**A)¿Qué implementaciones provee Java para utilizar un Map? ¿Cuáles de ellas**

**son destinadas a uso general?**

Java proporciona varias implementaciones de la interfaz Map en el paquete java.util, cada una diseñada para casos de uso específicos. A continuación, te explico las principales implementaciones y cuáles de ellas son destinadas para uso general:

1. HashMap

Descripción: Es una de las implementaciones de Map más común. Almacena los elementos en una tabla hash, lo que permite acceder a los elementos en tiempo constante O(1) en promedio para operaciones de inserción, búsqueda y eliminación.

Uso: Es adecuada para situaciones donde el orden de inserción no es importante, y se necesita una estructura de datos rápida para búsquedas y almacenamiento.

Características:

Permite null como clave y como valor.

No es sincronizada (no es segura para múltiples hilos).

No garantiza ningún orden de los elementos.

Uso General: Sí, es una implementación de uso general.

1. LinkedHashMap

Descripción: Extiende HashMap y mantiene una lista doblemente enlazada de los elementos según el orden en que fueron insertados. Esto permite que el mapa mantenga un orden de inserción.

Uso: Ideal cuando se desea un Map con orden predecible, como el orden de inserción o el orden de acceso (se puede configurar).

Características:

Permite null como clave y como valor.

No es sincronizada.

Mantiene el orden de inserción o el de acceso.

Uso General: Sí, es una implementación de uso general.

1. TreeMap

Descripción: Implementa la interfaz NavigableMap, que a su vez extiende SortedMap. Almacena los elementos en un árbol rojo-negro, lo que permite mantener los elementos ordenados de acuerdo con el orden natural de las claves o un Comparator proporcionado.

Uso: Es útil cuando se necesita un Map ordenado de forma natural o con un orden específico definido por un Comparator.

Características:

No permite null como clave (aunque permite null como valor).

No es sincronizada.

Mantiene los elementos ordenados de acuerdo con el orden de las claves.

Uso General: Sí, es una implementación de uso general.

1. Hashtable

Descripción: Es una implementación más antigua de Map, introducida antes de que existiera la interfaz Map. Es similar a HashMap, pero está sincronizada, lo que significa que es segura para su uso en aplicaciones multihilo.

Uso: Generalmente se desaconseja su uso en favor de ConcurrentHashMap para entornos de múltiples hilos, ya que tiene una implementación de sincronización menos eficiente.

Características:

No permite null como clave ni como valor.

Está sincronizada.

No garantiza ningún orden de los elementos.

Uso General: Aunque es de uso general, su uso es menos común en las aplicaciones modernas debido a la existencia de alternativas más eficientes.

1. ConcurrentHashMap

Descripción: Es una implementación de Map optimizada para su uso en aplicaciones multihilo. Permite operaciones concurrentes eficientes dividiendo internamente el mapa en segmentos.

Uso: Ideal para aplicaciones de alto rendimiento que requieren acceso concurrente sin bloquear todo el mapa.

Características:

No permite null como clave ni como valor.

Está sincronizada para operaciones concurrentes.

No garantiza ningún orden de los elementos.

Uso General: Sí, es ampliamente utilizada para aplicaciones multihilo, aunque no es de uso general en aplicaciones de un solo hilo.

Resumen

De estas implementaciones, las que se consideran de uso general para aplicaciones de un solo hilo son:

HashMap

LinkedHashMap

TreeMap

Para entornos concurrentes, la implementación recomendada es ConcurrentHashMap en lugar de Hashtable, debido a su eficiencia mejorada en aplicaciones multihilo.

**B) Investigue cómo consultar sí un mapa contiene una determinada clave (key).**

**Explique qué métodos deben implementar las claves para que esto funcione**

**Correctamente**

Se utiliza el metodo boolean containsKey(Object key), retorna true si el mapa contiene una asociacion para la claveespecificada. Esto lo hace usando el metodo equals de object. Por lo que la clase de key deberia tener definido este metodo para definir el criterio de comparacion entre claves.

1. **¿Con qué método se puede recuperar el objeto asociado a una clave? ¿Qué**

**pasa sí la clave no existe en el mapa?**

Con el metodo V get(Object key) devuelve un objeto tipo V (value) si es que existe asociasion con la clave pasada como parametro, caso contrario devuelve null

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

V put(K key, V value) asocia estos valores en el map. Si existe previamente, sobreescribe el valor. Devuelve el value para el cual estaba asociada la key. Para agregar valores nulos, depende de la implementacion, si esta no soporta valores null, al intentarlo tendra retorno null pointer exception.

1. **Determine cómo se pueden eliminar claves y valores de un mapa. ¿Es necesario controlar la presencia de alguno de ellos?**

V remove(Object key) elimina el mapeo a la key recibida, si existe. El retorno null, en caso del que la clase que implementa Map permita valores null, no necesariamente implica que el mapa no contenia la key recibida, tambien puede ser el caso de que la key estaba mapeada a un valor null.

1. **Investigue cómo reemplazar un valor en un mapa**

con el comando put, segun punto D si se provee la key.

1. **Teniendo en cuenta los métodos keySet(), values() y entrySet(), explique de qué formas se puede iterar un mapa ¿Es posible utilizar streams?**

Set<K> keySet() devuelve un conjunto Set que contiene todas las claves del mapa.

Value

Collection<V> values() devuelve una “Collection” de los valores de un mapa,

Set<Map.Entry<K, V>> entrySet() devuelve un conjunto Set de las entradas (pares key-value del mapa).

Todas estan asociadas al mapa, por lo que si se modifica el Set, se modifica el mapa y viceversa.

Todas deberian poder iterar mediante stream, dado que implementan “Collection”.

**2da parte**

1. **Liste los métodos que debe contener una clase que implementa la interface Bag<T>.**

Boolean add(T element)

Int ocurrencesOf(T element)

Void removeOcurrence(T element)

Void removeAll(T element)

Int size ()

1. **Explique cómo implementaría un Bag<T> usando composición con un Map<K, V> ¿De qué tipo tendrían que ser las claves y valores del Map?**

Implementando Bag T usando composicion, las keys deberian ser los elementos, y los values las ocurrencias.

Bag<T> extends HasMap<T, Integer>

1. **Implemente la interfaz Bag<T>, utilizando AbstractCollection<T> como superclase, y componga con un Map<T, V>.**
2. **---**

**¿Cuáles son los beneficios de utilizar AbstractCollection como superclase**

**para implementar el Bag?**

Te obliga a implementar ciertas funciones derivadas de los metodos abstractos de la clase abstract Collection. Entiendo que esto hace que emules completamente el comportamiento de una coleccion pero con una base que reside en una estructura tipo map.

Por otro lado, al usar abstractCollection usamos una clase abstracta, esto reduce la cantidad de metodos a implementar, como el caso de implementar una interfaz (Collection)

**¿Qué ventajas tiene componer con un Map para implementar el Bag?**

Evita el chequeo de elementos repetidos. Facilita el acceso a los elementos y su cantidad, con mayor rapidez usando hasMap, hace menos costosa su actualizacion en cuanto a tiempos.

**En lugar de componer con un Map, ¿es posible extenderlo para poder**

**implementar el Bag? ¿Qué diferencias tendría esa solución con respecto a la**

**planteada en este ejercicio**?

El bag deberia implementarse necesariamente con Integer, es decir Bag<T, Integer>.

**¿Para qué cree que podría ser útil un objeto Bag?**

Evaluar ocurrencias en algun contexto ? situaciones con alta frecuencia en acceso a informacion ?

public int occurrencesOf(T element) {

return elements.get(element);

}

Verificar si contiene key, sino debe devolver 0.

@Override

public void removeOccurrence(T element) {

if(elements.containsKey(element) && elements.get(element)>0) {

elements.remove(element);

}

}

Verificar >1 sino debo remover

Si es >1 debo hacer put y restar 1 al value.

@Override

public void removeAll(T element) {

if(elements.get(element)>0) {

elements.remove(element);

}

}

Ver en documentación que hace el map en caso de remover una key inexistente, en base a eso sea chequeara o no que la clave exista.

Para test hacer por separado por cada clase una clase test, y los métodos a testear cada uno con su test por separado.