Listas. Clases utilizadas

Ventajas

- Acceso aleatorio.
- Están ordenadas (collection.sort()).
- Añadir / eliminar sin restricción.
- ListIterator modifica en cualquier dirección.
- Sintaxis similar a Arrays.

Inconvenientes

- Bajo rendimiento en operaciones concretas que se resolverían mejor con otras interfaces.

Listas. Clases utilizadas

ArrayList

- Muy rápida accediendo a elementos.
- Se adapta a un gran número de escenarios.

LinkedList

- Listas enlazadas.
- Gran eficiencia agregando y eliminando elementos.

Vector

- Considerada como colección obsoleta.
- Utilizada únicamente en operaciones de concurrencia.

CopyOnWriteArrayList

- Utilizada en programas concurrentes.
- Eficiente en operaciones de lectura pero muy poco eficiente en operaciones de escritura.

Sets. Clases utilizadas

Ventajas

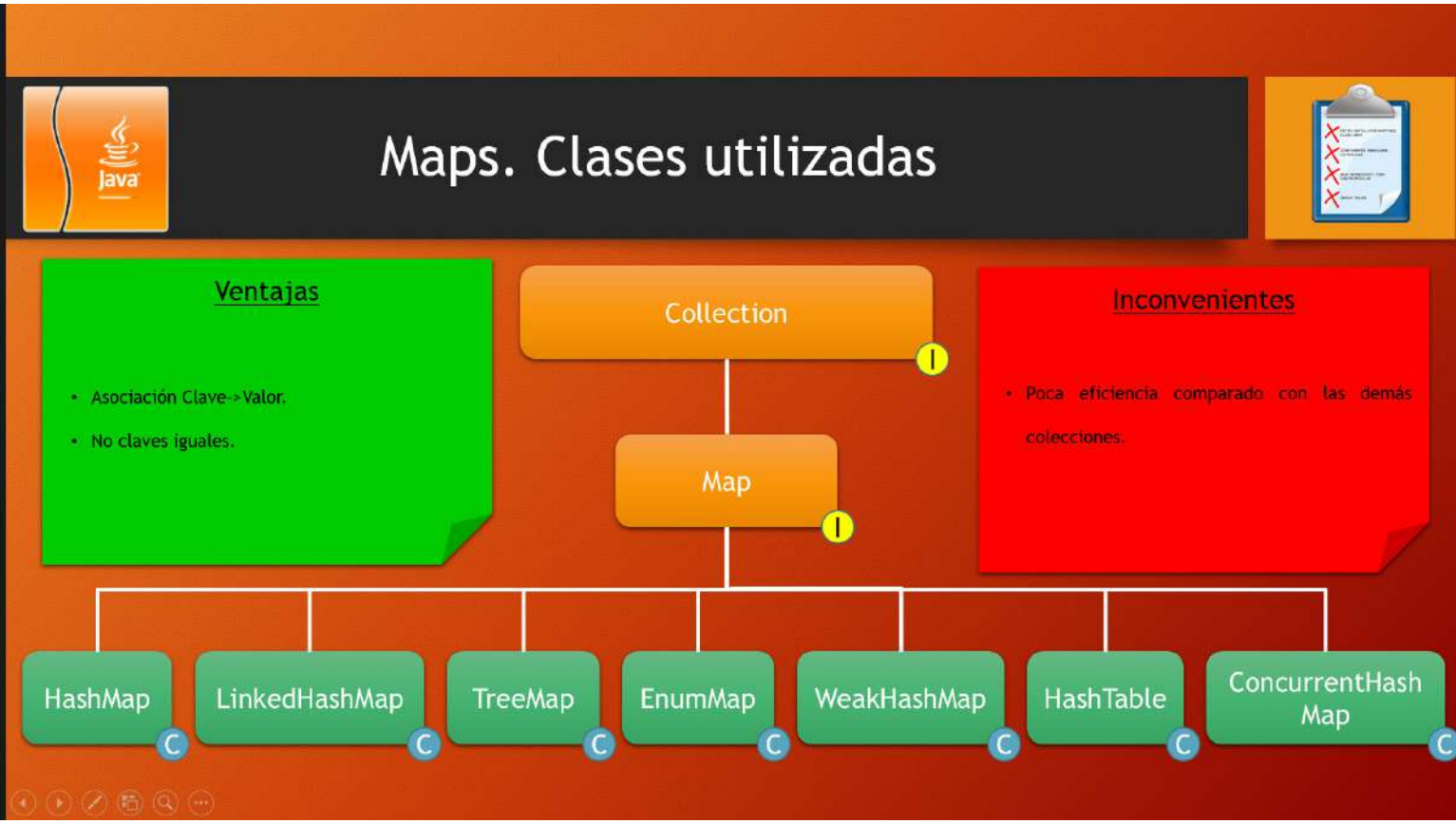
- No permiten elementos duplicados.
- Uso sencillo del método add que además asegura no elementos duplicados.

Inconvenientes

- No tienen acceso aletorio.
- Poca eficiencia a la hora de ordenar elementos (Y no siempre se puede).

Sets. Clases utilizadas

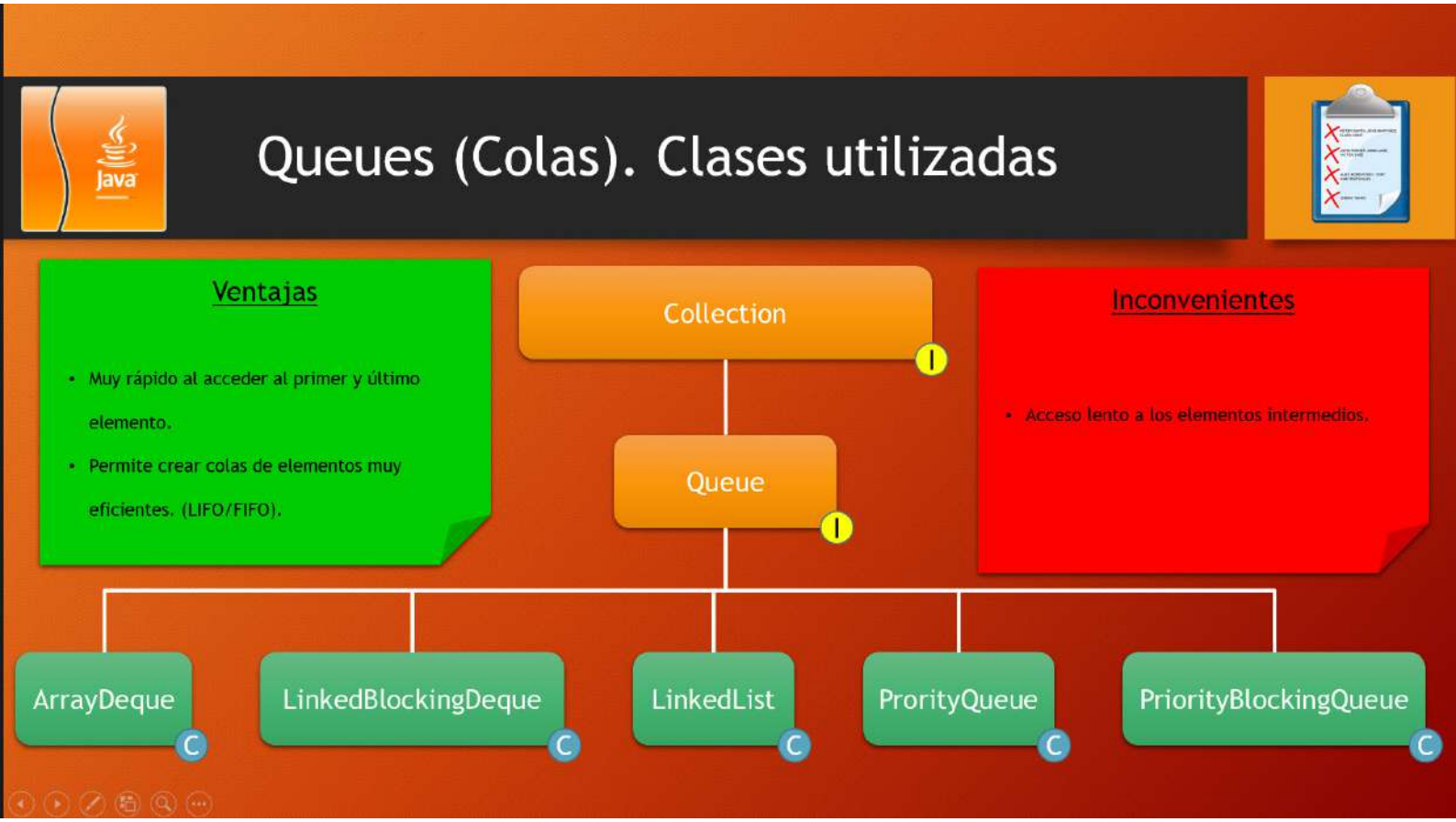
HashSet	LinkedHashSet	TreeSet	EnumSet	CopyOnWrite ArraySet	ConcurrentSkipList Set
• Rápida. • No duplicados. • No ordenación. • No acc. aleatorio	• Ordenación por entrada. • Eficiente al acceder. • No eficiente al agregar.	• Es ordenado. • Poco eficiente.	• La mejor para tipos enumerados.	• Específico concurrencia. • Eficiente lectura. • Poca eficiente escritura. • Poco eficiente al eliminar.	• Específico concurrencia. • Admite ordenación. • Con muchos elementos no es eficiente.


Maps. Clases utilizadas
Ventajas
• Asociación Clave->Valor.
• No claves iguales.
Collection
I
Map
I
Inconvenientes
• Poca eficiencia comparado con las demás colecciones.
HashMap
C
LinkedHashMap
C
TreeMap
C
EnumMap
C
WeakHashMap
C
HashTable
C
ConcurrentHash Map
C

Maps. Clases utilizadas

HashMap	LinkedHashMap	TreeMap	EnumMap	WeakHashMap	HashTable	ConcurrentHashMap
• No ordenación. • Eficiente.	• Ordenado por inserción. • Permite ordenación por uso. • Eficiente lectura. • Poca eficiente escritura	• Ordenado por clave. • Poco eficiente en todas sus operaciones	• Permite enum como claves. • Muy eficiente	• Utilizado para crear elementos que vaya borrando el sistema si no se utilizan. • Muy poco eficiente	• Considerado obsoleto. • Utilizado en operaciones de concurrencia	• Utilizado en concurrencia. • No permite nulos

Queues (Colas). Clases utilizadas



Queues (Colas). Clases utilizadas

ArrayDeque

- Gran eficiencia.
- La más utilizada.

LinkedBlockingDeque

- Utilizado en programación concurrente.

LinkedList

- Rendimiento inferior al ArrayDeque

ProrityQueue

- Para utilizar un Comparator.
- El primer elemento dependerá de propiedad elegida.

PriorityBlockingQueue

- Igual que el anterior pero más eficiente en programación concurrente