

Módulo profesional Programación

Repaso de estructuras de datos dinámicas

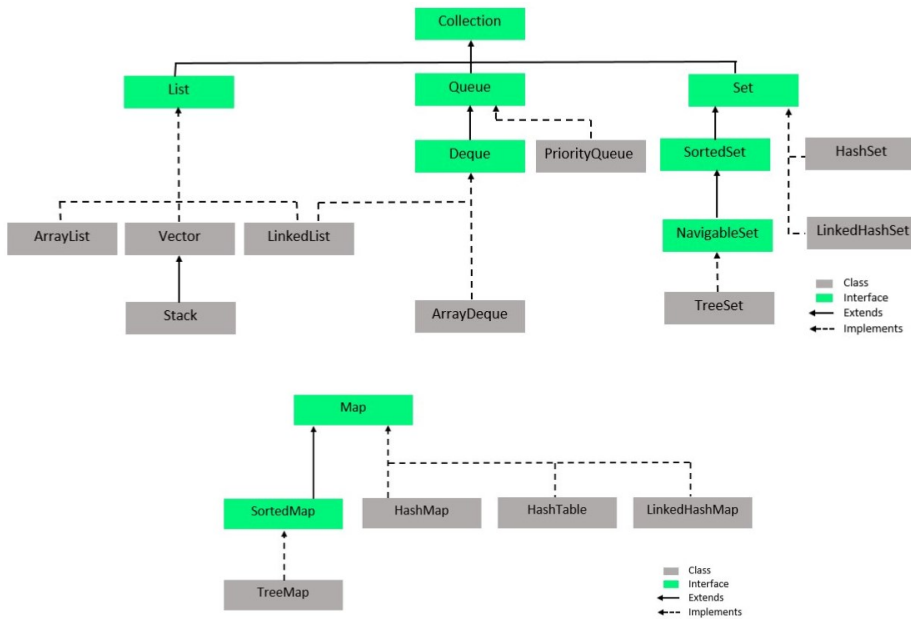
¿Qué estructura de datos utilizar?

Vectores y Matrices

2	4	3	1	5
0	1	2	3	4

4 filas

5 columnas					
	0	1	2	3	4
0	1	8	7	15	16
1	6	2	14	17	5
2	9	10	3	18	10
3	12	11	13	4	20



Eligiendo una colección

Si quieres escribir un programa que trate a los objetos como una colección de elementos tienes **varias opciones**.

El siguiente resumen te ayuda a elegir una colección apropiada para tu aplicación.

Eligiendo una colección

	 A List of Books Lista (List)	 A Set of Books Conjunto (Set)	 Keys Values Mapa (Map)	 A Stack of Books Pilas/Colas (Stack/Queue)
Elementos	Valor	Valor	Clave/Valor	Valor
Acceso	Acceso por posición	Acceso posicional no permitido	Acceso por clave	Acceso por extremos
Duplicados	Admite duplicados	No admite duplicados	No admite claves duplicadas	
Orden	Mantiene el orden de inserción	Depende de la implementación	Depende de la implementación	

Eligiendo una colección






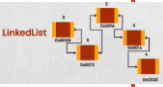
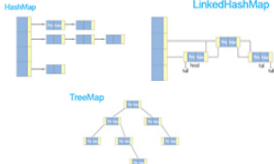
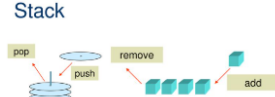
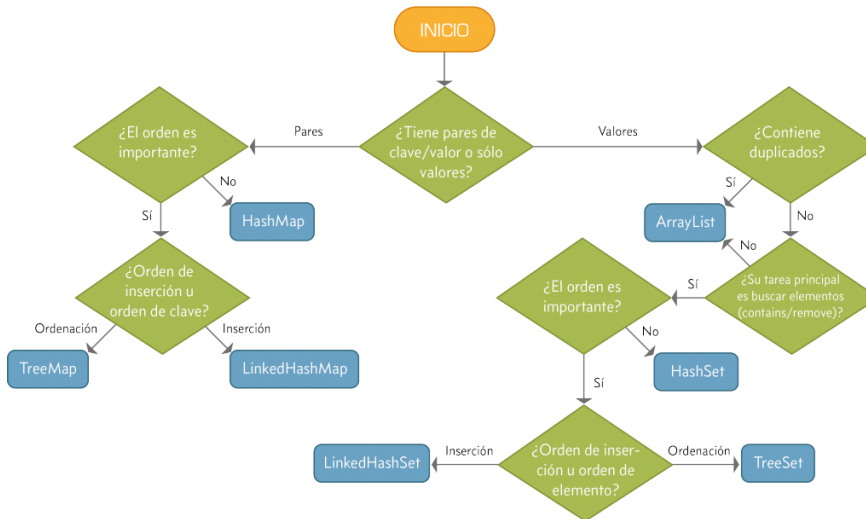
 A List of Books Lista (List)	 A Set of Books Conjunto (Set)	 Keys Values Mapa (Map)	 A Stack of Books Pilas/Colas (Stack/Queue)
ArrayList LinkedList 1	HashSet LinkedHashSet TreeSet 2	HashMap LinkedHashMap TreeMap 3	Stack LinkedList 4 5
 Acceso rápido por posición	 Eficiente en inserciones/borrados Importa el orden TreeSet menos eficiente	 Importa el orden TreeMap menos eficiente	 LIFO (Last-In-First-Out) access method FIFO (First-In-First-Out) access method

Diagrama de decisión para uso de colecciones Java



Eligiendo una colección

PASO 1 ¿Cómo quieres acceder a valores individuales? Tienes varias opciones:

- Si accedes a los **valores por un índice**. Utilizar una lista.
- Si accedes a los **valores por una clave** que no forma parte del objeto. Utilizar un **mapa**.
- Si accedes **a los valores sólo en uno de los extremos**. Utilice una **cola** (para el acceso del primero en entrar, primero en salir) o una **pila** (para el acceso del último en entrar, primero en salir).
- Si no es necesario acceder a los valores individuales por posición. Ve a los pasos 3 y 4

Eligiendo una colección

PASO 2 Determina los tipos de elementos o los tipos de clave/valor.

Para una **lista o un conjunto**, determina el tipo de los elementos que deseas almacenar. Por ejemplo, si almacenas un conjunto de libros, entonces el tipo de elemento es Libro.

Del mismo modo, **para un mapa**, determina los tipos de las claves y los valores asociados. Si quieres buscar libros por ID, puede utilizar un Mapa<Integer, Libro> o un Mapa<String, Libro>, dependiendo de el tipo de ID.

Eligiendo una colección

Paso 3 Determina si el orden de los elementos o de las claves es importante. Cuando se visitan elementos de una colección o claves de un mapa, **¿Te importa el orden en el que se visitan?** Tiene varias opciones:

- Si Los elementos o las claves deben estar **ordenados**. Utiliza un **TreeSet** o un **TreeMap**. Ve al paso 6.
- Si los elementos deben estar en el mismo **orden en que fueron insertados**. Tu elección se reduce a un **LinkedList** o un **ArrayList**.
- No importa. Mientras consigas visitar todos los elementos, no te importa en qué orden. Si elegiste un mapa en el Paso 1, usa un **HashMap** y ve al Paso 5.

Eligiendo una colección

Paso 4 Para una **colección**, determina qué operaciones deben ser eficientes. Tienes varias opciones:

- Encontrar elementos debe ser eficiente. Utiliza un **HashSet**.
- Debe ser eficiente añadir o eliminar elementos al principio o, siempre que se esté ya está inspeccionando un elemento allí, otra posición. Utiliza un **LinkedList**.
- Sólo insertas o eliminas al final, o recoges tan pocos elementos que no te preocupa la velocidad. Utiliza una **ArrayList**.

Eligiendo una colección

Paso 5 Para los **conjuntos y los mapas**, decide si necesitas implementar los métodos **hashCode** y **equals**

- Si tus elementos o claves pertenecen a una clase que otra persona ha implementado, comprueba si la clase tiene sus propios métodos **hashCode** y **equals**. Si es así, ya está todo listo. Este es el caso de la mayoría de las clases de la biblioteca estándar de Java, como **String**, **Integer**, **Rectangle**, etc.
- Si no, decide si puedes comparar los elementos por identidad. Este es el caso si nunca construyes dos elementos distintos con el mismo contenido. En ese caso, no es necesario hacer nada, los métodos **hashCode** e **equals** de la clase **Object** son apropiados.
- En caso contrario, deberás implementar tus propios métodos **equals** y **hashCode**

Recursos

- [Tutorial de Java Collections](#)