent());

      System.out.println("Second parameter is present: " + b.isPresent());

      //Optional.or - returns the value if present otherwise returns
      //the default value passed.
      Integer value1 = a.or(new Integer(0));

      //Optional.get - gets the value, value should be present
      Integer value2 = b.get();

      return value1 + value2;
   }
}
```

## 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.

```
First parameter is present: false
Second parameter is present: true
10
```

# 4. Guava – Preconditions Class

Preconditions provide static methods to check that a method or a constructor is invoked with proper parameter or not. It checks the pre-conditions. Its methods throw IllegalArgumentException on failure.

## Class Declaration

Following is the declaration for **com.google.common.base.Preconditions** class:

```
@GwtCompatible

public final class Preconditions

   extends Object
```

## Class Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static void checkArgument(boolean expression)**<br>Ensures the truth of an expression involving one or more parameters to the calling method. |
| 2 | **static void checkArgument(boolean expression, Object errorMessage)**<br>Ensures the truth of an expression involving one or more parameters to the calling method. |
| 3 | **static void checkArgument(boolean expression, String errorMessageTemplate, Object. errorMessageArgs)**<br>Ensures the truth of an expression involving one or more parameters to the calling method. |
| 4 | **static int checkElementIndex(int index, int size)**<br>Ensures that index specifies a valid element in an array, list or a string of size. |
| 5 | **static int checkElementIndex(int index, int size, String desc)**<br>Ensures that index specifies a valid element in an array, list, or a string of size. |
| 6 | **static &lt;T&gt; T checkNotNull(T reference)**<br>Ensures that an object reference passed as a parameter to the calling method is not null. |
| 7 | **static &lt;T&gt; T checkNotNull(T reference, Object errorMessage)**<br>Ensures that an object reference passed as a parameter to the calling method is not null. |

| | |
|---|---|
| 8 | **static <T> T checkNotNull(T reference, String errorMessageTemplate, Object... errorMessageArgs)**<br>Ensures that an object reference passed as a parameter to the calling method is not null. |
| 9 | **static int checkPositionIndex(int index, int size)**<br>Ensures that index specifies a valid position in an array, list or a string of size. |
| 10 | **static int checkPositionIndex(int index, int size, String desc)**<br>Ensures that index specifies a valid position in an array, list or a string of size. |
| 11 | **static void checkPositionIndexes(int start, int end, int size)**<br>Ensures that start and end specify a valid positions in an array, list or a string of size, and are in order. |
| 12 | **static void checkState(boolean expression)**<br>Ensures the truth of an expression involving the state of the calling instance, but not involving any parameters to the calling method. |
| 13 | **static void checkState(boolean expression, Object errorMessage)**<br>Ensures the truth of an expression involving the state of the calling instance, but not involving any parameters to the calling method. |
| 14 | **static void checkState(boolean expression, String errorMessageTemplate, Object... errorMessageArgs)**<br>Ensures the truth of an expression involving the state of the calling instance, but not involving any parameters to the calling method. |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of Preconditions Class

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

### GuavaTester.java

```
import com.google.common.base.Preconditions;


public class GuavaTester {


   public static void main(String args[]){
      GuavaTester guavaTester = new GuavaTester();
      try {
         System.out.println(guavaTester.sqrt(-3.0));
      }catch(IllegalArgumentException e){
         System.out.println(e.getMessage());
      }
      try {
         System.out.println(guavaTester.sum(null,3));
      }catch(NullPointerException e){
         System.out.println(e.getMessage());
      }
      try {
         System.out.println(guavaTester.getValue(6));
      }catch(IndexOutOfBoundsException e){
         System.out.println(e.getMessage());
      }
   }

   public double sqrt(double input) throws IllegalArgumentException {
      Preconditions.checkArgument(input > 0.0,
         "Illegal Argument passed: Negative value %s.", input);
      return Math.sqrt(input);
   }

   public int sum(Integer a, Integer b){
      a = Preconditions.checkNotNull(a,
         "Illegal Argument passed: First parameter is Null.");
      b = Preconditions.checkNotNull(b,
         "Illegal Argument passed: Second parameter is Null.");
      return a+b;
   }

   public int getValue(int input){
      int[] data = {1,2,3,4,5};
      Preconditions.checkElementIndex(input,data.length,
         "Illegal Argument passed: Invalid index.");
      return 0;
   }
}
```

## 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.

```
Illegal Argument passed: Negative value -3.0.
Illegal Argument passed: First parameter is Null.
Illegal Argument passed: Invalid index. (6) must be less than size (5)
```

# 5. Guava – Ordering Class

Ordering can be seen as an enriched comparator with enhanced chaining functionality, multiple utility methods, multi-type sorting capability, etc.

### Class Declaration

Following is the declaration for **com.google.common.collect.Ordering<T>** class:

```
@GwtCompatible
public abstract class Ordering<T>
   extends Object
      implements Comparator<T>
```

### Class Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static Ordering<Object> allEqual()**<br>Returns an ordering which treats all values as equal, indicating "no ordering." Passing this ordering to any stable sort algorithm results in no change to the order of elements. |
| 2 | **static Ordering<Object> arbitrary()**<br>Returns an arbitrary ordering over all objects, for which compare(a, b) == 0 implies a == b (identity equality). |
| 3 | **int binarySearch(List<? extends T> sortedList, T key)**<br>Searches sortedList for key using the binary search algorithm. |
| 4 | **abstract int compare(T left, T right)**<br>Compares its two arguments for order. |
| 5 | **<U extends T> Ordering<U> compound(Comparator<? super U> secondaryComparator)**<br>Returns an ordering which first uses the ordering this, but which in the event of a "tie", then delegates to secondaryComparator. |
| 6 | **static <T> Ordering<T> compound(Iterable<? extends Comparator<? super T>> comparators)**<br>Returns an ordering which tries each given comparator in order until a non-zero result is found, returning that result, and returning zero only if all comparators return zero. |

| | |
|---|---|
| 7 | **static <T> Ordering<T> explicit(List<T> valuesInOrder)**<br>Returns an ordering that compares objects according to the order in which they appear in the given list. |
| 8 | **static <T> Ordering<T> explicit(T leastValue, T... remainingValuesInOrder)**<br>Returns an ordering that compares objects according to the order in which they are given to this method. |
| 9 | **static <T> Ordering<T> from(Comparator<T> comparator)**<br>Returns an ordering based on an existing comparator instance. |
| 10 | **<E extends T> List<E> greatestOf(Iterable<E> iterable, int k)**<br>Returns the k greatest elements of the given iterable according to this ordering, in order from greatest to least. |
| 11 | **<E extends T> List<E> greatestOf(Iterator<E> iterator, int k)**<br>Returns the k greatest elements from the given iterator according to this ordering, in order from greatest to least. |
| 12 | **<E extends T> ImmutableList<E> immutableSortedCopy(Iterable<E> elements)**<br>Returns an immutable list containing elements sorted by this ordering. |
| 13 | **boolean isOrdered(Iterable<? extends T> iterable)**<br>Returns true if each element in iterable after the first is greater than or equal to the element that preceded it, according to this ordering. |
| 14 | **boolean isStrictlyOrdered(Iterable<? extends T> iterable)**<br>Returns true if each element in iterable after the first is strictly greater than the element that preceded it, according to this ordering. |
| 15 | **<E extends T> List<E> leastOf(Iterable<E> iterable, int k)**<br>Returns the k least elements of the given iterable according to this ordering, in order from least to greatest. |
| 16 | **<E extends T> List<E> leastOf(Iterator<E> elements, int k)** |
| | Returns the k least elements from the given iterator according to this ordering, in order from least to greatest. |
| 17 | **<S extends T> Ordering<Iterable<S>> lexicographical()**<br>Returns a new ordering which sorts iterables by comparing corresponding elements pairwise until a nonzero result is found; imposes "dictionary order". |
| 18 | **<E extends T> E max(E a, E b)**<br>Returns the greater of the two values according to this ordering. |

| | |
|---|---|
| 19 | **<E extends T> E max(E a, E b, E c, E... rest)**<br>Returns the greatest of the specified values according to this ordering. |
| 20 | **<E extends T> E max(Iterable<E> iterable)**<br>Returns the greatest of the specified values according to this ordering. |
| 21 | **<E extends T> E max(Iterator<E> iterator)**<br>Returns the greatest of the specified values according to this ordering. |
| 22 | **<E extends T> E min(E a, E b)**<br>Returns the lesser of the two values according to this ordering. |
| 23 | **<E extends T> E min(E a, E b, E c, E... rest)**<br>Returns the least of the specified values according to this ordering. |
| 24 | **<E extends T> E min(Iterable<E> iterable)**<br>Returns the least of the specified values according to this ordering. |
| 25 | **<E extends T> E min(Iterator<E> iterator)**<br>Returns the least of the specified values according to this ordering. |
| 26 | **static <C extends Comparable> Ordering<C> natural()**<br>Returns a serializable ordering that uses the natural order of the values. |
| 27 | **<S extends T> Ordering<S> nullsFirst()**<br>Returns an ordering that treats null as less than all other values and uses this to compare non-null values. |
| 28 | **<S extends T> Ordering<S> nullsLast()**<br>Returns an ordering that treats null as greater than all other values and uses this ordering to compare non-null values. |
| 29 | **<F> Ordering<F> onResultOf(Function<F,? extends T> function)**<br>Returns a new ordering on F which orders elements by first applying a function to them, then comparing those results using this. |
| 30 | **<S extends T> Ordering<S> reverse()**<br>Returns the reverse of this ordering; the Ordering equivalent to Collections.reverseOrder(Comparator). |
| 31 | **<E extends T> List<E> sortedCopy(Iterable<E> elements)**<br>Returns a mutable list containing elements sorted by this ordering; use this only when the resulting list may need further modification, or may contain null. |
| 32 | **static Ordering<Object> usingToString()**<br>Returns an ordering that compares objects by the natural ordering of their string representations as returned by toString(). |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of Ordering Class

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

## GuavaTester.java

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import com.google.common.collect.Ordering;

public class GuavaTester {
   public static void main(String args[]){
      List<Integer> numbers = new ArrayList<Integer>();
      numbers.add(new Integer(5));
      numbers.add(new Integer(2));
      numbers.add(new Integer(15));
      numbers.add(new Integer(51));
      numbers.add(new Integer(53));
      numbers.add(new Integer(35));
      numbers.add(new Integer(45));
      numbers.add(new Integer(32));
      numbers.add(new Integer(43));
      numbers.add(new Integer(16));

      Ordering ordering = Ordering.natural();
      System.out.println("Input List: ");
      System.out.println(numbers);

      Collections.sort(numbers,ordering );
      System.out.println("Sorted List: ");
      System.out.println(numbers);

      System.out.println("======================");
      System.out.println("List is sorted: " + ordering.isOrdered(numbers));
      System.out.println("Minimum: " + ordering.min(numbers));
      System.out.println("Maximum: " + ordering.max(numbers));

      Collections.sort(numbers,ordering.reverse());
      System.out.println("Reverse: " + numbers);

      numbers.add(null);
      System.out.println("Null added to Sorted List: ");
      System.out.println(numbers);

      Collections.sort(numbers,ordering.nullsFirst());
      System.out.println("Null first Sorted List: ");
      System.out.println(numbers);
      System.out.println("======================");

      List<String> names = new ArrayList<String>();
      names.add("Ram");
      names.add("Shyam");
      names.add("Mohan");
      names.add("Sohan");
      names.add("Ramesh");
      names.add("Suresh");
      names.add("Naresh");
      names.add("Mahesh");
      names.add(null);
      names.add("Vikas");
      names.add("Deepak");

      System.out.println("Another List: ");
      System.out.println(names);

       Collections.sort(names,ordering.nullsFirst().reverse());
      System.out.println("Null first then reverse sorted list: ");
      System.out.println(names);
   }
}
```

## 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.

```
Input List:
[5, 2, 15, 51, 53, 35, 45, 32, 43, 16]
Sorted List:
[2, 5, 15, 16, 32, 35, 43, 45, 51, 53]
======================
List is sorted: true
Minimum: 2
Maximum: 53
Reverse: [53, 51, 45, 43, 35, 32, 16, 15, 5, 2]
Null added to Sorted List:
[53, 51, 45, 43, 35, 32, 16, 15, 5, 2, null]
Null first Sorted List:
[null, 2, 5, 15, 16, 32, 35, 43, 45, 51, 53]
======================
Another List:
[Ram, Shyam, Mohan, Sohan, Ramesh, Suresh, Naresh, Mahesh, null, Vikas, Deepak]
Null first then reverse sorted list:
[Vikas, Suresh, Sohan, Shyam, Ramesh, Ram, Naresh, Mohan, Mahesh, Deepak, null]
```

# 6. Guava – Objects Class

Objects class provides helper functions applicable to all objects such as equals, hashCode, etc.

## Class Declaration

Following is the declaration for **com.google.common.base.Objects** class:

```
@GwtCompatible
public final class Objects
   extends Object
```

## Class Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static boolean equal(Object a, Object b)** Determines whether two possibly-null objects are equal. |
| 2 | **static <T> T firstNonNull(T first, T second)** Deprecated. Use MoreObjects.firstNonNull(T, T) instead. This method is scheduled for removal in June 2016. |
| 3 | **static int hashCode(Object... objects)** Generates a hash code for multiple values. |
| 4 | **static Objects.ToStringHelper toStringHelper(Class<?> clazz)** Deprecated. Use MoreObjects.toStringHelper(Class) instead. This method is scheduled for removal in June 2016. |
| 5 | **static Objects.ToStringHelper toStringHelper(Object self)** Deprecated. Use MoreObjects.toStringHelper(Object) instead. This method is scheduled for removal in June 2016. |
| 6 | **static Objects.ToStringHelper toStringHelper(String className)** Deprecated. Use MoreObjects.toStringHelper(String) instead. This method is scheduled for removal in June 2016. |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of Objects Class

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

### GuavaTester.java

```
import com.google.common.base.Objects;

public class GuavaTester {
   public static void main(String args[]){
      Student s1 = new Student("Mahesh", "Parashar", 1, "VI");
      Student s2 = new Student("Suresh", null, 3, null);

      System.out.println(s1.equals(s2));
      System.out.println(s1.hashCode());
      System.out.println(
      Objects.toStringHelper(s1)
         .add("Name",s1.getFirstName()+" " + s1.getLastName())
         .add("Class", s1.getClassName())
         .add("Roll No", s1.getRollNo())
         .toString());
   }
}

class Student {
   private String firstName;
   private String lastName;
   private int rollNo;
   private String className;

   public Student(String firstName, String lastName, int rollNo, String
   className){
      this.firstName = firstName;
      this.lastName = lastName;
      this.rollNo = rollNo;
      this.className = className;
   }
```

```
    @Override
    public boolean equals(Object object){
       if(!(object instanceof Student) || object == null){
          return false;
       }
       Student student = (Student)object;
       // no need to handle null here
       // Objects.equal("test", "test") == true
       // Objects.equal("test", null) == false
       // Objects.equal(null, "test") == false
       // Objects.equal(null, null) == true
       return Objects.equal(firstName, student.firstName) // first name can be null
         && Objects.equal(lastName, student.lastName) // last name can be null
         && Objects.equal(rollNo, student.rollNo)
         && Objects.equal(className, student.className);// class name can be null
    }

    @Override
    public int hashCode(){
       //no need to compute hashCode by self
       return Objects.hashCode(className,rollNo);
    }
    public String getFirstName() {
       return firstName;
    }
    public void setFirstName(String firstName) {
       this.firstName = firstName;
    }
    public String getLastName() {
       return lastName;
    }
    public void setLastName(String lastName) {
       this.lastName = lastName;
    }
    public int getRollNo() {
       return rollNo;
```

```
   }
   public void setRollNo(int rollNo) {
      this.rollNo = rollNo;
   }
   public String getClassName() {
      return className;
   }
   public void setClassName(String className) {
      this.className = className;
   }
}
```

### 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.

```
false
85871
Student{Name=Mahesh Parashar, Class=VI, Roll No=1}
```

# 7. 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

| S.N. | Method & Description |
|---|---|
| 1 | **static <C extends Comparable<?>> Range<C> all()**<br>Returns a range that contains every value of type C. |
| 2 | **boolean apply(C input)Deprecated.**<br>Provided only to satisfy the Predicate interface; use contains(C) instead. |
| 3 | **static <C extends Comparable<?>> Range<C> atLeast(C endpoint)**<br>Returns a range that contains all values greater than or equal to endpoint. |
| 4 | **static <C extends Comparable<?>> Range<C> atMost(C endpoint)**<br>Returns a range that contains all values less than or equal to endpoint. |
| 5 | **Range<C> canonical(DiscreteDomain<C> domain)**<br>Returns the canonical form of this range in the given domain. |
| 6 | **static <C extends Comparable<?>> Range<C> closed(C lower, C upper)**<br>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(C lower, C upper)**<br>Returns a range that contains all values greater than or equal to lower and strictly less than upper. |
| 8 | **boolean contains(C value)**<br>Returns true if value is within the bounds of this range. |

| | |
|---|---|
| 9 | **boolean containsAll(Iterable<? extends C> values)**<br>Returns true if every element in values is contained in this range. |
| 10 | **static <C extends Comparable<?>> Range<C> downTo(C endpoint, BoundType boundType)**<br>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)**<br>Returns the minimal range that contains all of the given values. |
| 12 | **boolean encloses(Range<C> other)**<br>Returns true if the bounds of other do not extend outside the bounds of this range. |
| 13 | **boolean equals(Object object)**<br>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(C endpoint)**<br>Returns a range that contains all values strictly greater than endpoint. |
| 15 | **int hashCode()**<br>Returns a hash code for this range. |
| 16 | **boolean hasLowerBound()**<br>Returns true if this range has a lower endpoint. |
| 17 | **boolean hasUpperBound()**<br>Returns true if this range has an upper endpoint. |
| 18 | **Range<C> intersection(Range<C> connectedRange)**<br>Returns the maximal range enclosed by both this range and connectedRange, if such a range exists. |
| 19 | **boolean isConnected(Range<C> other)**<br>Returns true if there exists a (possibly empty) range which is enclosed by both this range and other. |
| 20 | **boolean isEmpty()**<br>Returns true if this range is of the form [v..v) or (v..v]. |
| 21 | **static <C extends Comparable<?>> Range<C> lessThan(C endpoint)**<br>Returns a range that contains all values strictly less than endpoint. |
| 22 | **BoundType lowerBoundType()**<br>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()**<br>Returns the lower endpoint of this range. |
| 24 | **static <C extends Comparable<?>> Range<C> open(C lower, C upper)**<br>Returns a range that contains all values strictly greater than lower and strictly less than upper. |
| 25 | **static <C extends Comparable<?>> Range<C> openClosed(C lower, C upper)**<br>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(C lower, BoundType lowerType, C upper, BoundType upperType)**<br>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(C value)**<br>Returns a range that contains only the given value. |
| 28 | **Range<C> span(Range<C> other)**<br>Returns the minimal range that encloses both this range and other. |
| 29 | **String toString()**<br>Returns a string representation of this range, such as "[3..5)" (other examples are listed in the class documentation). |
| 30 | **BoundType upperBoundType()**<br>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()**<br>Returns the upper endpoint of this range. |
| 32 | **static <C extends Comparable<?>> Range<C> upTo(C endpoint, BoundType boundType)**<br>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

```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 ]
```

# 8. Guava – Throwables Class

Throwables class provides utility methods related to Throwable interface.

## Class Declaration

Following is the declaration for **com.google.common.base.Throwables** class:

```
public final class Throwables
   extends Object
```

## Class Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static List<Throwable> getCausalChain(Throwable throwable)** Gets a Throwable cause chain as a list. |
| 2 | **static Throwable getRootCause(Throwable throwable)** Returns the innermost cause of throwable. |
| 3 | **static String getStackTraceAsString(Throwable throwable)** Returns a string containing the result of toString(), followed by the full, recursive stack trace of throwable. |
| 4 | **static RuntimeException propagate(Throwable throwable)** Propagates throwable as-is if it is an instance of RuntimeException or Error, or else as a last resort, wraps it in a RuntimeException then propagates. |
| 5 | **static <X extends Throwable> void propagateIfInstanceOf(Throwable throwable, Class<X> declaredType)** Propagates throwable exactly as-is, if and only if it is an instance of declaredType. |
| 6 | **static void propagateIfPossible(Throwable throwable)** Propagates throwable exactly as-is, if and only if it is an instance of RuntimeException or Error. |
| 7 | **static <X extends Throwable> void propagateIfPossible(Throwable throwable, Class<X> declaredType)** Propagates throwable exactly as-is, if and only if it is an instance of RuntimeException, Error, or declaredType. |
| 8 | **static <X1 extends Throwable,X2 extends Throwable>void propagateIfPossible(Throwable throwable, Class<X1> declaredType1,** |

| | |
|---|---|
| | **Class<X2> declaredType2)** Propagates throwable exactly as-is, if and only if it is an instance of RuntimeException, Error, declaredType1, or declaredType2. |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of Throwables Class

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

### GuavaTester.java

```
import java.io.IOException;


import com.google.common.base.Objects;
import com.google.common.base.Throwables;


public class GuavaTester {
   public static void main(String args[]){
      GuavaTester tester = new GuavaTester();
      try {
         tester.showcaseThrowables();
      } catch (InvalidInputException e) {
         //get the root cause
         System.out.println(Throwables.getRootCause(e));
      }catch (Exception e) {
         //get the stack trace in string format
         System.out.println(Throwables.getStackTraceAsString(e));

      }


      try {
         tester.showcaseThrowables1();
      }catch (Exception e) {
         System.out.println(Throwables.getStackTraceAsString(e));

      }
   }
   public void showcaseThrowables() throws InvalidInputException{
      try {
         sqrt(-3.0);
      } catch (Throwable e) {
         //check the type of exception and throw it
         Throwables.propagateIfInstanceOf(e, InvalidInputException.class);

         Throwables.propagate(e);
      }
   }

   public void showcaseThrowables1(){
      try {
         int[] data = {1,2,3};
         getValue(data, 4);
      } catch (Throwable e) {
         Throwables.propagateIfInstanceOf(e, IndexOutOfBoundsException.class);

         Throwables.propagate(e);
      }
   }

   public double sqrt(double input) throws InvalidInputException{
      if(input < 0) throw new InvalidInputException();
      return Math.sqrt(input);
   }

   public double getValue(int[] list, int index) throws IndexOutOfBoundsException {
      return list[index];
   }

   public void dummyIO() throws IOException {
      throw new IOException();
   }
}

class InvalidInputException extends Exception { }
```

## 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.

```
InvalidInputException
java.lang.ArrayIndexOutOfBoundsException: 4
	at GuavaTester.getValue(GuavaTester.java:52)
	at GuavaTester.showcaseThrowables1(GuavaTester.java:38)
	at GuavaTester.main(GuavaTester.java:19)
```

# 9. Guava – Collections Utilities

Guava introduces many advanced collections based on developers' experience in application development works. Given below is a list of useful collections:

| S.N. | Collection name & Description |
|---|---|
| 1 | **Multiset**<br>An extension to Set interface to allow duplicate elements. |
| 2 | **Multimap**<br>An extension to Map interface so that its keys can be mapped to multiple values at a time. |
| 3 | **BiMap**<br>An extension to Map interface to support inverse operations. |
| 4 | **Table**<br>Table represents a special map where two keys can be specified in combined fashion to refer to a single value. |

## Multiset Interface

Multiset interface extends 'Set' to have duplicate elements, and provides various utility methods to deal with the occurrences of such elements in a set.

### Interface Declaration

Following is the declaration for **com.google.common.collect.Multiset<E>** interface:

```
@GwtCompatible
public interface Multiset<E>
   extends Collection<E>
```

### Interface Methods

| S.N. | Method & Description |
|---|---|
| 1 | **boolean add(E element)**<br>Adds a single occurrence of the specified element to this multiset. |
| 2 | **int add(E element, int occurrences)**<br>Adds a number of occurrences of an element to this multiset. |

tutorialspoint
SIMPLYEASYLEARNING

| | |
|---|---|
| 3 | **boolean contains(Object element)**<br>Determines whether this multiset contains the specified element. |
| 4 | **boolean containsAll(Collection<?> elements)**<br>Returns true if this multiset contains at least one occurrence of each element in the specified collection. |
| 5 | **int count(Object element)**<br>Returns the number of occurrences of an element in this multiset (the count of the element). |
| 6 | **Set<E> elementSet()**<br>Returns the set of distinct elements contained in this multiset. |
| 7 | **Set<Multiset.Entry<E>> entrySet()**<br>Returns a view of the contents of this multiset, grouped into Multiset.Entry instances, each providing an element of the multiset and the count of that element. |
| 8 | **boolean equals(Object object)**<br>Compares the specified object with this multiset for equality. |
| 9 | **int hashCode()**<br>Returns the hash code for this multiset. |
| 10 | **Iterator<E> iterator()**<br>Returns an iterator over the elements in this collection. |
| 11 | **boolean remove(Object element)**<br>Removes a single occurrence of the specified element from this multiset, if present. |
| 12 | **int remove(Object element, int occurrences)**<br>Removes a number of occurrences of the specified element from this multiset. |
| 13 | **boolean removeAll(Collection<?> c)**<br>Removes all of this collection's elements that are also contained in the specified collection (optional operation). |
| 14 | **boolean retainAll(Collection<?> c)**<br>Retains only the elements in this collection that are contained in the specified collection (optional operation). |
| 15 | **int setCount(E element, int count)**<br>Adds or removes the necessary occurrences of an element such that the element attains the desired count. |
| 16 | **boolean setCount(E element, int oldCount, int newCount)** |

| | |
|---|---|
| | Conditionally sets the count of an element to a new value, as described in setCount(Object, int), provided that the element has the expected current count. |
| 17 | **String toString()**<br>Returns a string representation of the object. |

## Methods Inherited

This interface inherits methods from the following interface:

- java.util.Collection

## Example of Multiset

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

### GuavaTester.java

```
import java.util.Iterator;
import java.util.Set;

import com.google.common.collect.HashMultiset;
import com.google.common.collect.Multiset;

public class GuavaTester {

   public static void main(String args[]){
      //create a multiset collection
      Multiset<String> multiset = HashMultiset.create();
      multiset.add("a");
      multiset.add("b");
      multiset.add("c");
      multiset.add("d");
      multiset.add("a");
      multiset.add("b");
      multiset.add("c");
      multiset.add("b");
      multiset.add("b");
      multiset.add("b");
      //print the occurrence of an element
      System.out.println("Occurrence of 'b' : "+multiset.count("b"));
      //print the total size of the multiset
      System.out.println("Total Size : "+multiset.size());
      //get the distinct elements of the multiset as set
      Set<String> set = multiset.elementSet();
      //display the elements of the set
      System.out.println("Set [");
      for (String s : set) {
         System.out.println(s);
      }
      System.out.println("]");
      //display all the elements of the multiset using iterator
      Iterator<String> iterator  = multiset.iterator();
      System.out.println("MultiSet [");
      while(iterator.hasNext()){
         System.out.println(iterator.next());
      }
      System.out.println("]");
      //display the distinct elements of the multiset with their occurrence count
      System.out.println("MultiSet [");
      for (Multiset.Entry<String> entry : multiset.entrySet())
      {
         System.out.println("Element: "+entry.getElement() +", Occurrence(s): " + entry.getCount());
      }
      System.out.println("]");

      //remove extra occurrences
      multiset.remove("b",2);
      //print the occurrence of an element
      System.out.println("Occurence of 'b' : "+multiset.count("b"));
   }
}
```

## 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.

```
Occurence of 'b' : 5
Total Size : 10
Set [
d
b
c
a
]
MultiSet [
d
b
b
b
b
b
c
c
a
a
]
MultiSet [
Element: d, Occurence(s): 1
Element: b, Occurence(s): 5
Element: c, Occurence(s): 2
Element: a, Occurence(s): 2
]
Occurence of 'b' : 3
```

## Multimap Interface

Multimap interface extends Map so that its keys can be mapped to multiple values at a time.

### Interface Declaration

Following is the declaration for **com.google.common.collect.Multimap<K,V>** interface:

```
@GwtCompatible

public interface Multimap<K,V>
```

## Interface Methods

| S.N. | Method & Description |
|---|---|
| 1 | **Map<K,Collection<V>> asMap()**<br>Returns a view of this multimap as a Map from each distinct key to the nonempty collection of that key's associated values. |
| 2 | **void clear()**<br>Removes all key-value pairs from the multimap, leaving it empty. |
| 3 | **boolean containsEntry(Object key, Object value)**<br>Returns true if this multimap contains at least one key-value pair with the key and the value. |
| 4 | **boolean containsKey(Object key)**<br>Returns true if this multimap contains at least one key-value pair with the key. |
| 5 | **boolean containsValue(Object value)**<br>Returns true if this multimap contains at least one key-value pair with the value. |
| 6 | **Collection<Map.Entry<K,V>> entries()**<br>Returns a view collection of all key-value pairs contained in this multimap, as Map.Entry instances. |
| 7 | **boolean equals(Object obj)**<br>Compares the specified object with this multimap for equality. |
| 8 | **Collection<V> get(K key)**<br>Returns a view collection of the values associated with key in this multimap, if any. |
| 9 | **int hashCode()**<br>Returns the hash code for this multimap. |
| 10 | **boolean isEmpty()**<br>Returns true if this multimap contains no key-value pairs. |
| 11 | **Multiset<K> keys()**<br>Returns a view collection containing the key from each key-value pair in this multimap, without collapsing duplicates. |

| | |
|---|---|
| 12 | **Set<K> keySet()**<br>Returns a view collection of all distinct keys contained in this multimap. |
| 13 | **boolean put(K key, V value)**<br>Stores a key-value pair in this multimap. |
| 14 | **boolean putAll(K key, Iterable<? extends V> values)**<br>Stores a key-value pair in this multimap for each of values, all using the same key, key. |
| 15 | **boolean putAll(Multimap<? extends K,? extends V> multimap)**<br>Stores all key-value pairs of multimap in this multimap, in the order returned by multimap.entries(). |
| 16 | **boolean remove(Object key, Object value)**<br>Removes a single key-value pair with the key and the value from this multimap, if such exists. |
| 17 | **Collection<V> removeAll(Object key)**<br>Removes all values associated with the key. |
| 18 | **Collection<V> replaceValues(K key, Iterable<? extends V> values)**<br>Stores a collection of values with the same key, replacing any existing values for that key. |
| 19 | **int size()**<br>Returns the number of key-value pairs in this multimap. |
| 20 | **Collection<V> values()**<br>Returns a view collection containing the value from each key-value pair contained in this multimap, without collapsing duplicates (so values().size() == size()). |

## Example of Multimap

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

### GuavaTester.java

```
import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.Set;

import com.google.common.collect.ArrayListMultimap;
import com.google.common.collect.Multimap;
```

```
public class GuavaTester {
   public static void main(String args[]){
      GuavaTester tester = new GuavaTester();
      Multimap<String,String> multimap = tester.getMultimap();

      List<String> lowerList = (List<String>)multimap.get("lower");
      System.out.println("Initial lower case list");
      System.out.println(lowerList.toString());
      lowerList.add("f");
      System.out.println("Modified lower case list");
      System.out.println(lowerList.toString());

      List<String> upperList = (List<String>)multimap.get("upper");
      System.out.println("Initial upper case list");
      System.out.println(upperList.toString());
      upperList.remove("D");
      System.out.println("Modified upper case list");
      System.out.println(upperList.toString());

      Map<String, Collection<String>> map = multimap.asMap();
      System.out.println("Multimap as a map");
      for (Map.Entry<String,  Collection<String>> entry : map.entrySet())
      {
         String key = entry.getKey();
         Collection<String> value =  multimap.get("lower");
         System.out.println(key + ":" + value);
      }

      System.out.println("Keys of Multimap");
      Set<String> keys =  multimap.keySet();
      for(String key:keys){
         System.out.println(key);
      }

      System.out.println("Values of Multimap");
      Collection<String> values = multimap.values();
      System.out.println(values);
   }
   private Multimap<String,String> getMultimap(){
      //Map<String, List<String>>
      // lower -> a, b, c, d, e
      // upper -> A, B, C, D

      Multimap<String,String> multimap = ArrayListMultimap.create();

      multimap.put("lower", "a");
      multimap.put("lower", "b");
      multimap.put("lower", "c");
      multimap.put("lower", "d");
      multimap.put("lower", "e");

      multimap.put("upper", "A");
      multimap.put("upper", "B");
      multimap.put("upper", "C");
      multimap.put("upper", "D");
      return multimap;
   }
}
```

## 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.

```
Initial lower case list
[a, b, c, d, e]
Modified lower case list
[a, b, c, d, e, f]
Initial upper case list
[A, B, C, D]
Modified upper case list
```

```
[A, B, C]
Multimap as a map
upper:[a, b, c, d, e, f]
lower:[a, b, c, d, e, f]
Keys of Multimap
upper
lower
Values of Multimap
[A, B, C, a, b, c, d, e, f]
```

## BiMap Interface

A BiMap is a special kind of map which maintains an inverse view of the map while ensuring that no duplicate values are present in the map and a value can be used safely to get the key back.

### Interface Declaration

Following is the declaration for **com.google.common.collect.Bimap<K,V>** interface:

```
@GwtCompatible
public interface BiMap<K,V>
extends Map<K,V>
```

### Interface Methods

| S.N. | Method & Description |
|---|---|
| 1 | **V forcePut(K key, V value)**<br>An alternate form of ‘put’ that silently removes any existing entry with the value before proceeding with the put(K, V) operation. |
| 2 | **BiMap<V,K> inverse()**<br>Returns the inverse view of this bimap, which maps each of this bimap's values to its associated key. |
| 3 | **V put(K key, V value)**<br>Associates the specified value with the specified key in this map (optional operation). |
| 4 | **void putAll(Map<? extends K,? extends V> map)**<br>Copies all of the mappings from the specified map to this map (optional operation). |
| 5 | **Set<V> values()** |

| | |
|---|---|
| | Returns a Collection view of the values contained in this map. |

## Methods Inherited

This class inherits methods from the following interface:

- java.util.Map

## Example of BiMap

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

## GuavaTester.java

```
import com.google.common.collect.BiMap;
import com.google.common.collect.HashBiMap;

public class GuavaTester {

   public static void main(String args[]){
      BiMap<Integer, String> empIDNameMap = HashBiMap.create();

      empIDNameMap.put(new Integer(101), "Mahesh");
      empIDNameMap.put(new Integer(102), "Sohan");
      empIDNameMap.put(new Integer(103), "Ramesh");

      //Emp Id of Employee "Mahesh"
      System.out.println(empIDNameMap.inverse().get("Mahesh"));
   }
}
```

## 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.

```
101
```

## Table Interface

Table represents a special map where two keys can be specified in combined fashion to refer to a single value. It is similar to creating a map of maps.

### Interface Declaration

Following is the declaration for **com.google.common.collect.Table<R,C,V>** interface:

```
@GwtCompatible

public interface Table<R,C,V>
```

### Interface Methods

| S.N. | Method & Description |
|---|---|
| 1 | **Set<Table.Cell<R,C,V>> cellSet()**<br>Returns a set of all row key/column key/value triplets. |
| 2 | **void clear()**<br>Removes all mappings from the table. |
| 3 | **Map<R,V> column(C columnKey)**<br>Returns a view of all mappings that have the given column key. |
| 4 | **Set<C> columnKeySet()**<br>Returns a set of column keys that have one or more values in the table. |
| 5 | **Map<C,Map<R,V>> columnMap()**<br>Returns a view that associates each column key with the corresponding map from row keys to values. |
| 6 | **boolean contains(Object rowKey, Object columnKey)**<br>Returns true if the table contains a mapping with the specified row and column keys. |
| 7 | **boolean containsColumn(Object columnKey)**<br>Returns true if the table contains a mapping with the specified column. |
| 8 | **boolean containsRow(Object rowKey)**<br>Returns true if the table contains a mapping with the specified row key. |
| 9 | **boolean containsValue(Object value)**<br>Returns true if the table contains a mapping with the specified value. |
| 10 | **boolean equals(Object obj)**<br>Compares the specified object with this table for equality. |

| | |
|---|---|
| 11 | **V get(Object rowKey, Object columnKey)**<br>Returns the value corresponding to the given row and column keys, or null if no such mapping exists. |
| 12 | **int hashCode()**<br>Returns the hash code for this table. |
| 13 | **boolean isEmpty()**<br>Returns true if the table contains no mappings. |
| 14 | **V put(R rowKey, C columnKey, V value)**<br>Associates the specified value with the specified keys. |
| 15 | **void putAll(Table<? extends R,? extends C,? extends V> table)**<br>Copies all mappings from the specified table to this table. |
| 16 | **V remove(Object rowKey, Object columnKey)**<br>Removes the mapping, if any, associated with the given keys. |
| 17 | **Map<C,V> row(R rowKey)**<br>Returns a view of all mappings that have the given row key. |
| 18 | **Set<R> rowKeySet()**<br>Returns a set of row keys that have one or more values in the table. |
| 19 | **Map<R,Map<C,V>> rowMap()**<br>Returns a view that associates each row key with the corresponding map from column keys to values. |
| 20 | **int size()**<br>Returns the number of row key / column key / value mappings in the table. |
| 21 | **Collection<V> values()**<br>Returns a collection of all values, which may contain duplicates. |

## Example of Table Interface

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

### GuavaTester.java

```
import java.util.Map;
import java.util.Set;

import com.google.common.collect.HashBasedTable;
import com.google.common.collect.Table;

```

```
public class GuavaTester {

   public static void main(String args[]){
      //Table<R,C,V> == Map<R,Map<C,V>>
      /*
      *  Company: IBM, Microsoft, TCS
      *  IBM             -> {101:Mahesh, 102:Ramesh, 103:Suresh}
      *  Microsoft       -> {101:Sohan, 102:Mohan, 103:Rohan }
      *  TCS             -> {101:Ram, 102: Shyam, 103: Sunil }
      *
      * */
      //create a table
      Table<String, String, String> employeeTable =
       HashBasedTable.create();

      //initialize the table with employee details
      employeeTable.put("IBM", "101","Mahesh");
      employeeTable.put("IBM", "102","Ramesh");
      employeeTable.put("IBM", "103","Suresh");

      employeeTable.put("Microsoft", "111","Sohan");
      employeeTable.put("Microsoft", "112","Mohan");
      employeeTable.put("Microsoft", "113","Rohan");

      employeeTable.put("TCS", "121","Ram");
      employeeTable.put("TCS", "122","Shyam");
      employeeTable.put("TCS", "123","Sunil");

      //get Map corresponding to IBM
      Map<String,String> ibmEmployees =  employeeTable.row("IBM");

      System.out.println("List of IBM Employees");
      for(Map.Entry<String, String> entry : ibmEmployees.entrySet()){
         System.out.println("Emp Id: " + entry.getKey() + ", Name: " +
         entry.getValue());
      }

      //get all the unique keys of the table
      Set<String> employers = employeeTable.rowKeySet();
      System.out.print("Employers: ");
      for(String employer: employers){
         System.out.print(employer + " ");
      }
      System.out.println();

      //get a Map corresponding to 102
      Map<String,String> EmployerMap =  employeeTable.column("102");
      for(Map.Entry<String, String> entry : EmployerMap.entrySet()){
         System.out.println("Employer: " + entry.getKey() + ", Name: " +
         entry.getValue());
      }
   }
}
```

## 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.

```
List of IBM Employees
Emp Id: 102, Name: Ramesh
Emp Id: 101, Name: Mahesh
Emp Id: 103, Name: Suresh
Employers: IBM TCS Microsoft
Employer: IBM, Name: Ramesh
```

# 10. Guava – Caching Utilities

Guava provides a very powerful memory based caching mechanism by an interface LoadingCache<K,V>. Values are automatically loaded in the cache and it provides many utility methods useful for caching needs.

## Interface Declaration

Following is the declaration for **com.google.common.cache.LoadingCache<K,V>** interface:

```
@Beta
@GwtCompatible
public interface LoadingCache<K,V>
   extends Cache<K,V>, Function<K,V>
```

## Interface Methods

| S.N. | Method & Description |
|---|---|
| 1 | **V apply(K key)**<br>Deprecated. Provided to satisfy the Function interface; use get(K) or getUnchecked(K) instead. |
| 2 | **ConcurrentMap<K,V> asMap()**<br>Returns a view of the entries stored in this cache as a thread-safe map. |
| 3 | **V get(K key)**<br>Returns the value associated with key in this cache, first loading that value if necessary. |
| 4 | **ImmutableMap<K,V> getAll(Iterable<? extends K> keys)**<br>Returns a map of the values associated with keys, creating or retrieving those values if necessary. |
| 5 | **V getUnchecked(K key)**<br>Returns the value associated with key in this cache, first loading that value if necessary. |
| 6 | **void refresh(K key)**<br>Loads a new value for key, possibly asynchronously. |

## Example of LoadingCache

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

### GuavaTester.java

```
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;

import com.google.common.base.MoreObjects;
import com.google.common.cache.CacheBuilder;
import com.google.common.cache.CacheLoader;
import com.google.common.cache.LoadingCache;

public class GuavaTester {
   public static void main(String args[]){
      //create a cache for employees based on their employee id
      LoadingCache<string, Employee>employeeCache =
      CacheBuilder.newBuilder()
            .maximumSize(100) // maximum 100 records can be cached
            .expireAfterAccess(30, TimeUnit.MINUTES)
             // cache will expire after 30 minutes of access
            .build(new CacheLoader<String, Employee>(){
             // build the cacheloader
               @Override
               public Employee load(String empId) throws Exception {
                  // make the expensive call
                  return getFromDatabase(empId);
               }
            });

      try {
         //on first invocation, cache will be populated with
         corresponding
         //employee record
         System.out.println("Invocation #1");
         System.out.println(employeeCache.get("100"));
         System.out.println(employeeCache.get("103"));
```

```
         System.out.println(employeeCache.get("110"));
         //second invocation, data will be returned from cache
         System.out.println("Invocation #2");
         System.out.println(employeeCache.get("100"));
         System.out.println(employeeCache.get("103"));
         System.out.println(employeeCache.get("110"));

      } catch (ExecutionException e) {
         e.printStackTrace();
      }
   }

   private static Employee getFromDatabase(String empId){
      Employee e1 = new Employee("Mahesh", "Finance", "100");
      Employee e2 = new Employee("Rohan", "IT", "103");
      Employee e3 = new Employee("Sohan", "Admin", "110");

      Map <String, Employee> database = new HashMap <String, Employee> ();
      database.put("100", e1);
      database.put("103", e2);
      database.put("110", e3);
      System.out.println("Database hit for" + empId);
      return database.get(empId);
   }
}

class Employee {
   String name;
   String dept;
   String emplD;

   public Employee(String name, String dept, String empID){
      this.name = name;
      this.dept = dept;
      this.emplD = empID;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public String getDept() {
      return dept;
   }
   public void setDept(String dept) {
      this.dept = dept;
   }
   public String getEmplD() {
      return emplD;
   }
   public void setEmplD(String emplD) {
      this.emplD = emplD;
   }

   @Override
   public String toString() {
      return MoreObjects.toStringHelper(Employee.class)
      .add("Name", name)
      .add("Department", dept)
      .add("Emp Id", emplD).toString();
   }
}
```

## 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.

```
Invocation #1
Database hit for100
Employee{Name=Mahesh, Department=Finance, Emp Id=100}
Database hit for103
Employee{Name=Rohan, Department=IT, Emp Id=103}
Database hit for110
Employee{Name=Sohan, Department=Admin, Emp Id=110}
Invocation #2
Employee{Name=Mahesh, Department=Finance, Emp Id=100}
Employee{Name=Rohan, Department=IT, Emp Id=103}
Employee{Name=Sohan, Department=Admin, Emp Id=110}
```

# 11. Guava – String Utilities

Guava introduces many advanced string utilities based on developers' experience in application development works. Following is the list of useful string based utilities:

| S.N. | Utility name & Description |
|---|---|
| 1 | **Joiner**<br>Utility to join objects, string etc. |
| 2 | **Splitter**<br>Utility to split string. |
| 3 | **CharMatcher**<br>Utility for character operations. |
| 4 | **CaseFormat**<br>Utility for changing string formats. |

## Joiner Class

Joiner provides various methods to handle joining operations on string, objects, etc.

### Class Declaration

Following is the declaration for **com.google.common.base.Joiner** class:

```
@GwtCompatible
public class Joiner
   extends Object
```

### Class Methods

| S.N. | Method & Description |
|---|---|
| 1 | **<A extends Appendable> A appendTo(A appendable, Iterable<?> parts)**<br>Appends the string representation of each of the parts, using the previously configured separator between each, to appendable. |
| 2 | **<A extends Appendable> A appendTo(A appendable, Iterator<?> parts)**<br>Appends the string representation of each of the parts, using the previously configured separator between each, to appendable. |

| | |
|---|---|
| 3 | **<A extends Appendable> A appendTo(A appendable, Object[] parts)**<br>Appends the string representation of each of the parts, using the previously configured separator between each, to appendable. |
| 4 | **<A extends Appendable> A appendTo(A appendable, Object first, Object second, Object... rest)**<br>Appends to appendable the string representation of each of the remaining arguments. |
| 5 | **StringBuilder appendTo(StringBuilder builder, Iterable<?> parts)**<br>Appends the string representation of each of the parts, using the previously configured separator between each, to builder. |
| 6 | **StringBuilder appendTo(StringBuilder builder, Iterator<?> parts)**<br>Appends the string representation of each of the parts, using the previously configured separator between each, to builder. |
| 7 | **StringBuilder appendTo(StringBuilder builder, Object[] parts)**<br>Appends the string representation of each of the parts, using the previously configured separator between each, to builder. |
| 8 | **StringBuilder appendTo(StringBuilder builder, Object first, Object second, Object... rest)**<br>Appends to builder the string representation of each of the remaining arguments. |
| 9 | **String join(Iterable<?> parts)**<br>Returns a string containing the string representation of each of the parts, using the previously configured separator between each. |
| 10 | **String join(Iterator<?> parts)**<br>Returns a string containing the string representation of each of the parts, using the previously configured separator between each. |
| 11 | **String join(Object[] parts)**<br>Returns a string containing the string representation of each of the parts, using the previously configured separator between each. |
| 12 | **String join(Object first, Object second, Object... rest)**<br>Returns a string containing the string representation of each argument, using the previously configured separator between each. |
| 13 | **static Joiner on(char separator)**<br>Returns a joiner which automatically places separator between consecutive elements. |
| 14 | **static Joiner on(String separator)**<br>Returns a joiner which automatically places separator between consecutive elements. |

| | |
|---|---|
| 15 | **Joiner skipNulls()**<br>Returns a joiner with the same behavior as this joiner, except automatically skipping over any provided null elements. |
| 16 | **Joiner useForNull(String nullText)**<br>Returns a joiner with the same behavior as this one, except automatically substituting nullText for any provided null elements. |
| 17 | **Joiner.MapJoiner withKeyValueSeparator(String keyValueSeparator)**<br>Returns a MapJoiner using the given key-value separator, and the same configuration as this Joiner otherwise. |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of Joiner Class

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

### GuavaTester.java

```
import java.util.Arrays;
import com.google.common.base.Joiner;

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

   private void testJoiner(){
      System.out.println(Joiner.on(",")
         .skipNulls()
         .join(Arrays.asList(1,2,3,4,5,null,6)));
   }
}
```

### 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.

```
1,2,3,4,5,6
```

## Splitter Class

Splitter provides various methods to handle splitting operations on string, objects, etc.

### Class Declaration

Following is the declaration for **com.google.common.base.Splitter** class:

```
@GwtCompatible(emulated=true)

public final class Splitter

   extends Object
```

### Class Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static Splitter fixedLength(int length)**<br>Returns a splitter that divides strings into pieces of the given length. |
| 2 | **Splitter limit(int limit)**<br>Returns a splitter that behaves equivalently to this splitter but stops splitting after it reaches the limit. |
| 3 | **Splitter omitEmptyStrings()**<br>Returns a splitter that behaves equivalently to this splitter, but automatically omits empty strings from the results. |
| 4 | **static Splitter on(char separator)**<br>Returns a splitter that uses the given single-character separator. |
| 5 | **static Splitter on(CharMatcher separatorMatcher)**<br>Returns a splitter that considers any single character matched by the given CharMatcher to be a separator. |

| | |
|---|---|
| 6 | **static Splitter on(Pattern separatorPattern)**<br>Returns a splitter that considers any subsequence matching pattern to be a separator. |
| 7 | **static Splitter on(String separator)**<br>Returns a splitter that uses the given fixed string as a separator. |
| 8 | **static Splitter onPattern(String separatorPattern)**<br>Returns a splitter that considers any subsequence matching a given pattern (regular expression) to be a separator. |
| 9 | **Iterable<String> split(CharSequence sequence)**<br>Splits sequence into string components and makes them available through an Iterator, which may be lazily evaluated. |
| 10 | **List<String> splitToList(CharSequence sequence)**<br>Splits sequence into string components and returns them as an immutable list. |
| 11 | **Splitter trimResults()**<br>Returns a splitter that behaves equivalently to this splitter, but automatically removes leading and trailing whitespace from each returned substring; equivalent to trimResults(CharMatcher.WHITESPACE). |
| 12 | **Splitter trimResults(CharMatcher trimmer)**<br>Returns a splitter that behaves equivalently to this splitter, but removes all leading or trailing characters matching the given CharMatcher from each returned substring. |
| 13 | **Splitter.MapSplitter withKeyValueSeparator(char separator)**<br>Returns a MapSplitter which splits entries based on this splitter, and splits entries into keys and values using the specified separator. |
| 14 | **Splitter.MapSplitter withKeyValueSeparator(Splitter keyValueSplitter)**<br>Returns a MapSplitter which splits entries based on this splitter, and splits entries into keys and values using the specified key-value splitter. |
| 15 | **Splitter.MapSplitter withKeyValueSeparator(String separator)**<br>Returns a MapSplitter which splits entries based on this splitter, and splits entries into keys and values using the specified separator. |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of Splitter Class

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

## GuavaTester.java

```
import com.google.common.base.Splitter;

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

   private void testSplitter(){
      System.out.println(Splitter.on(',')
         .trimResults()
         .omitEmptyStrings()
         .split("the ,quick, , brown         , fox,              jumps,
          over, the, lazy, little dog."));
   }
}
```

### 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.

```
[the, quick, brown, fox, jumps, over, the, lazy, little dog.]
```

## CharMatcher Class

CharMatcher provides various methods to handle various JAVA types for char values.

### Class Declaration

Following is the declaration for **com.google.common.base.CharMatcher** class:

```
@GwtCompatible(emulated=true)
public final class CharMatcher
   extends Object
```

## Fields

| S.N. | Field & Description |
|---|---|
| 1 | **static CharMatcher ANY**<br>Matches any character. |
| 2 | **static CharMatcher ASCII**<br>Determines whether a character is ASCII, meaning that its code point is less than 128. |
| 3 | **static CharMatcher BREAKING_WHITESPACE**<br>Determines whether a character is a breaking whitespace (that is, a whitespace which can be interpreted as a break between words for formatting purposes). |
| 4 | **static CharMatcher DIGIT**<br>Determines whether a character is a digit according to Unicode. |
| 5 | **static CharMatcher INVISIBLE**<br>Determines whether a character is invisible; that is, if its Unicode category is any of SPACE_SEPARATOR, LINE_SEPARATOR, PARAGRAPH_SEPARATOR, CONTROL, FORMAT, SURROGATE, and PRIVATE_USE according to ICU4J. |
| 6 | **static CharMatcher JAVA_DIGIT**<br>Determines whether a character is a digit according to Java's definition. |
| 7 | **static CharMatcher JAVA_ISO_CONTROL**<br>Determines whether a character is an ISO control character as specified by Character.isISOControl(char). |
| 8 | **static CharMatcher JAVA_LETTER**<br>Determines whether a character is a letter according to Java's definition. |
| 9 | **static CharMatcher JAVA_LETTER_OR_DIGIT**<br>Determines whether a character is a letter or digit according to Java's definition. |
| 10 | **static CharMatcher JAVA_LOWER_CASE**<br>Determines whether a character is lower case according to Java's definition. |
| 11 | **static CharMatcher JAVA_UPPER_CASE**<br>Determines whether a character is upper case according to Java's definition. |
| 12 | **static CharMatcher NONE**<br>Matches no characters. |
| 13 | **static CharMatcher SINGLE_WIDTH**<br>Determines whether a character is single-width (not double-width). |

| | |
|---|---|
| 14 | **static CharMatcher WHITESPACE**<br>Determines whether a character is whitespace according to the latest Unicode standard, as illustrated here. |

## Constructor(s)

| S.N. | Constructor & Description |
|---|---|
| 1 | **protected CharMatcher()**<br>Constructor for use by subclasses. |

## Class Methods

| S.N. | Methods & Description |
|---|---|
| 1 | **CharMatcher and(CharMatcher other)**<br>Returns a matcher that matches any character matched by both this matcher and other. |
| 2 | **static CharMatcher anyOf(CharSequence sequence)**<br>Returns a char matcher that matches any character present in the given character sequence. |
| 3 | **boolean apply(Character character)**<br>Deprecated. Provided only to satisfy the Predicate interface; use matches(char) instead. |
| 4 | **String collapseFrom(CharSequence sequence, char replacement)**<br>Returns a string copy of the input character sequence, with each group of consecutive characters that match this matcher replaced by a single replacement character. |
| 5 | **int countIn(CharSequence sequence)**<br>Returns the number of matching characters found in a character sequence. |
| 6 | **static CharMatcher forPredicate(Predicate<? super Character> predicate)**<br>Returns a matcher with identical behavior to the given Character-based predicate, but which operates on primitive char instances instead. |
| 7 | **int indexIn(CharSequence sequence)**<br>Returns the index of the first matching character in a character sequence, or -1 if no matching character is present. |
| 8 | **int indexIn(CharSequence sequence, int start)**<br>Returns the index of the first matching character in a character sequence, |

| | |
|---|---|
| | starting from a given position, or -1 if no character matches after that position. |
| 9 | **static CharMatcher inRange(char startInclusive, char endInclusive)** Returns a char matcher that matches any character in a given range (both endpoints are inclusive). |
| 10 | **static CharMatcher is(char match)** Returns a char matcher that matches only one specified character. |
| 11 | **static CharMatcher isNot(char match)** Returns a char matcher that matches any character except the one specified. |
| 12 | **int lastIndexIn(CharSequence sequence)** Returns the index of the last matching character in a character sequence, or -1 if no matching character is present. |
| 13 | **abstract boolean matches(char c)** Determines a true or false value for the given character. |
| 14 | **boolean matchesAllOf(CharSequence sequence)** Returns true if a character sequence contains only matching characters. |
| 15 | **boolean matchesAnyOf(CharSequence sequence)** Returns true if a character sequence contains at least one matching character. |
| 16 | **boolean matchesNoneOf(CharSequence sequence)** Returns true if a character sequence contains no matching characters. |
| 17 | **CharMatcher negate()** Returns a matcher that matches any character not matched by this matcher. |
| 18 | **static CharMatcher noneOf(CharSequence sequence)** Returns a char matcher that matches any character not present in the given character sequence. |
| 19 | **CharMatcher or(CharMatcher other)** Returns a matcher that matches any character matched by either this matcher or other. |
| 20 | **CharMatcher precomputed()** Returns a char matcher functionally equivalent to this one, but which may be faster to query than the original; your mileage may vary. |
| 21 | **String removeFrom(CharSequence sequence)** Returns a string containing all non-matching characters of a character sequence, in order. |

| | |
|---|---|
| 22 | **String replaceFrom(CharSequence sequence, char replacement)**<br>Returns a string copy of the input character sequence, with each character that matches this matcher replaced by a given replacement character. |
| 23 | **String replaceFrom(CharSequence sequence, CharSequence replacement)**<br>Returns a string copy of the input character sequence, with each character that matches this matcher replaced by a given replacement sequence. |
| 24 | **String retainFrom(CharSequence sequence)**<br>Returns a string containing all matching characters of a character sequence, in order. |
| 25 | **String toString()**<br>Returns a string representation of this CharMatcher, such as CharMatcher.or(WHITESPACE, JAVA_DIGIT). |
| 26 | **String trimAndCollapseFrom(CharSequence sequence, char replacement)**<br>Collapses groups of matching characters exactly as collapseFrom(java.lang.CharSequence, char) does, except that groups of matching characters at the start or end of the sequence are removed without replacement. |
| 27 | **String trimFrom(CharSequence sequence)**<br>Returns a substring of the input character sequence that omits all characters this matcher matches from the beginning and from the end of the string. |
| 28 | **String trimLeadingFrom(CharSequence sequence)**<br>Returns a substring of the input character sequence that omits all characters this matcher matches from the beginning of the string. |
| 29 | **String trimTrailingFrom(CharSequence sequence)**<br>Returns a substring of the input character sequence that omits all characters this matcher matches from the end of the string. |

## Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

## Example of CharMatcher Class

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

### GuavaTester.java

```
import com.google.common.base.CharMatcher;
import com.google.common.base.Splitter;

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

   private void testCharMatcher(){
      System.out.println(CharMatcher.DIGIT.retainFrom("mahesh123")); //
      only the digits
      System.out.println(CharMatcher.WHITESPACE.trimAndCollapseFrom("
      Mahesh     Parashar ", ' '));
      // trim whitespace at ends, and replace/collapse whitespace into
      single spaces
      System.out.println(CharMatcher.JAVA_DIGIT.replaceFrom("mahesh123",
      "*")); // star out all digits

System.out.println(CharMatcher.JAVA_DIGIT.or(CharMatcher.JAVA_LOWER_CASE).retai
nFrom("mahesh123"));
      // eliminate all characters that aren't digits or lowercase
   }
}
```

### 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.

```
123
Mahesh Parashar
mahesh***
mahesh123
```

## CaseFormat Class

CaseFormat is a utility class to provide conversion between various ASCII char formats.

### Class Declaration

Following is the declaration for **com.google.common.base.CaseFormat** class:

```
@GwtCompatible
public enum CaseFormat
   extends Enum<CaseFormat>
```

### Enum Constants

| S.N. | Enum Constant & Description |
|---|---|
| 1 | **LOWER_CAMEL**<br>Java variable naming convention, e.g., "lowerCamel". |
| 2 | **LOWER_HYPHEN**<br>Hyphenated variable naming convention, e.g., "lower-hyphen". |
| 3 | **LOWER_UNDERSCORE**<br>C++ variable naming convention, e.g., "lower_underscore". |
| 4 | **UPPER_CAMEL**<br>Java and C++ class naming convention, e.g., "UpperCamel". |
| 5 | **UPPER_UNDERSCORE**<br>Java and C++ constant naming convention, e.g., "UPPER_UNDERSCORE". |

### Methods

| S.N. | Method & Description |
|---|---|
| 1 | **Converter<String,String> converterTo(CaseFormat targetFormat)**<br>Returns a Converter that converts strings from this format to targetFormat. |
| 2 | **String to(CaseFormat format, String str)** |

| | |
|---|---|
| | Converts the specified String str from this format to the specified format. |
| 3 | **static CaseFormat valueOf(String name)**<br>Returns the enum constant of this type with the specified name. |
| 4 | **static CaseFormat[] values()**<br>Returns an array containing the constants of this enum type, in the order they are declared. |

## Methods Inherited

This class inherits methods from the following classes:

- java.lang.Enum
- java.lang.Object

## Example of CaseFormat Class

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

### GuavaTester.java

```
import com.google.common.base.CaseFormat;

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

   private void testCaseFormat(){
      String data = "test_data";

       System.out.println(CaseFormat.LOWER_HYPHEN.to(CaseFormat
       .LOWER_CAMEL, "test-data"));

       System.out.println(CaseFormat.LOWER_UNDERSCORE.to(CaseFormat
       .LOWER_CAMEL, "test_data"));

       System.out.println(CaseFormat.UPPER_UNDERSCORE.to(CaseFormat
       .UPPER_CAMEL, "test_data"));
   }
}
```

## 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.

```
testData

testData

TestData
```

# 12. Guava – Primitive Utilities

As primitive types of Java cannot be used to pass in generics or in collections as input, Guava provided a lot of Wrapper Utilities classes to handle primitive types as Objects. Following is the list of useful primitive processing utilities:

| S.N. | Utility name & Description |
|---|---|
| 1 | **Bytes**<br>Utility for primitive byte. |
| 2 | **Shorts**<br>Utility for primitive short. |
| 3 | **Ints**<br>Utility for primitive int. |
| 4 | **Longs**<br>Utility for primitive long. |
| 5 | **Floats**<br>Utility for primitive float. |
| 6 | **Doubles**<br>Utility for primitive double. |
| 7 | **Chars**<br>Utility for primitive char. |
| 8 | **Booleans**<br>Utility for primitive boolean. |

## Bytes Class

Bytes is a utility class for primitive type byte.

### Class Declaration

Following is the declaration for **com.google.common.primitives.Bytes** class:

```
@GwtCompatible
public final class Bytes
   extends Object
```

### Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static List<Byte> asList(byte... backingArray)** Returns a fixed-size list backed by the specified array, similar to Arrays.asList(Object[]). |
| 2 | **static byte[] concat(byte[]... arrays)** Returns the values from each provided array combined into a single array. |
| 3 | **static boolean contains(byte[] array, byte target)** Returns true if the target is present as an element anywhere in array. |
| 4 | **static byte[] ensureCapacity(byte[] array, int minLength, int padding)** Returns an array containing the same values as the array, but guaranteed to be of a specified minimum length. |
| 5 | **static int hashCode(byte value)** Returns a hash code for value; equal to the result of invoking ((Byte) value).hashCode(). |
| 6 | **static int indexOf(byte[] array, byte target)** Returns the index of the first appearance of the value target in array. |
| 7 | **static int indexOf(byte[] array, byte[] target)** Returns the start position of the first occurrence of the specified target within the array, or -1 if there is no such occurrence. |
| 8 | **static int lastIndexOf(byte[] array, byte target)** Returns the index of the last appearance of the value target in array. |
| 9 | **static byte[] toArray(Collection<? extends Number> collection)** Returns an array containing each value of collection, converted to a byte value in the manner of Number.byteValue(). |

### Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

### Example of Bytes Class

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

### GuavaTester.java

```
import java.util.List;
import com.google.common.primitives.Bytes;

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

   private void testBytes(){
      byte[] byteArray = {1,2,3,4,5,5,7,9,9};

      //convert array of primitives to array of objects
      List<Byte> objectArray = Bytes.asList(byteArray);
      System.out.println(objectArray.toString());

      //convert array of objects to array of primitives
      byteArray = Bytes.toArray(objectArray);
      System.out.print("[ ");
      for(int i = 0; i< byteArray.length ; i++){
         System.out.print(byteArray[i] + " ");
      }
      System.out.println("]");
      byte data = 5;
      //check if element is present in the list of primitives or not
      System.out.println("5 is in list? "+ Bytes.contains(byteArray,
      data));

      //Returns the index
      System.out.println("Index of 5: " + Bytes.indexOf(byteArray,data));

      //Returns the last index maximum
      System.out.println("Last index of 5: " +
      Bytes.lastIndexOf(byteArray,data));
   }
}
```

### 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.

```
[1, 2, 3, 4, 5, 5, 7, 9, 9]
[ 1 2 3 4 5 5 7 9 9 ]
5 is in list? true
Index of 5: 4
Last index of 5: 5
```

## Shorts Class

Shorts is a utility class for primitive type short.

### Class Declaration

Following is the declaration for **com.google.common.primitives.Shorts** class:

```
@GwtCompatible
public final class Shorts
   extends Object
```

### Fields

| S.N. | Field & Description |
|---|---|
| 1 | **static int BYTES**<br>The number of bytes required to represent a primitive short value. |
| 2 | **static short MAX_POWER_OF_TWO**<br>The largest power of two that can be represented as a short. |

### Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static List<Short> asList(short... backingArray)**<br>Returns a fixed-size list backed by the specified array, similar to |

| | |
|---|---|
| | Arrays.asList(Object[]). |
| 2 | **static short checkedCast(long value)** Returns the short value that is equal to value, if possible. |
| 3 | **static int compare(short a, short b)** Compares the two specified short values. |
| 4 | **static short[] concat(short[]... arrays)** Returns the values from each provided array combined into a single array. |
| 5 | **static boolean contains(short[] array, short target)** Returns true if target is present as an element anywhere in array. |
| 6 | **static short[] ensureCapacity(short[] array, int minLength, int padding)** Returns an array containing the same values as array, but guaranteed to be of a specified minimum length. |
| 7 | **static short fromByteArray(byte[] bytes)** Returns the short value whose big-endian representation is stored in the first 2 bytes of bytes; equivalent to ByteBuffer.wrap(bytes).getShort(). |
| 8 | **static short fromBytes(byte b1, byte b2)** Returns the short value whose byte representation is the given 2 bytes, in big-endian order; equivalent to Shorts.fromByteArray(new byte[] {b1, b2}). |
| 9 | **static int hashCode(short value)** Returns a hash code for value; equal to the result of invoking ((Short) value).hashCode(). |
| 10 | **static int indexOf(short[] array, short target)** Returns the index of the first appearance of the value target in array. |
| 11 | **static int indexOf(short[] array, short[] target)** Returns the start position of the first occurrence of the specified target within array, or -1 if there is no such occurrence. |
| 12 | **static String join(String separator, short... array)** Returns a string containing the supplied short values separated by separator. |
| 13 | **static int lastIndexOf(short[] array, short target)** Returns the index of the last appearance of the value target in array. |
| 14 | **static Comparator<short[]> lexicographicalComparator()** Returns a comparator that compares two short arrays lexicographically. |
| 15 | **static short max(short... array)** |

| | |
|---|---|
| | Returns the greatest value present in array. |
| 16 | **static short min(short... array)**<br>Returns the least value present in array. |
| 17 | **static short saturatedCast(long value)**<br>Returns the short nearest in value to value. |
| 18 | **static Converter<String,Short> stringConverter()**<br>Returns a serializable converter object that converts between strings and shorts using Short.decode(java.lang.String) and Short.toString(). |
| 19 | **static short[] toArray(Collection<? extends Number> collection)**<br>Returns an array containing each value of collection, converted to a short value in the manner of Number.shortValue(). |
| 20 | **static byte[] toByteArray(short value)**<br>Returns a big-endian representation of value in a 2-element byte array; equivalent to ByteBuffer.allocate(2).putShort(value).array(). |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of Shorts Class

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

## GuavaTester.java

```
import java.util.List;


import com.google.common.primitives.Shorts;


public class GuavaTester {


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


   private void testShorts(){
      short[] shortArray = {1,2,3,4,5,6,7,8,9};
      //convert array of primitives to array of objects
      List<Short> objectArray = Shorts.asList(shortArray);
      System.out.println(objectArray.toString());

      //convert array of objects to array of primitives
      shortArray = Shorts.toArray(objectArray);
      System.out.print("[ ");
      for(int i = 0; i< shortArray.length ; i++){
         System.out.print(shortArray[i] + " ");
      }
      System.out.println("]");
      short data = 5;
      //check if element is present in the list of primitives or not
      System.out.println("5 is in list? "+ Shorts.contains(shortArray,
      data));

      //Returns the minimum
      System.out.println("Min: " + Shorts.min(shortArray));

      //Returns the maximum
      System.out.println("Max: " + Shorts.max(shortArray));
      data = 2400;
      //get the byte array from an integer
      byte[] byteArray = Shorts.toByteArray(data);
      for(int i = 0; i< byteArray.length ; i++){
         System.out.print(byteArray[i] + " ");
      }
   }
}
```

## 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.

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[ 1 2 3 4 5 6 7 8 9 ]
5 is in list? true
Min: 1
Max: 9
9 96
```

## Ints Class

Ints is a utility class for primitive type int.

### Class Declaration

Following is the declaration for **com.google.common.primitives.Ints** class:

```
@GwtCompatible
public final class Ints
   extends Object
```

### Fields

| S.N. | Field & Description |
|---|---|
| 1 | **static int BYTES**<br>The number of bytes required to represent a primitive int value. |
| 2 | **static int MAX_POWER_OF_TWO**<br>The largest power of two that can be represented as a int. |

### Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static List<Integer> asList(int... backingArray)**<br>Returns a fixed-size list backed by the specified array, similar to Arrays.asList(Object[]). |
| 2 | **static int checkedCast(long value)**<br>Returns the int value that is equal to value, if possible. |
| 3 | **static int compare(int a, int b)**<br>Compares the two specified int values. |

| | |
|---|---|
| 4 | **static int[] concat(int[]... arrays)**<br>Returns the values from each provided array combined into a single array. |
| 5 | **static boolean contains(int[] array, int target)**<br>Returns true if target is present as an element anywhere in array. |
| 6 | **static int[] ensureCapacity(int[] array, int minLength, int padding)**<br>Returns an array containing the same values as array, but guaranteed to be of a specified minimum length. |
| 7 | **static int fromByteArray(byte[] bytes)**<br>Returns the int value whose big-endian representation is stored in the first 4 bytes of bytes; equivalent to ByteBuffer.wrap(bytes).getInt(). |
| 8 | **static int fromBytes(byte b1, byte b2, byte b3, byte b4)**<br>Returns the int value whose byte representation is the given 4 bytes, in big-endian order; equivalent to Ints.fromByteArray(new byte[] {b1, b2, b3, b4}). |
| 9 | **static int hashCode(int value)**<br>Returns a hash code for value; equal to the result of invoking ((Integer) value).hashCode(). |
| 10 | **static int indexOf(int[] array, int target)**<br>Returns the index of the first appearance of the value target in array. |
| 11 | **static int indexOf(int[] array, int[] target)**<br>Returns the start position of the first occurrence of the specified target within array, or -1 if there is no such occurrence. |
| 12 | **static String join(String separator, int... array)**<br>Returns a string containing the supplied int values separated by separator. |
| 13 | **static int lastIndexOf(int[] array, int target)**<br>Returns the index of the last appearance of the value target in array. |
| 14 | **static Comparator<int[]> lexicographicalComparator()**<br>Returns a comparator that compares two int arrays lexicographically. |
| 15 | **static int max(int... array)**<br>Returns the greatest value present in array. |
| 16 | **static int min(int... array)**<br>Returns the least value present in array. |
| 17 | **static int saturatedCast(long value)**<br>Returns the int nearest in value to value. |

| | |
|---|---|
| 18 | **static Converter<String,Integer> stringConverter()**<br>Returns a serializable converter object that converts between strings and integers using Integer.decode(java.lang.String) and Integer.toString(). |
| 19 | **static int[] toArray(Collection<? extends Number> collection)**<br>Returns an array containing each value of collection, converted to a int value in the manner of Number.intValue(). |
| 20 | **static byte[] toByteArray(int value)**<br>Returns a big-endian representation of value in a 4-element byte array; equivalent to ByteBuffer.allocate(4).putInt(value).array(). |
| 21 | **static Integer tryParse(String string)**<br>Parses the specified string as a signed decimal integer value. |

### Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

### Example of Ints Class

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

### GuavaTester.java

```
import java.util.List;


import com.google.common.primitives.Ints;


public class GuavaTester {


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


   private void testInts(){
      int[] intArray = {1,2,3,4,5,6,7,8,9};


      //convert array of primitives to array of objects
      List<Integer> objectArray = Ints.asList(intArray);
      System.out.println(objectArray.toString());

      //convert array of objects to array of primitives
      intArray = Ints.toArray(objectArray);
      System.out.print("[ ");
      for(int i = 0; i< intArray.length ; i++){
         System.out.print(intArray[i] + " ");
      }
      System.out.println("]");
      //check if element is present in the list of primitives or not
      System.out.println("5 is in list? "+ Ints.contains(intArray, 5));

      //Returns the minimum
      System.out.println("Min: " + Ints.min(intArray));

      //Returns the maximum
      System.out.println("Max: " + Ints.max(intArray));

      //get the byte array from an integer
      byte[] byteArray = Ints.toByteArray(20000);
      for(int i = 0; i< byteArray.length ; i++){
         System.out.print(byteArray[i] + " ");
      }
   }
}
```

## 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.

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[ 1 2 3 4 5 6 7 8 9 ]
5 is in list? true
Min: 1
Max: 9
0 0 78 32
```

## Longs Class

Longs is a utility class for primitive type long.

### Class Declaration

Following is the declaration for **com.google.common.primitives.Longs** class:

```
@GwtCompatible
public final class Longs
   extends Object
```

### Fields

| S.N. | Field & Description |
|---|---|
| 1 | **static int BYTES**<br>The number of bytes required to represent a primitive long value. |
| 2 | **static long MAX_POWER_OF_TWO**<br>The largest power of two that can be represented as a long. |

### Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static List<Long> asList(long... backingArray)**<br>Returns a fixed-size list backed by the specified array, similar to Arrays.asList(Object[]). |
| 2 | **static int compare(long a, long b)**<br>Compares the two specified long values. |
| 3 | **static long[] concat(long[]... arrays)**<br>Returns the values from each provided array combined into a single array. |
| 4 | **static boolean contains(long[] array, long target)** |

| | |
|---|---|
| | Returns true if target is present as an element anywhere in array. |
| 5 | **static long[] ensureCapacity(long[] array, int minLength, int padding)**<br>Returns an array containing the same values as array, but guaranteed to be of a specified minimum length. |
| 6 | **static long fromByteArray(byte[] bytes)**<br>Returns the long value whose big-endian representation is stored in the first 8 bytes of bytes; equivalent to ByteBuffer.wrap(bytes).getLong(). |
| 7 | **static long fromBytes(byte b1, byte b2, byte b3, byte b4, byte b5, byte b6, byte b7, byte b8)**<br>Returns the long value whose byte representation is the given 8 bytes, in big-endian order; equivalent to Longs.fromByteArray(new byte[] {b1, b2, b3, b4, b5, b6, b7, b8}). |
| 8 | **static int hashCode(long value)**<br>Returns a hash code for value; equal to the result of invoking ((Long) value).hashCode(). |
| 9 | **static int indexOf(long[] array, long target)**<br>Returns the index of the first appearance of the value target in array. |
| 10 | **static int indexOf(long[] array, long[] target)**<br>Returns the start position of the first occurrence of the specified target within array, or -1 if there is no such occurrence. |
| 11 | **static String join(String separator, long... array)**<br>Returns a string containing the supplied long values separated by separator. |
| 12 | **static int lastIndexOf(long[] array, long target)**<br>Returns the index of the last appearance of the value target in array. |
| 13 | **static Comparator<long[]> lexicographicalComparator()**<br>Returns a comparator that compares two long arrays lexicographically. |
| 14 | **static long max(long... array)**<br>Returns the greatest value present in array. |
| 15 | **static long min(long... array)**<br>Returns the least value present in array. |
| 16 | **static Converter<String,Long> stringConverter()**<br>Returns a serializable converter object that converts between strings and longs using Long.decode(java.lang.String) and Long.toString(). |
| 17 | **static long[] toArray(Collection<? extends Number> collection)**<br>Returns an array containing each value of collection, converted to a long value in |

| | |
|---|---|
| | the manner of Number.longValue(). |
| 18 | **static byte[] toByteArray(long value)**<br>Returns a big-endian representation of value in an 8-element byte array; equivalent to ByteBuffer.allocate(8).putLong(value).array(). |
| 19 | **static Long tryParse(String string)**<br>Parses the specified string as a signed decimal long value. |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of Longs Class

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

### GuavaTester.java

```
import java.util.List;


import com.google.common.primitives.Ints;
import com.google.common.primitives.Longs;


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


   private void testLongs(){
      long[] longArray = {1,2,3,4,5,6,7,8,9};


      //convert array of primitives to array of objects
      List<Long> objectArray = Longs.asList(longArray);
      System.out.println(objectArray.toString());


      //convert array of objects to array of primitives
      longArray = Longs.toArray(objectArray);
      System.out.print("[ ");
      for(int i = 0; i< longArray.length ; i++){
         System.out.print(longArray[i] + " ");
      }
      System.out.println("]");
      //check if element is present in the list of primitives or not
      System.out.println("5 is in list? "+ Longs.contains(longArray, 5));

      //Returns the minimum
      System.out.println("Min: " + Longs.min(longArray));

      //Returns the maximum
      System.out.println("Max: " + Longs.max(longArray));

      //get the byte array from an integer
      byte[] byteArray = Longs.toByteArray(20000);
      for(int i = 0; i< byteArray.length ; i++){
         System.out.print(byteArray[i] + " ");
      }
   }
}
```

### 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.

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[ 1 2 3 4 5 6 7 8 9 ]
5 is in list? true
Min: 1
Max: 9
0 0 0 0 0 0 78 32
```

## Floats Class

Floats is a utility class for primitive type float.

## Class Declaration

Following is the declaration for **com.google.common.primitives.Floats** class:

```
@GwtCompatible(emulated=true)
   public final class Floats
      extends Object
```

## Fields

| S.N. | Field & Description |
|---|---|
| 1 | **static int BYTES**<br>The number of bytes required to represent a primitive float value. |

## Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static List<Float> asList(float... backingArray)**<br>Returns a fixed-size list backed by the specified array, similar to Arrays.asList(Object[]). |
| 2 | **static int compare(float a, float b)**<br>Compares the two specified float values using Float.compare(float, float). |
| 3 | **static float[] concat(float[]... arrays)**<br>Returns the values from each provided array combined into a single array. |
| 4 | **static boolean contains(float[] array, float target)**<br>Returns true if target is present as an element anywhere in array. |
| 5 | **static float[] ensureCapacity(float[] array, int minLength, int padding)**<br>Returns an array containing the same values as the array, but guaranteed to be of a specified minimum length. |
| 6 | **static int hashCode(float value)**<br>Returns a hash code for value; equal to the result of invoking ((Float) value).hashCode(). |
| 7 | **static int indexOf(float[] array, float target)**<br>Returns the index of the first appearance of the value target in array. |
| 8 | **static int indexOf(float[] array, float[] target)**<br>Returns the start position of the first occurrence of the specified target within array, or -1 if there is no such occurrence. |

| | |
|---|---|
| 9 | **static boolean isFinite(float value)**<br>Returns true if value represents a real number. |
| 10 | **static String join(String separator, float... array)**<br>Returns a string containing the supplied float values, converted to strings as specified by Float.toString(float), and separated by separator. |
| 11 | **static int lastIndexOf(float[] array, float target)**<br>Returns the index of the last appearance of the value target in array. |
| 12 | **static Comparator<float[]> lexicographicalComparator()**<br>Returns a comparator that compares two float arrays lexicographically. |
| 13 | **static float max(float... array)**<br>Returns the greatest value present in array, using the same rules of comparison as Math.min(float, float). |
| 14 | **static float min(float... array)**<br>Returns the least value present in array, using the same rules of comparison as Math.min(float, float). |
| 15 | **static Converter<String,Float> stringConverter()**<br>Returns a serializable converter object that converts between strings and floats using Float.valueOf(java.lang.String) and Float.toString(). |
| 16 | **static float[] toArray(Collection<? extends Number> collection)**<br>Returns an array containing each value of collection, converted to a float value in the manner of Number.floatValue(). |
| 17 | **static Float tryParse(String string)**<br>Parses the specified string as a single-precision floating point value. |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of Floats Class

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

### GuavaTester.java

```
import java.util.List;
import com.google.common.primitives.Floats;

public class GuavaTester {

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

   private void testFloats(){
      float[] floatArray =
      {1.0f,2.0f,3.0f,4.0f,5.0f,6.0f,7.0f,8.0f,9.0f};

      //convert array of primitives to array of objects
      List<Float> objectArray = Floats.asList(floatArray);
      System.out.println(objectArray.toString());

      //convert array of objects to array of primitives
      floatArray = Floats.toArray(objectArray);
      System.out.print("[ ");
      for(int i = 0; i< floatArray.length ; i++){
         System.out.print(floatArray[i] + " ");
      }
      System.out.println("]");
      //check if element is present in the list of primitives or not
      System.out.println("5.0 is in list? "+ Floats.contains(floatArray,
      5.0f));

      //return the index of element
      System.out.println("5.0 position in list "+
      Floats.indexOf(floatArray, 5.0f));

      //Returns the minimum
      System.out.println("Min: " + Floats.min(floatArray));
      //Returns the maximum
      System.out.println("Max: " + Floats.max(floatArray));
   }
}
```

### 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.

```
[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
[ 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 ]
5.0 is in list? true
5.0 position in list 4
Min: 1.0
Max: 9.0
```

## Doubles Class

Doubles is a utility class for primitive type double.

### Class Declaration

Following is the declaration for **com.google.common.primitives.Doubles** class:

```
@GwtCompatible(emulated=true)
   public final class Doubles
      extends Object
```

### Fields

| S.N. | Field & Description |
|---|---|

| | |
|---|---|
| 1 | **static int BYTES**<br>The number of bytes required to represent a primitive double value. |

## Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static List<Double> asList(double... backingArray)**<br>Returns a fixed-size list backed by the specified array, similar to Arrays.asList(Object[]). |
| 2 | **static int compare(double a, double b)**<br>Compares the two specified double values. |
| 3 | **static double[] concat(double[]... arrays)**<br>Returns the values from each provided array combined into a single array. |
| 4 | **static boolean contains(double[] array, double target)**<br>Returns true if target is present as an element anywhere in array. |
| 5 | **static double[] ensureCapacity(double[] array, int minLength, int padding)**<br>Returns an array containing the same values as the array, but guaranteed to be of a specified minimum length. |
| 6 | **static int hashCode(double value)**<br>Returns a hash code for value; equal to the result of invoking ((Double) value).hashCode(). |
| 7 | **static int indexOf(double[] array, double target)**<br>Returns the index of the first appearance of the value target in array. |
| 8 | **static int indexOf(double[] array, double[] target)**<br>Returns the start position of the first occurrence of the specified target within array, or -1 if there is no such occurrence. |
| 9 | **static boolean isFinite(double value)**<br>Returns true if value represents a real number. |
| 10 | **static String join(String separator, double... array)**<br>Returns a string containing the supplied double values, converted to strings as specified by Double.toString(double), and separated by separator. |
| 11 | **static int lastIndexOf(double[] array, double target)**<br>Returns the index of the last appearance of the value target in array. |
| 12 | **static Comparator<double[]> lexicographicalComparator()** |

| | |
|---|---|
| | Returns a comparator that compares two double arrays lexicographically. |
| 13 | **static double max(double... array)**<br>Returns the greatest value present in array, using the same rules of comparison as Math.max(double, double). |
| 14 | **static double min(double... array)**<br>Returns the least value present in array, using the same rules of comparison as Math.min(double, double). |
| 15 | **static Converter<String,Double> stringConverter()**<br>Returns a serializable converter object that converts between strings and doubles using Double.valueOf(java.lang.String) and Double.toString(). |
| 16 | **static double[] toArray(Collection<? extends Number> collection)**<br>Returns an array containing each value of collection, converted to a double value in the manner of Number.doubleValue(). |
| 17 | **static Double tryParse(String string)**<br>Parses the specified string as a double-precision floating point value. |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of Doubles Class

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

### GuavaTester.java

```
import java.util.List;


import com.google.common.primitives.Doubles;


public class GuavaTester {


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


   private void testDoubles(){
      double[] doubleArray = {1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0};

      //convert array of primitives to array of objects
      List<Double> objectArray = Doubles.asList(doubleArray);
      System.out.println(objectArray.toString());

      //convert array of objects to array of primitives
      doubleArray = Doubles.toArray(objectArray);
      System.out.print("[ ");
      for(int i = 0; i< doubleArray.length ; i++){
         System.out.print(doubleArray[i] + " ");
      }
      System.out.println("]");
      //check if element is present in the list of primitives or not
      System.out.println("5.0 is in list? "+
      Doubles.contains(doubleArray, 5.0f));

      //return the index of element
      System.out.println("5.0 position in list "+
      Doubles.indexOf(doubleArray, 5.0f));

      //Returns the minimum
      System.out.println("Min: " + Doubles.min(doubleArray));

      //Returns the maximum
      System.out.println("Max: " + Doubles.max(doubleArray));
   }
}
```

## 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.

```
[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
[ 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 ]
5.0 is in list? true
5.0 position in list 4
Min: 1.0
Max: 9.0
```

## Chars Class

Chars is a utility class for primitive type char.

### Class Declaration

Following is the declaration for **com.google.common.primitives.Chars** class:

```
@GwtCompatible(emulated=true)
   public final class Chars
      extends Object
```

### Fields

| S.N. | Field & Description |
|---|---|
| 1 | **static int BYTES**<br>The number of bytes required to represent a primitive char value. |

### Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static List<Character> asList(char... backingArray)**<br>Returns a fixed-size list backed by the specified array, similar to Arrays.asList(Object[]). |
| 2 | **static char checkedCast(long value)**<br>Returns the char value that is equal to value, if possible. |
| 3 | **static int compare(char a, char b)**<br>Compares the two specified char values. |
| 4 | **static char[] concat(char[]... arrays)**<br>Returns the values from each provided array combined into a single array. |

| | |
|---|---|
| 5 | **static boolean contains(char[] array, char target)**<br>Returns true if target is present as an element anywhere in array. |
| 6 | **static char[] ensureCapacity(char[] array, int minLength, int padding)**<br>Returns an array containing the same values as array, but guaranteed to be of a specified minimum length. |
| 7 | **static char fromByteArray(byte[] bytes)**<br>Returns the char value whose big-endian representation is stored in the first 2 bytes of bytes; equivalent to ByteBuffer.wrap(bytes).getChar(). |
| 8 | **static char fromBytes(byte b1, byte b2)**<br>Returns the char value whose byte representation is the given 2 bytes, in big-endian order; equivalent to Chars.fromByteArray(new byte[] {b1, b2}). |
| 9 | **static int hashCode(char value)**<br>Returns a hash code for value; equal to the result of invoking ((Character) value).hashCode(). |
| 10 | **static int indexOf(char[] array, char target)**<br>Returns the index of the first appearance of the value target in array. |
| 11 | **static int indexOf(char[] array, char[] target)**<br>Returns the start position of the first occurrence of the specified target within array, or -1 if there is no such occurrence. |
| 12 | **static String join(String separator, char... array)**<br>Returns a string containing the supplied char values separated by separator. |
| 13 | **static int lastIndexOf(char[] array, char target)**<br>Returns the index of the last appearance of the value target in array. |
| 14 | **static Comparator<char[]> lexicographicalComparator()**<br>Returns a comparator that compares two char arrays lexicographically. |
| 15 | **static char max(char... array)**<br>Returns the greatest value present in array. |
| 16 | **static char min(char... array)**<br>Returns the least value present in array. |
| 17 | **static char saturatedCast(long value)**<br>Returns the char nearest in value to value. |
| 18 | **static char[] toArray(Collection<Character> collection)**<br>Copies a collection of Character instances into a new array of primitive char values. |

| | |
|---|---|
| 19 | **static byte[] toByteArray(char value)** Returns a big-endian representation of value in a 2-element byte array; equivalent to ByteBuffer.allocate(2).putChar(value).array(). |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of Chars Class

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

### GuavaTester.java

```
import java.util.List;
import com.google.common.primitives.Chars;

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

   private void testChars(){
      char[] charArray = {'a','b','c','d','e','f','g','h'};

      //convert array of primitives to array of objects
      List<Character> objectArray = Chars.asList(charArray);
      System.out.println(objectArray.toString());

      //convert array of objects to array of primitives
      charArray = Chars.toArray(objectArray);
      System.out.print("[ ");
      for(int i = 0; i< charArray.length ; i++){
         System.out.print(charArray[i] + " ");
      }
      System.out.println("]");
      //check if element is present in the list of primitives or not
      System.out.println("c is in list? "+ Chars.contains(charArray, 'c'));
      //return the index of element
      System.out.println("c position in list "+ Chars.indexOf(charArray, 'c'));

      //Returns the minimum
      System.out.println("Min: " + Chars.min(charArray));

      //Returns the maximum
      System.out.println("Max: " + Chars.max(charArray));
   }
}
```

### 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.

```
[a, b, c, d, e, f, g, h]
[ a b c d e f g h ]
c is in list? true
c position in list 2
Min: a
Max: h
```

## Booleans Class

Booleans is a utility class for primitive type Boolean.

### Class Declaration

Following is the declaration for **com.google.common.primitives.Booleans** class:

```
@GwtCompatible(emulated=true)
   public final class Booleans
      extends Object
```

## Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static List<Boolean> asList(boolean... backingArray)**<br>Returns a fixed-size list backed by the specified array, similar to Arrays.asList(Object[]). |
| 2 | **static int compare(boolean a, boolean b)**<br>Compares the two specified boolean values in the standard way (false is considered less than true). |
| 3 | **static boolean[] concat(boolean[]... arrays)**<br>Returns the values from each provided array combined into a single array. |
| 4 | **static boolean contains(boolean[] array, boolean target)**<br>Returns true if target is present as an element anywhere in array. |
| 5 | **static int countTrue(boolean... values)**<br>Returns the number of values that are true. |
| 6 | **static boolean[] ensureCapacity(boolean[] array, int minLength, int padding)**<br>Returns an array containing the same values as array, but guaranteed to be of a specified minimum length. |
| 7 | **static int hashCode(boolean value)**<br>Returns a hash code for value; equal to the result of invoking ((Boolean) value).hashCode(). |
| 8 | **static int indexOf(boolean[] array, boolean target)**<br>Returns the index of the first appearance of the value target in array. |
| 9 | **static int indexOf(boolean[] array, boolean[] target)**<br>Returns the start position of the first occurrence of the specified target within array, or -1 if there is no such occurrence. |
| 10 | **static String join(String separator, boolean... array)**<br>Returns a string containing the supplied boolean values separated by separator. |
| 11 | **static int lastIndexOf(boolean[] array, boolean target)**<br>Returns the index of the last appearance of the value target in array. |

| | |
|---|---|
| 12 | **static Comparator<boolean[]> lexicographicalComparator()** Returns a comparator that compares two boolean arrays lexicographically. |
| 13 | **static boolean[] toArray(Collection<Boolean> collection)** Copies a collection of Boolean instances into a new array of primitive boolean values. |

### Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

### Example of Booleans Class

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

### GuavaTester.java

```
import java.util.List;
import com.google.common.primitives.Booleans;

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

   private void testBooleans(){
      boolean[] booleanArray = {true,true,false,true,true,false,false};

      //convert array of primitives to array of objects
      List<Boolean> objectArray = Booleans.asList(booleanArray);
      System.out.println(objectArray.toString());

      //convert array of objects to array of primitives
      booleanArray = Booleans.toArray(objectArray);
      System.out.print("[ ");
      for(int i = 0; i< booleanArray.length ; i++){
         System.out.print(booleanArray[i] + " ");
      }
      System.out.println("]");
      //check if element is present in the list of primitives or not
      System.out.println("true is in list? "+
      Booleans.contains(booleanArray, true));

      //return the first index of element
      System.out.println("true position in list "+
      Booleans.indexOf(booleanArray, true));

      //Returns the count of true values
      System.out.println("true occured: " + Booleans.countTrue());

      //Returns the comparisons
      System.out.println("false Vs true: " + Booleans.compare(false,
      true));
      System.out.println("false Vs false: " + Booleans.compare(false,
      false));
      System.out.println("true Vs false: " + Booleans.compare(true,
      false));
      System.out.println("true Vs true: " + Booleans.compare(true,
      true));
   }
}
```

## 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.

```
[true, true, false, true, true, false, false]
[ true true false true true false false ]
true is in list? true
true position in list 0
true occured: 0
false Vs true: -1
false Vs false: 0
true Vs false: 1
true Vs true: 0
```

# 13. Guava – Math Utilities

Guava provides Mathematics related Utilities classes to handle int, long and BigInteger. Following is the list of useful utilities:

| S.N. | Utility name & Description |
|---|---|
| 1 | **IntMath**<br>Math utility for int. |
| 2 | **LongMath**<br>Math utility for long. |
| 3 | **BigIntegerMath**<br>Math utility for BigInteger. |

## IntMath Class

IntMath provides utility methods on int.

### Class Declaration

Following is the declaration for **com.google.common.math.IntMath** class:

```
@GwtCompatible(emulated=true)
public final class IntMath
   extends Object
```

### Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static int binomial(int n, int k)**<br>Returns n choose k, also known as the binomial coefficient of n and k, or Integer.MAX_VALUE if the result does not fit in an int. |
| 2 | **static int checkedAdd(int a, int b)**<br>Returns the sum of a and b, provided it does not overflow. |
| 3 | **static int checkedMultiply(int a, int b)**<br>Returns the product of a and b, provided it does not overflow. |

| | |
|---|---|
| 4 | **static int checkedPow(int b, int k)**<br>Returns the b to the kth power, provided it does not overflow. |
| 5 | **static int checkedSubtract(int a, int b)**<br>Returns the difference of a and b, provided it does not overflow. |
| 6 | **static int divide(int p, int q, RoundingMode mode)**<br>Returns the result of dividing p by q, rounding using the specified RoundingMode. |
| 7 | **static int factorial(int n)**<br>Returns n!, that is, the product of the first n positive integers, 1 if n == 0, or Integer.MAX_VALUE if the result does not fit in a int. |
| 8 | **static int gcd(int a, int b)**<br>Returns the greatest common divisor of a, b. |
| 9 | **static boolean isPowerOfTwo(int x)**<br>Returns true if x represents a power of two. |
| 10 | **static int log10(int x, RoundingMode mode)**<br>Returns the base-10 logarithm of x, rounded according to the specified rounding mode. |
| 11 | **static int log2(int x, RoundingMode mode)**<br>Returns the base-2 logarithm of x, rounded according to the specified rounding mode. |
| 12 | **static int mean(int x, int y)**<br>Returns the arithmetic mean of x and y, rounded towards negative infinity. |
| 13 | **static int mod(int x, int m)**<br>Returns x mod m, a non-negative value less than m. |
| 14 | **static int pow(int b, int k)**<br>Returns b to the kth power. |
| 15 | **static int sqrt(int x, RoundingMode mode)**<br>Returns the square root of x, rounded with the specified rounding mode. |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of IntMath Class

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

### GuavaTester.java

```
import java.math.RoundingMode;
import com.google.common.math.IntMath;

public class GuavaTester {

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

   private void testIntMath(){
      try{
         System.out.println(IntMath.checkedAdd(Integer.MAX_VALUE,
         Integer.MAX_VALUE));
      }catch(ArithmeticException e){
         System.out.println("Error: " + e.getMessage());
      }

      System.out.println(IntMath.divide(100, 5,
      RoundingMode.UNNECESSARY));
      try{
         //exception will be thrown as 100 is not completely divisible by
         3 thus rounding
         // is required, and RoundingMode is set as UNNESSARY
         System.out.println(IntMath.divide(100, 3,
         RoundingMode.UNNECESSARY));
      }catch(ArithmeticException e){
         System.out.println("Error: " + e.getMessage());
      }

      System.out.println("Log2(2): "+IntMath.log2(2,
      RoundingMode.HALF_EVEN));

      System.out.println("Log10(10): "+IntMath.log10(10,
      RoundingMode.HALF_EVEN));
      System.out.println("sqrt(100): "+IntMath.sqrt(IntMath.pow(10,2),
      RoundingMode.HALF_EVEN));

      System.out.println("gcd(100,50): "+IntMath.gcd(100,50));

      System.out.println("modulus(100,50): "+IntMath.mod(100,50));

      System.out.println("factorial(5): "+IntMath.factorial(5));
   }
}
```

### 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.

```
Error: overflow
20
Error: mode was UNNECESSARY, but rounding was necessary
Log2(2): 1
Log10(10): 1
sqrt(100): 10
gcd(100,50): 50
modulus(100,50): 0
factorial(5): 120
```

## LongMath Class

LongMath provides utility methods on long.

## Class Declaration

Following is the declaration for **com.google.common.math.LongMath** class:

```
@GwtCompatible(emulated=true)
public final class LongMath
   extends Object
```

## Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static long binomial(int n, int k)**<br>Returns n choose k, also known as the binomial coefficient of n and k, or Long.MAX_VALUE if the result does not fit in a long. |
| 2 | **static long checkedAdd(long a, long b)**<br>Returns the sum of a and b, provided it does not overflow. |
| 3 | **static long checkedMultiply(long a, long b)**<br>Returns the product of a and b, provided it does not overflow. |
| 4 | **static long checkedPow(long b, int k)**<br>Returns the b to the kth power, provided it does not overflow. |
| 5 | **static long checkedSubtract(long a, long b)**<br>Returns the difference of a and b, provided it does not overflow. |
| 6 | **static long divide(long p, long q, RoundingMode mode)**<br>Returns the result of dividing p by q, rounding using the specified RoundingMode. |
| 7 | **static long factorial(int n)**<br>Returns n!, that is, the product of the first n positive integers, 1 if n == 0, or Long.MAX_VALUE if the result does not fit in a long. |
| 8 | **static long gcd(long a, long b)**<br>Returns the greatest common divisor of a, b. |
| 9 | **static boolean isPowerOfTwo(long x)**<br>Returns true if x represents a power of two. |
| 10 | **static int log10(long x, RoundingMode mode)**<br>Returns the base-10 logarithm of x, rounded according to the specified rounding mode. |
| 11 | **static int log2(long x, RoundingMode mode)**<br>Returns the base-2 logarithm of x, rounded according to the specified rounding |

| | |
|---|---|
| | mode. |
| 12 | **static long mean(long x, long y)**<br>Returns the arithmetic mean of x and y, rounded toward negative infinity. |
| 13 | **static int mod(long x, int m)**<br>Returns x mod m, a non-negative value less than m. |
| 14 | **static long mod(long x, long m)**<br>Returns x mod m, a non-negative value less than m. |
| 15 | **static long pow(long b, int k)**<br>Returns b to the kth power. |
| 16 | **static long sqrt(long x, RoundingMode mode)**<br>Returns the square root of x, rounded with the specified rounding mode. |

### Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

### Example of LongMath Class

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

### GuavaTester.java

```
import java.math.RoundingMode;
import com.google.common.math.LongMath;

public class GuavaTester {

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

   private void testLongMath(){
      try{
         System.out.println(LongMath.checkedAdd(Long.MAX_VALUE,
          Long.MAX_VALUE));
      }catch(ArithmeticException e){
         System.out.println("Error: " + e.getMessage());
      }

      System.out.println(LongMath.divide(100, 5,
      RoundingMode.UNNECESSARY));
      try{
         //exception will be thrown as 100 is not completely divisible by
          // 3, thus rounding is required, and
          // RoundingMode is set as UNNESSARY
         System.out.println(LongMath.divide(100, 3, RoundingMode.UNNECESSARY));
      }catch(ArithmeticException e){
         System.out.println("Error: " + e.getMessage());
      }

      System.out.println("Log2(2): "+LongMath.log2(2, RoundingMode.HALF_EVEN));

      System.out.println("Log10(10): "+LongMath.log10(10,
      RoundingMode.HALF_EVEN));

      System.out.println("sqrt(100): "+LongMath.sqrt(LongMath.pow(10,2),
      RoundingMode.HALF_EVEN));

      System.out.println("gcd(100,50): "+LongMath.gcd(100,50));

      System.out.println("modulus(100,50): "+LongMath.mod(100,50));

      System.out.println("factorial(5): "+LongMath.factorial(5));
   }
}
```

## 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.

```
Error: overflow
20
Error: mode was UNNECESSARY, but rounding was necessary
Log2(2): 1
Log10(10): 1
sqrt(100): 10
gcd(100,50): 50
modulus(100,50): 0
factorial(5): 120
```

## BigIntegerMath Class

BigIntegerMath provides utility methods on BigInteger.

### Class Declaration

Following is the declaration for **com.google.common.math.BigIntegerMath** class:

```
@GwtCompatible(emulated=true)
public final class BigIntegerMath
   extends Object
```

### Methods

| S.N. | Method & Description |
|---|---|
| 1 | **static BigInteger binomial(int n, int k)**<br>Returns n choose k, also known as the binomial coefficient of n and k, that is, n! / (k! (n - k)!). |
| 2 | **static BigInteger divide(BigInteger p, BigInteger q, RoundingMode mode)**<br>Returns the result of dividing p by q, rounding using the specified RoundingMode. |
| 3 | **static BigInteger factorial(int n)**<br>Returns n!, that is, the product of the first n positive integers, or 1 if n == 0. |
| 4 | **static boolean isPowerOfTwo(BigInteger x)**<br>Returns true if x represents a power of two. |
| 5 | **static int log10(BigInteger x, RoundingMode mode)**<br>Returns the base-10 logarithm of x, rounded according to the specified rounding mode. |

| | |
|---|---|
| 6 | **static int log2(BigInteger x, RoundingMode mode)** Returns the base-2 logarithm of x, rounded according to the specified rounding mode. |
| 7 | **static BigInteger sqrt(BigInteger x, RoundingMode mode)** Returns the square root of x, rounded with the specified rounding mode. |

## Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

## Example of BigIntegerMath Class

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

## GuavaTester.java

```
import java.math.BigInteger;
import java.math.RoundingMode;

import com.google.common.math.BigIntegerMath;

public class GuavaTester {

   public static void main(String args[]){
      GuavaTester tester = new GuavaTester();
      tester.testBigIntegerMath();
   }
   private void testBigIntegerMath(){
      System.out.println(BigIntegerMath.divide(BigInteger.TEN, new
      BigInteger("2"), RoundingMode.UNNECESSARY));
      try{
         //exception will be thrown as 100 is not completely divisible by
          // 3, thus rounding is required, and
          // RoundingMode is set as UNNESSARY
         System.out.println(BigIntegerMath.divide(BigInteger.TEN, new
         BigInteger("3"), RoundingMode.UNNECESSARY));
      }catch(ArithmeticException e){
         System.out.println("Error: " + e.getMessage());
      }
      System.out.println("Log2(2): "+BigIntegerMath.log2(new
      BigInteger("2"), RoundingMode.HALF_EVEN));

      System.out.println("Log10(10):
      "+BigIntegerMath.log10(BigInteger.TEN, RoundingMode.HALF_EVEN));

      System.out.println("sqrt(100):
      "+BigIntegerMath.sqrt(BigInteger.TEN.multiply(BigInteger.TEN),
      RoundingMode.HALF_EVEN));

      System.out.println("factorial(5): "+BigIntegerMath.factorial(5));
   }
}
```

### 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.

```
5
Error: Rounding necessary
Log2(2): 1
Log10(10): 1
sqrt(100): 10
factorial(5): 120
```