Primera Parte ejercicio 25 Bag

A) Lea la documentación de Map en Java y responda:

1) ¿Qué implementaciones provee Java para utilizar un Map?
¿Cuáles de ellas son destinadas a uso general?

La documentación enumera varias clases que implementan Map:
- AbstractMap
- Attributes
- AuthProvider,
- ConcurrentHashMap
- ConcurrentSkipListMap
- EnumMap
- HashMap
- Hashtable
- IdentityHashMap
- LinkedHashMap
- Properties
- Provider
- RenderingHints
- SimpleBindings,
- TabularDataSupport
- TreeMap
- UIDefaults
- WeakHashMap.

Las implementaciones de uso general :
-*HashMap:*
No garantiza orden.
Permite claves y valores nulos.
La más eficiente en promedio.
-*TreeMap:*
Ordena las claves según su orden natural o un Comparator.
No permite claves nulas.
Valores sí pueden ser nulos.
-*LinkedHashMap:*
Mantiene el orden de inserción.
Permite claves y valores nulos.
Levemente más lenta que HashMap.

2) Investigue cómo consultar sí un map contiene una determinada clave (key).
Explique qué métodos deben implementar las claves para que esto funcione correctamente.

método para saber si existe determinada clave: **boolean containsKey (Object key)**

Según la especificación de Map:

(key==null ? k==null : key.equals(k))
si la key que busca es null busca una key dentro del map que sea null
si la key tiene un valor compara con cada valor de key con .equals()
en caso de encontrar la key en el map retorna verdadero.

para que esta consulta funcione correctamente, si se usan claves de tipo personalizado (objetos), debemos asegurar que esos objetos implementen correctamente:

- *equals (Object o)* **:** para comparar igualdad lógica de claves.
- *hashCode ()* **:** porque muchas implementaciones de Map basadas en hash (por ejemplo HashMap) usan hashCode()
para ubicar la clave y luego equals() para verificar.

Advertencia:
Se debe tener mucho cuidado si se usan objetos mutables (que pueden cambiar su estado) como claves de un map.
Si una clave cambia mientras está dentro del map, puede "romperse" la estructura interna del Map.

Ejemplo:
- Guardas un objeto como clave → Java calcula su hashCode
- Si luego modificas ese objeto y cambia su hashCode o equals():
- El map ya no puede encontrar la clave, porque está "mal ubicada".

Por eso se recomienda que las claves sean inmutables (como String, Integer) o que no cambies los atributos usados en equals/hashCode.

Resumen:
Debes implementar correctamente equals() , hashCode() y evitar que la clave cambie mientras esté dentro del Map.

3) ¿Con qué método se puede recuperar el objeto asociado a una clave?
¿Qué pasa sí la clave no existe en el map?

método para obtener valor asociado a una clave determinada: **V get (Object key)**

V: el tipo del valor
Devuelve el valor al que está asociada la clave especificada, o null si este map no contiene ninguna asociación para esa clave.

Importante:
Si el map permite valores null, entonces que get() devuelva null no necesariamente significa que la clave no existe en el map.
También es posible que la clave exista pero esté asociada explícitamente al valor null.

Ejemplo:
Si el map acepta valores null (por ejemplo, HashMap sí los acepta), entonces:
Caso 1: La clave NO está en el map → get() devuelve null
Caso 2: La clave SÍ está en el map, pero su valor es null → get() también devuelve null

Entonces null tiene dos significados posibles.
¿Cómo distinguir si la clave existe realmente?
La operación *containsKey* puede usarse para distinguir estos 2 casos:
Si containsKey (key) es true → la clave existe (aunque su valor sea null)
Si containsKey(key) es false → la clave no está en el mapa

4) Investigue cómo agregar claves y valores a un map.
¿Qué pasa sí la clave ya se encontraba en el map?
¿Permite agregar claves y/o valores nulos?

método para agregar (o actualizar) : **V put (K key, V value)**

Asocia el valor especificado con la clave especificada en este mapa.
Si el mapa ya contenía una asociación para esa clave, el valor viejo se reemplaza por el nuevo.

- Si la clave no existe, se agrega.
- Si la clave ya existe, se reemplaza el valor anterior por el nuevo.

Devuelve el valor previo asociado a la clave, o null si no había ninguna asociación para esa clave.
(Un valor null devuelto también puede significar que la clave estaba asociada a null, si la implementación permite valores null.)

Ejemplo:

V anterior = map.put(k, v);

posibles valores de anterior:
- clave NO existía → devuelve **null**
- clave existía → devuelve el **valor anterior**
- clave existía y su valor previo era **null** → devuelve null, igual que el caso anterior

Algunas implementaciones de Map tienen restricciones sobre las claves o los valores que pueden contener.
Por ejemplo, algunas implementaciones prohíben claves o valores null.
Intentar insertar una clave o valor inválido lanza una excepción no verificada, típicamente NullPointerException o ClassCastException.

El uso de null depende de la implementación:

HashMap: permite null en clave y valor.

TreeMap: generalmente NO permite null en clave.
(No acepta null como clave **cuando usa orden natural**, porque no puede comparar null)

Hashtable: NO permite null en ningún lado.

5) Determine cómo se pueden eliminar claves y valores de un map.
¿Es necesario controlar la presencia de alguno de ellos?

método para eliminar clave y valor: **V remove (Object key)**
elimina la clave y el valor asociado a la Clave de existir.

retorna el valor que el map tenía asociado previamente a esa clave, o null si el map no tenía ninguna asociación para esa clave.

Ejemplo:

V viejo = map.remove(key);

posibles valores de viejo:
- La clave existía → devuelve el valor que tenía.
- La clave no existía → devuelve null.
- La clave existía con valor null → devuelve null.

En el último caso si se quiere controlar si realmente se eliminó algo:

```
if (map.containsKey ("A"))
    map.remove ("A");
```

Primero se verifica si la clave está presente en el mapa.
Si la clave existe, se realiza la eliminación, sin importar si el valor asociado es null o tiene otro valor.
En cambio, si la clave no existe, simplemente no se elimina nada.
De esta manera, antes de borrar, nos aseguramos de que la clave esté realmente almacenada en el mapa.

El método remove también permite enviar el valor junto con la clave para asegurarse de que se elimine exactamente ese par clave–valor. No es obligatorio usarlo así, solo es una forma de garantizar que se borre lo correcto y evitar eliminar otro elemento por error.

map.remove("A", "Hola"); // Solo se elimina si A → "Hola"

en el método remove No se puede eliminar una clave sin eliminar también su valor.
Java Maps siempre eliminan parejas completas clave–valor.

método para vaciar map: **map.clear ();**

* Algunas implementaciones no soportan eliminar. En ese caso, llamar remove o clear lanza una UnsupportedOperationException.

Ejemplo:

```
Map<String, Integer> mapa = Map.of(1,2);   // mapa inmodificable
mapa.remove (1); // lanza excepción
```

6) Investigue cómo reemplazar un valor en un map.

Además del método put, la interfaz Map provee métodos específicos para reemplazar valores:

1) **default V replace (K key, V value)**
- Si la clave existe → reemplaza el valor y devuelve el anterior.
- Si la clave no existe → no agrega nada y devuelve null.

1) Ejemplo:
```
Map<String, Integer> map = new HashMap<>();
map.put ("A", 10);
Integer viejo = map.replace ("A", 20); // Se reemplaza 10 → 20
System.out.println(viejo); // Imprime: 10
map.replace ("B", 30); // No hace nada (la clave "B" no existe)
```

2) **default boolean replace (K key, V viejoValor, V nuevoValor)**
- Solo reemplaza si coincide la clave *y también* el valor actual.
- Ideal para evitar cambios por error.

2)Ejemplo:
```
Map<String, Integer> map = new HashMap<>();
map.put ("A", 10);
boolean reemplazo1 = map.replace ("A", 10, 20); // true (se reemplaza)
boolean reemplazo2 = map.replace ("A", 10, 30); // false (ya no vale 10)
System.out.println(reemplazo1); // Imprime: true
System.out.println(reemplazo2); // Imprime: false
```

3) **replaceAll (BiFunction <? super K, ? super V, ? extends V> function)**
Reemplaza los valores de todas las entradas del mapa aplicando la función indicada.
- Recorre todo el mapa y reemplaza solo los valores, manteniendo las claves.
- Útil para actualizar todo en base a alguna regla.

3)Ejemplo:
```
Map <String, Integer> map = new HashMap<>();
map.put("A", 10);
map.put("B", 5);
map.put("C", 3);
map.replaceAll ((k, v) -> v * 2); // Duplica todos los valores
System.out.println(map); // Imprime: {A=20, B=10, C=6}
```

Estos métodos permiten **actualizar valores sin correr el riesgo de modificar una clave inexistente**, a diferencia de put, que también puede agregar.

Son especialmente útiles cuando queremos:

- Actualizar solo si la clave ya estaba registrada → replace(key, value)
- **Actualizar solo si el valor coincide con uno esperado** → replace(key, oldValue, newValue)
- **Modificar todos los valores del mapa en base a una regla** → replaceAll(...)

7) Teniendo en cuenta los métodos:
keySet()
values()
entrySet()
explique de qué formas se puede iterar un mapa
¿Es posible utilizar streams?

1- **keySet()** → devuelve un Set <K> que representa las claves del map.
Los cambios en el map se reflejan en el conjunto, y viceversa (agarra la vista respaldada).

*1) Iterar sobre las claves (**keySet()**) y luego obtener el valor para cada clave:*

```
for (K key : map.keySet() ) {
    V value = map.get(key);
   // procesar key + value
}
```

* *Esta forma puede implicar una consulta get() para cada clave, lo que puede no ser tan eficiente dependiendo de la implementación.*

2- **values()** → devuelve una Collection <V> con los valores del map.
De nuevo, es una vista respaldada.

2) *Iterar sobre los valores (values()):*

```
for (V value : map.values()) {
   // procesar value
}
```

* *Se pierde la referencia directa a la clave, es solo el valor.*

3- **entrySet()** → devuelve un Set < Map.Entry<K,V> > con los pares clave-valor.
Cada elemento es un Map.Entry <K,V> que permite obtener la clave (getKey ()) y el valor (getValue ()), e incluso modificar el valor con setValue (), si la implementación lo permite.

3) Iterar sobre las entradas (entrySet()):

```
for (Map.Entry <K,V>  entry : map.entrySet()) {
   K key = entry.getKey();
   V value = entry.getValue();
   // procesar key + value
}
```

* *Esla forma más eficiente cuando se desea tanto clave como valor, porque evita la llamada adicional get() y se trabaja directamente con los pares.*

Uso de streams : Si, es posible usar streams en los maps, ya que keySet(), values(), entrySet())

implementan Collection/Set, se pueden convertir en streams.
Además, la interfaz Map tiene por defecto el método:
forEach (BiConsumer<? super K,? super V> acción) para iterar de forma funcional.
Ejemplo:

```
map.entrySet().stream()
   .filter (entry -> entry.getValue() != null)
   .forEach (entry -> System.out.println(entry.getKey() + " -> " + entry.getValue()));
```

** También usando map.keySet().stream() o map.values().stream() si sólo interesa una parte.*

Resumen:

SABER SI EXISTE UNA CLAVE EN MAP:
boolean containsKey (Object key)

DEVUELVE VALOR ASOCIADO A CLAVE:
V get (Object key)

AGREGAR O MODIFICAR CONTENIDO DE MAP:
put (key, value)
si la clave no existe la agrega y devuelve null.
si la clave existe remplaza por nuevo valor y retorna el viejo.

OTRAS FORMAS DE REMPLAZAR VALORES :
replace (K key, V value)
si existe clave remplaza por nuevo valor y devuelve el viejo si no existe clave retorna null.
boolean replace (K key, V viejoValor, V nuevoValor)
solo modifica si la clave y el valor viejo se encuentra en map de ser así modifica viejo valor por nuevo.
map.replaceAll ((k, v) -> v * 2)
aplica la funcion (v* 2) a todo los valores que tenga map, no modifica ninguna clave.

ELIMINAR CLAVE Y VALOR:
V remove (Object key)
retorna valor que map tenía asociado previamente a esa clave, o null si el map no tenía ninguna asociación para esa clave.
Antes de borrar podría verificar existencia de clave en map:

```
if (map.containsKey ("A"))
    map.remove ("A");
```

VACIAR MAP:
map.clear()

FORMAS DE ITERAR EN MAP:
1**.map.keySet()**

2.**map.values()**
3.**map. entrySet()**