Ejercicio 25 - A - Preguntas

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

- Implementaciones principales (Java SE):
  - HashMap
  - LinkedHashMap
  - TreeMap
  - EnumMap
  - WeakHashMap
  - IdentityHashMap
  - Hashtable (legado)
  - ConcurrentHashMap (concurrente)
- Destinadas a uso general (las que se usan en la mayoría de los casos):
  - HashMap: mapa hash no ordenado, buen rendimiento general.
  - LinkedHashMap: como HashMap pero mantiene orden de inserción (o acceso) — útil cuando se necesita orden estable.
  - TreeMap: mapa ordenado (implementa SortedMap / NavigableMap), útil cuando se requiere orden por clave. (Las demás son especializadas: EnumMap para claves enum, WeakHashMap para claves GC-weak, IdentityHashMap usa == en vez de equals, Hashtable y ConcurrentHashMap para uso concurrente/legado.)

b) ¿Cómo consultar si un mapa contiene una determinada clave? ¿Qué deben implementar las claves?

- Método: map.containsKey(Object key)
- Comportamiento:
  - En mapas basados en hashing (HashMap, LinkedHashMap, WeakHashMap, IdentityHashMap? Identity usa ==) la comparación usa hashCode() y equals(): las claves deben implementar correctamente equals(...) y hashCode() para que containsKey funcione correctamente.

    - En mapas ordenados (TreeMap) la búsqueda usa comparación por orden: las claves deben ser comparables (implementen Comparable) o debe proveerse un Comparator al TreeMap; de lo contrario se lanzará ClassCastException o NullPointerException si se usa null.
- Nota: containsValue(Object value) existe para comprobar valores; la comprobación de valores usa equals(...) sobre los valores.

c) ¿Con qué método se recupera el objeto asociado a una clave? ¿Qué pasa si la clave no existe?

- Obtener valor: V v = map.get(Object key)
- Si la clave no existe, get devuelve null.
- Ambigüedad: si hay una entrada mapeada explícitamente a null, get también devuelve null; para distinguir:
    - usar map.containsKey(key) para saber si la clave está presente o
    - usar map.getOrDefault(key, defaultValue) (Java 8+) para devolver un valor por defecto si falta.
- Ejemplo:
    - Integer x = map.get("k");
    - Integer x = map.getOrDefault("k", 0);

d) ¿Cómo agregar claves y valores? ¿Qué pasa si la clave ya estaba? ¿Se admiten nulls?

- Agregar / reemplazar: map.put(K key, V value)
    - Si la clave no existía, se inserta la nueva entrada.
    - Si la clave ya existía, put reemplaza el valor anterior y devuelve el valor previo (o null si no había valor previo o era null).
- También hay putIfAbsent(K key, V value) (ConcurrentMap/Map default con implementaciones) que pone solo si no existe.
- ¿Permite null key / null values? Depende de la implementación:
    - HashMap, LinkedHashMap, WeakHashMap, IdentityHashMap: permiten una clave null (HashMap permite exactamente una clave null) y valores null (varios valores null).

    - TreeMap: normalmente no permite null para claves si usa orden natural (lanzará NPE o CCE); si se provee Comparator que acepte null, puede depender del Comparator.
    - EnumMap: no permite clave null (NullPointerException).
    - Hashtable: no permite claves ni valores null (lanza NullPointerException).
    - ConcurrentHashMap: no permite claves ni valores null.
- Resumen: comprueba la implementación antes de usar nulls.

e) ¿Cómo eliminar claves/valores? ¿Hay que controlar su presencia?

- Métodos:
    - V removed = map.remove(Object key) — elimina la entrada por clave y devuelve el valor eliminado (o null si no existía o si el valor era null).
    - boolean removed = map.remove(Object key, Object value) — elimina sólo si la clave está mapeada al valor dado; devuelve true si se eliminó.
    - map.clear() — elimina todas las entradas.
- No es estrictamente necesario comprobar containsKey antes de remove salvo que quieras usar la presencia para lógica adicional; remove devuelve información útil (valor previo o boolean) para saber si hubo borrado.
- Ejemplo:
    - Integer old = map.remove("k");
    - boolean ok = map.remove("k", 42);

f) ¿Cómo reemplazar un valor en un mapa?

- Métodos específicos:
    - V old = map.replace(K key, V value) — reemplaza el valor existente para key; devuelve valor previo (o null si no existía).
    - boolean ok = map.replace(K key, V oldValue, V newValue) — reemplaza solo si el valor actual es oldValue; devuelve true si se realizó.
    - map.replaceAll(BiFunction<? super K,? super V,? extends V> function) — reemplaza todos los valores con el resultado de la función.
- Otras alternativas con API funcional (Java 8+):

    - map.computeIfPresent(key, (k,v) → nuevoValor)
    - map.compute(key, (k,v) → nuevaAsignación) — permite crear o eliminar (devuelve null para eliminar).
    - map.merge(key, value, BiFunction) — combina valores.
- Ejemplo:
    - map.replace("k", 100);
    - map.computeIfPresent("k", (k,v) → v + 1);

g) Iterar un mapa: keySet(), values(), entrySet(). ¿Se puede usar streams?

- keySet(): devuelve Set<K>. Iterar sobre claves:
    - for (K k : map.keySet()) { V v = map.get(k); ... } — pero esta forma hace lookups extra; no es la más eficiente si necesitas clave y valor.
- values(): devuelve Collection<V>. Iterar solo valores:
    - for (V v : map.values()) { ... }
- entrySet(): devuelve Set<Map.Entry<K,V>>. Forma recomendada para iterar claves y valores eficientemente:
    - for (Map.Entry<K,V> e : map.entrySet()) { K k = e.getKey(); V v = e.getValue(); }
    - entry.setValue(...) puede modificar el valor de la entrada.
- API funcional / forEach:
    - map.forEach((k,v) → { ... }); — conviene para acciones sencillas sin crear streams.
- Streams:
    - Sí, se pueden usar streams sobre keySet, values o entrySet:
        - map.entrySet().stream().filter(e → ...).map(e → ...) ...
        - map.keySet().stream()
        - map.values().stream()
    - También es posible crear streams paralelos con .parallelStream() en las colecciones.
    - Ejemplos:

        - map.entrySet().stream().filter(e → e.getValue() > 10).map(Map.Entry::getKey).collect(...)
        - map.values().stream().distinct().count()
- Notas de rendimiento y concurrencia:
    - Iterar entrySet es la forma más eficiente para leer clave+valor sin búsquedas adicionales.
    - Para mapas concurrentes (ConcurrentHashMap) la vista y sus streams tienen semánticas de concurrencia específicas; evita suponer consistencia rígida mientras hay escrituras concurrentes.
- Ejemplo de iteración compacta:
    - map.forEach((k,v) → System.out.println(k + " → " + v));
    - map.entrySet().stream().filter(e → e.getValue() != null).forEach(e → ...);

Resumen breve final:

- Para la mayoría de usos elige HashMap (o LinkedHashMap si necesitas orden) o TreeMap (si necesitas orden por clave).
- Para containsKey en mapas hash, implementa equals y hashCode correctamente; en TreeMap usa Comparable o un Comparator.
- get devuelve null si no existe (usar containsKey o getOrDefault para distinguir).
- put inserta/reemplaza; comportamiento con null depende de la implementación (HashMap permite null, Hashtable/ConcurrentHashMap no).
- remove/remove(key,value)/clear son los métodos para borrar; no es obligatorio comprobar la presencia antes de remove (usa el valor de retorno si lo necesitas).
- replace/compute/merge/replaceAll son las formas de actualizar valores.
- Itera preferentemente sobre entrySet para obtener clave y valor; puedes usar forEach y también streams sobre entrySet/keySet/values.