

Talleres UTP 2024

Google Guava

Mg. Díaz Sánchez Fernando





OBJETIVOS DEL TALLER

Familiarizarse con las principales funcionalidades de la librería Google Guava



Contenido

1. String Utilities
2. Exception Utilities
3. Converters
4. Collections
5. Resources



Antes de empezar

1. Maven

```
<dependency>  
  <groupId>com.google.guava</groupId>  
  <artifactId>guava</artifactId>  
  <version>33.3.0-jre</version>  
</dependency>
```

¿Qué es Google Guava?

Guava is a set of **core Java libraries** from Google that widely used by them and many other companies as well.

<https://github.com/google/guava>

¿Qué es Google Guava?

Fecha de lanzamiento: **15 Set 2009**

Ultima versión: **33.3.0 (Ago 2024)**

Principales paquetes

base

collect

graph

hash

io


util.concurrent

String Utilities | Strings Class

- Brinda algunas utilidades de texto de fácil uso y de propósito general
- Se encuentran dentro del paquete base
- Algunos métodos: repeat, padStart, padEnd, commonPrefix, isNullOrEmpty
- Destaca: **lenientFormat**

```
// Sin LenientFormat
String nombre = "Fernando";
int edad = 19;
float saldo = 500.0f;
String msg = "Nombre: %s, Edad: %d, Saldo: %f%n";
System.out.printf(msg,
    nombre,
    edad,
    saldo);
```

```
// Con LenientFormat
String nombre_lf = "Fernando";
int edad_lf = 19;
float saldo_lf = 500.0f;
String msg_lf = "Nombre: %s, Edad: %s, Saldo: %s%n";
System.out.printf(msg_lf,
    nombre_lf,
    edad_lf,
    saldo_lf);
```



String Utilities | Joiner

- Junta cadenas u objetos incluyendo un texto separador

```
Joiner joiner = Joiner.on( separator: ";").skipNulls();  
System.out.println(joiner.join( first: "Juan", second: null, ...rest: "Ana", "Rosa"));
```



```
System.out.println(joiner.join( first: "Gaby", second: 19, ...rest: 500.5f, true));
```



String Utilities | Splitter

- Divide cadenas según criterios comunes o personalizados

```
String ciudades = "Cix,Lima,Aqp";  
Iterable<String> splitCity = Splitter.on( separator: "," ).split(ciudades);
```



```
String dataCity = ",Cix,, Lima ,Aqp,,,,,Iqt,";  
Iterable<String> newSplit = Splitter.on( separator: "," ).trimResults().omitEmptyStrings()  
    .split(dataCity);
```



- Realiza operaciones sobre coincidencias de cadenas

```
String data = "U21432454";  
String solo_digitos = CharMatcher.digit().retainFrom(data);
```

U21432454



21432454

```
String codigo = "U21432454";  
String letra = CharMatcher.digit().removeFrom(codigo);
```

U21432454



U

- Métodos que facilitan la revisión de valores de parámetros
- Si no se cumplen las “precondiciones”, se genera una excepción

```
public static String getName(String[] names, int idx){  
    checkNotNull(names, errorMessage: "names cannot be null");  
    checkArgument(expression: idx >= 0, errorMessageTemplate: "idx %s cannot be negative",idx);  
    checkPositionIndex(idx,names.length, desc: "invalid index value");  
    return names[idx];  
}
```

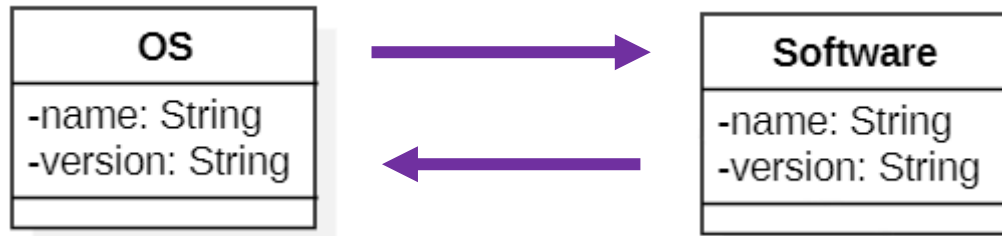


Exceptions Utils | Throwable

- Métodos que facilitan el control de generación de excepciones
- Permite propagar según los tipos de instancias de excepciones

```
public static void printAll(String value) throws InvalidUTPCode {  
    try {  
        System.out.println(Strings.repeat( string: "#", count: 30));  
        printUTPCode(value);  
        System.out.println(Strings.repeat( string: "#", count: 30));  
    } catch (InvalidUTPCode | IllegalArgumentException e) {  
        throwIfInstanceOf(e, InvalidUTPCode.class);  
        throwIfUnchecked(e);  
        throw new AssertionError(e);  
    }  
}
```

- Permite convertir objetos que se encuentran muy relacionados
- Facilita la conversión en reversa y de múltiples instancias



Clases relacionadas

```
OS os1 = new OS( name: "Linux", version: "6.0");
OS os2 = new OS( name: "FreeBSD", version: "14");
OS os3 = new OS( name: "Windows", version: "11");
OS os4 = new OS( name: "MacOS", version: "12");

List<OS> lista = new ArrayList<>(List.of(os1,os2,os3,os4));

Converter<OS, Software> converter = Converter
    .from(Software::valueOf, OS::valueOf);

// Convertir instancias OS a Software
Iterable<Software> softwares = converter.convertAll(lista);
softwares.forEach(System.out::println);

// Conversión inversa de Software a OS
Iterable<OS> os_list = converter.reverse().convertAll(softwares);
os_list.forEach(System.out::println);
```

- Colecciones cuyo contenido no podrá ser modificado
- Su uso es muy recomendado como técnica defensiva
- Garantiza:
 - **Shallow Immutability** (No add, remove, replace)
 - **Null-hostility** (No null)
 - **Thread safety** (Acceso seguro en varias threads)
 - **Integrity** (No se permiten subclases)



Classes Immutables

Interface	JDK or Guava?	Immutable Version
Collection	JDK	ImmutableCollection
List	JDK	ImmutableList
Set	JDK	ImmutableSet
SortedSet/NavigableSet	JDK	ImmutableSortedSet
Map	JDK	ImmutableMap
SortedMap	JDK	ImmutableSortedMap
Multiset	Guava	ImmutableMultiset
SortedMultiset	Guava	ImmutableSortedMultiset
Multimap	Guava	ImmutableMultimap
ListMultimap	Guava	ImmutableListMultimap
SetMultimap	Guava	ImmutableSetMultimap
BiMap	Guava	ImmutableBiMap
ClassToInstanceMap	Guava	ImmutableClassToInstanceMap
Table	Guava	ImmutableTable

Source: [Google Guava](#)

Imagen extraída de: <https://main-cdn.sbermegamarket.ru/big1/hlr-system/479/461/601/111/011/4/600004877140b0.jpeg>



- El método **copyOf** permite realizar una copia defensiva de una colección

```
Software s1 = new Software( name: "Microsoft Office", version: "365");
Software s2 = new Software( name: "HeidiSQL", version: "12");
Software s3 = new Software( name: "PuTTY", version: "1.7");
Software s4 = new Software( name: "MariaDB", version: "10.5");
List<Software> lista = new ArrayList<>(List.of(s1,s2,s3,s4));
lista.add(s1);
```

Permitido

```
ImmutableSet<Software> listain = ImmutableSet.copyOf(lista);
listain.add(s1);
```

Error

- El método **of** permite realizar una copia defensiva de un solo objeto o varios

Principales Interfaces



Multiset



Multimap



Bimap

	C1	C2	C3	C4
Row 1	V1	V2	V3	V5
Row 2	V4	V6	V8	V7
Row 3	V20	V14	V6	V9
Row 4	V10	V12	V33	V11
Row 5	V54	V26	V35	V48

Table

- Funciona de manera similar a un Set pero permite duplicados
- Los elementos en un Multiset que son iguales a otros son referidos como **ocurrencias** del mismo elemento
- El número total de ocurrencias de un elemento se denomina **count**
- Un elemento puede tener un máximo de `Integer.MAX_VALUE` ocurrencias



Implementación Multiset

ImmutableMultiset

ImmutableSortedMultiset

HashMultiset

LinkedHashMultiset

TreeMultiset

ConcurrentHashMultiset

Creación: `Impl.create()`

Ejemplo: `HashMultiset.create()`

Principales Métodos

Multiset<E>	
size()	int
isEmpty()	boolean
count(Object element)	int
containsAll(Collection<?> items)	boolean
contains(Object element)	boolean
add(E element)	boolean
remove(Object element)	boolean
elementSet()	Set<E>
setCount(E element, int count)	int
clear()	void

¿Qué devuelve/hace el método?

- > Devuelve la suma de ocurrencias de todos los elementos
- > True si está vacío y False si no lo está
- > Devuelve la cantidad de veces que aparece *element*
- > True si la colección *items* se encuentra en el Multiset
- > True si *element* se encuentra en el Multiset
- > Agrega un nuevo elemento
- > Elimina una ocurrencia de *element* en el Multiset
- > Devuelve el Set de elementos del Multiset
- > Reconfigura el conteo de *element* en el Multiset
- > Elimina todos los elementos del Multiset

Collections | Multimap

- Funciona de manera similar a un Map pero permite mapear múltiples valores
- Cada **key** puede estar asociada a múltiples **values**
- El contenido puede ser imaginado como key-**collection** o como key-value

key-collection

keyA → val1, val2

keyB → val3

key-value

keyA → val1

keyA → val2

keyB → val3



Implementación MultiMap

ArrayListMultimap

ForwardingListMultimap

ForwardingMultimap

ForwardingSetMultimap

ForwardingSortedSetMultimap

HashMultimap

ImmutableListMultimap

ImmutableMultimap

ImmutableSetMultimap

LinkedHashMultimap

LinkedListMultimap

TreeMultimap

Creación: `Impl.create()`

Ejemplo: `HashMultimap.create()`

Principales Métodos

Multimap<K, V>	
size()	int
isEmpty()	boolean
containsKey(Object key)	boolean
containsValue(Object value)	boolean
containsEntry(Object key, Object val)	boolean
put(K key, V value)	boolean
get(K key)	Collection<V>
remove(Object key, Object val)	boolean
keySet()	Set<K>
clear()	void
asMap()	Map<K, Collection<V>>

¿Qué devuelve/hace el método?

- Devuelve el número de elementos en el Multimap
- True si está vacío y False si no lo está
- True si *key* se encuentra en el Multimap
- True si *value* se encuentra en el Multimap
- True si el par *key-value* se encuentra en el Multimap
- Agrega un nuevo par *key-value*
- Devuelve la colección asociada a *key*
- Elimina el par *key-value* del Multimap
- Devuelve un Set con todas las *key* del Multimap
- Elimina todos los elementos del Multimap
- Devuelve una vista de los elementos como un Map

- Map "Bidireccional"
- Permite preservar la unicidad tanto del par key-value
- Esto habilita a los bimap el soporte para las **"inverse view"** en la cual se puede acceder al *value* desde la *key* y viceversa



Implementaciones BiMap

ImmutableBiMap

HashBiMap

EnumBiMap

EnumHashBiMap

key-value

keyA → val1

keyB → val2

value-key

val1 → keyA

val2 → keyB

Creación: *Impl.create()*

Ejemplo: HashBiMap.create()

Principales Métodos

Bimap<K, V>	
size()	int
isEmpty()	boolean
containsKey(Object key)	boolean
containsValue(Object value)	boolean
put(K key, V value)	boolean
get(K key)	V
remove(Object key, Object val)	boolean
inverse()	BiMap<V, K>
values()	Set<V>
clear()	void

¿Qué devuelve/hace el método?

- Devuelve el número de elementos en el Multimap
- True si está vacío y False si no lo está
- True si *key* se encuentra en el Multimap
- True si *value* se encuentra en el Multimap
- Agrega un nuevo par *key-value*
- Devuelve el valor (value) asociada a *key*
- Elimina el par *key-value* del Multimap
- Devuelve la estructura BiMap invertida
- Devuelve un Set de todos los valores (value) del BiMap
- Devuelve una vista de los elementos como un Map

Collections | Table

- Es una colección que asocia pares de claves (key) llamadas row key y column key con un único valor
- Permite almacenar datos en una estructura tipo tabla
- Con una tabla es posible indexar por más de un valor (row, col) y es el equivalente en Java a la siguiente definición:

Map<Row, Map<Col, Value>>

	C1	C2	C3	C4
Row 1	V1	V2	V3	V5
Row 2	V4	V6	V8	V7
Row 3	V20	V14	V6	V9
Row 4	V10	V12	V33	V11
Row 5	V54	V26	V35	V48

Implementaciones Table
ImmutableTable
HashBasedTable
TreeBasedTable
ArrayTable
Tables.newCustomTable

Creación: `Impl.create()`

Ejemplo: `HashBasedTable.create()`

Principales Métodos

Table<R, C, V>	
size()	int
isEmpty()	boolean
containsRow(Object rowKey)	boolean
containsColumn(Object columnKey)	boolean
put(R row, C col, V value)	V
get(R row, C col)	V
remove(R row, C col)	boolean
cellSet()	Set<Table.Cell<R,C,V>>
values()	Collection<V>
clear()	void

¿Qué devuelve/hace el método?

- Devuelve el número de elementos en la tabla
- True si está vacío y False si no lo está
- True si *rowkey* se encuentra dentro de las filas de la tabla
- True si *columnkey* se encuentra dentro de las columnas
- Agrega un valor (value) en la fila (row) y columna (col)
- Devuelve el valor (value) asociado a *row* y *col*
- Elimina el valor (value) asociado a *row* y *col*
- Devuelve un set con todas las celdas de la tabla
- Devuelve una colección con los valores (value) de la tabla
- Devuelve una vista de los elementos como un Map



Resources

Google Guava

<https://github.com/google/guava>

Snapshot API Docs

<https://guava.dev/releases/snapshot-jre/api/docs/index.html>

User Guide

<https://github.com/google/guava/wiki>

Thank You

fidiaz@utp.edu.pe