GUAVA
java collections framework

tutorialspoint
SIMPLYEASYLEARNING

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

Table of Contents

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()** Returns an Optional instance with no contained reference.
2	**abstract Set<T> asSet()** 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)** 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)** If nullableReference is non-null, returns an Optional instance containing that reference; otherwise returns absent().
5	**abstract T get()** Returns the contained instance, which must be present.

6	**abstract int hashCode()** Returns a hash code for this instance.
7	**abstract boolean isPresent()** 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)** Returns this Optional if it has a value present; secondChoice otherwise.
10	**abstract T or(Supplier<? extends T> supplier)** Returns the contained instance if it is present; supplier.get() otherwise.
11	**abstract T or(T defaultValue)** Returns the contained instance if it is present; defaultValue otherwise.
12	**abstract T orNull()** Returns the contained instance if it is present; null otherwise.
13	**static <T> Iterable<T> presentInstances(Iterable<? extends Optional<? extends T>> optionals)** Returns the value of each present instance from the supplied optionals, in order, skipping over occurrences of absent().
14	**abstract String toString()** Returns a string representation for this instance.
15	**abstract <V> Optional<V> transform(Function<? super T,V> function)** 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)** Ensures the truth of an expression involving one or more parameters to the calling method.
2	**static void checkArgument(boolean expression, Object errorMessage)** 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)** Ensures the truth of an expression involving one or more parameters to the calling method.
4	**static int checkElementIndex(int index, int size)** 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)** Ensures that index specifies a valid element in an array, list, or a string of size.

6	**static <T> T checkNotNull(T reference)** Ensures that an object reference passed as a parameter to the calling method is not null.
7	**static <T> T checkNotNull(T reference, Object errorMessage)** 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)** Ensures that an object reference passed as a parameter to the calling method is not null.
9	**static int checkPositionIndex(int index, int size)** 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)** 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)** 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)** 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)** 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)** 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()** 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()** 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)** Searches sortedList for key using the binary search algorithm.
4	**abstract int compare(T left, T right)** Compares its two arguments for order.
5	**<U extends T> Ordering<U> compound(Comparator<? super U> secondaryComparator)** 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)** 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)** 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)** 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)** Returns an ordering based on an existing comparator instance.
10	**<E extends T> List<E> greatestOf(Iterable<E> iterable, int k)** 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)** 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)** Returns an immutable list containing elements sorted by this ordering.
13	**boolean isOrdered(Iterable<? extends T> iterable)** 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)** 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)** 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()** 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)** 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)** Returns the greatest of the specified values according to this ordering.
20	**<E extends T> E max(Iterable<E> iterable)** Returns the greatest of the specified values according to this ordering.
21	**<E extends T> E max(Iterator<E> iterator)** Returns the greatest of the specified values according to this ordering.
22	**<E extends T> E min(E a, E b)** 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)** Returns the least of the specified values according to this ordering.
24	**<E extends T> E min(Iterable<E> iterable)** Returns the least of the specified values according to this ordering.
25	**<E extends T> E min(Iterator<E> iterator)** Returns the least of the specified values according to this ordering.
26	**static <C extends Comparable> Ordering<C> natural()** Returns a serializable ordering that uses the natural order of the values.
27	**<S extends T> Ordering<S> nullsFirst()** 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()** 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)** 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()** Returns the reverse of this ordering; the Ordering equivalent to Collections.reverseOrder(Comparator).
31	**<E extends T> List<E> sortedCopy(Iterable<E> elements)** 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()** 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() {
```