Colecciones

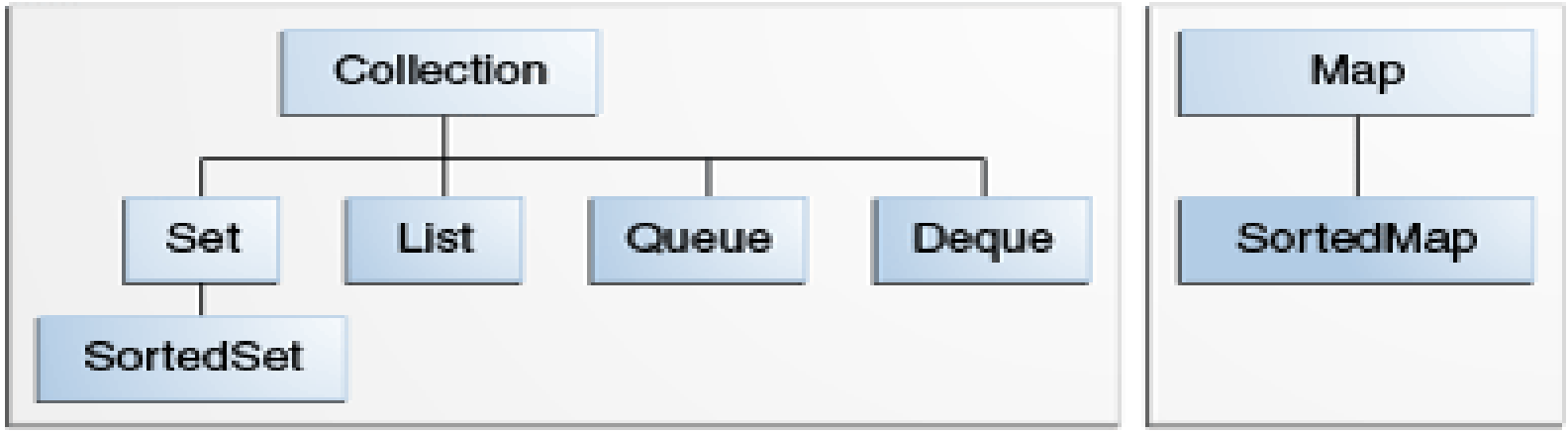
Core Collection Interfaces

Estructuras de datos

- Estructura de Datos: Es una representación de una colección de datos junto con las operaciones que se pueden hacer sobre ellos.
- Son reusables: se pueden usar para resolver distintos problemas.
- Para cada problema, distintas estructuras pueden tener diferentes tiempos de respuesta (eficiencia temporal).
- Comenzaremos usando estructuras ya provistas por Java para ver cómo se usan. No necesitamos saber cómo están implementadas para usarlas, aunque sí debemos saber cuál es la eficiencia de cada operación.

Colección

- Es un objeto simple que representa un grupo de objetos, llamados **elementos**.



Listas

Datos organizados secuencialmente.

Por ejemplo: nombre y legajo de los alumnos presentes, ordenados según cómo están sentados, de izquierda a derecha (borrados si se van y actualizando las posiciones si se cambian de lugar).

Listas

Tipo List<E> en Java.

Para crear una lista (existen varias implementaciones):

- List <String> alumnos = new LinkedList();
- List <String> alumnos = new ArrayList();

La clase Lista en la API de Java presenta distintas implementaciones:

- **LinkedList** (Lista doblemente enlazada).
- **ArrayList** (Lista sobre arreglos).

List: Implementaciones

- **ArrayList:** Implementada sobre un array de tamaño variable. Presenta problemas de eficiencia para agregar o borrar elementos en posiciones intermedias (ya veremos por qué). ArrayList es muy eficiente cuando se pide el valor almacenado en una posición arbitraria.

- **LinkedList:** Implementada sobre una lista enlazada. Resuelve algunos de los problemas de eficiencia para agregar o borrar de ArrayList, pero es ineficiente al momento de pedir el valor almacenado en una posición arbitraria.

Algunas Operaciones Sobre Listas

- **Agregar** un elemento:

boolean *add*(int index, E element);
boolean *add*(E element);

- **Borrar** un elemento:

boolean *remove*(int index);
boolean *remove*(E element);

- **Obtener** un elemento:

E *get*(int index);

- ¿**Contiene** un elemento dado?

boolean *contains*(E element);

Iteradores

Se puede recorrer una lista usando un iterador. La interfaz básica Iterator permite recorrer hacia adelante una colección. El orden de una iteración sobre una lista es hacia adelante a través de los elementos.

Uso de Iteradores:

```
List <Estudiante> list = new ArrayList <Estudiante> ();
// se agregan varios elementos...

Iterator itr = list.iterator();
while(itr.hasNext()){
        System.out.println(itr.next();
}
```

Iterando con forEach

```
List<String> lista= new ArrayList <String> ();
lista.add("hola");
lista.add("que");
lista.add("tal");
lista.add("estas");
//con un for tradicional
for (int i=0; i<lista.size();i++{
        System.out.println(lista.get(i));
}
//con forEach
for(String s :lista) {
        System.out.println(s);
}
```