TEMA 6. Estructuras de datos avanzadas y POO

Estructuras de datos avanzadas y POO. Colecciones en Java.

- No reinventar la rueda una y otra vez.
- Librerías en el propio lenguaje.
- Librerías de terceros.
  - Guava de Google.
  - Apache Commons Collections de fundación Apache.
- Interfaces sobre las que se define clases concretas.
- Modificación de las estructuras básicas vistas anteriormentes

Collections2	Utility methods related to all collections
Lists	Utility methods related to lists
Sets	Utility methods related to sets
Queues	Utility methods related to queues
Multisets, Multimaps	Utility methods related to multiset/map
Tables	Utility methods related to tables
Iterables	Utility methods related to collections and for-each
Iterators	Utility methods related to iterators and iterations
Ordering	Easy to create comparable and comparator orders
Immutable	Collections that cannot be modified

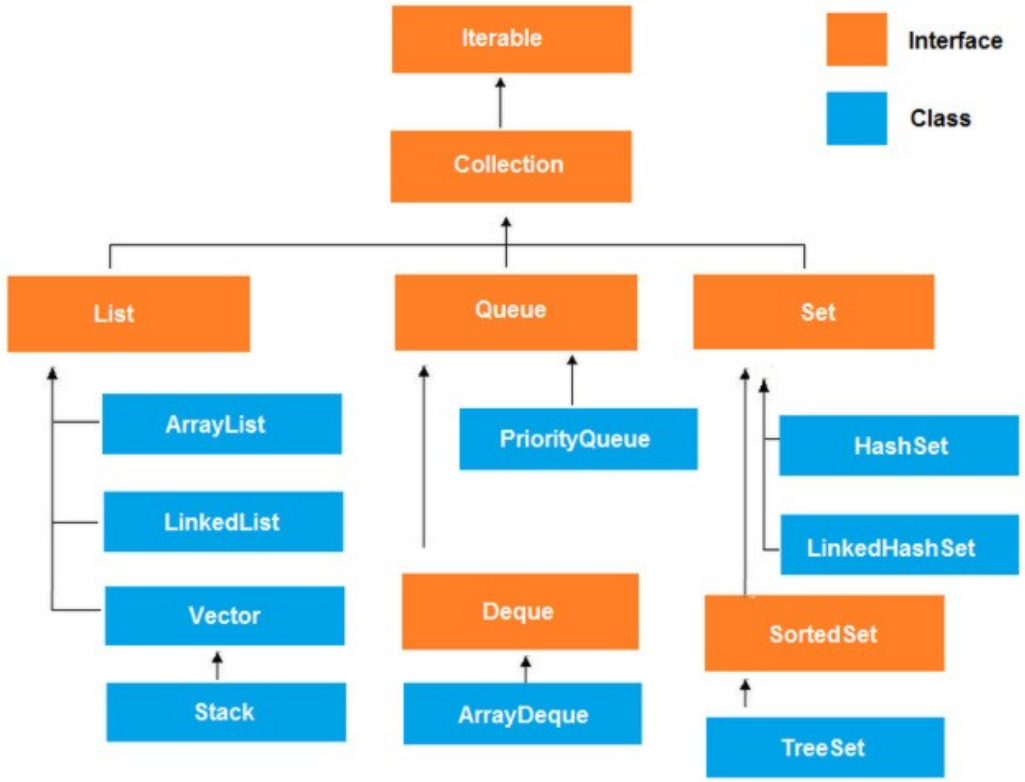
```
IterableMap map = new HashedMap();
MapIterator it = map.mapIterator();
while (it.hasNext()) {
  Object key = it.next();
  Object value = it.getValue();

  it.setValue(newValue);
}
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

Jerarquía de clases, interfaces.

- Versión reducida.
- Interfaces: Iterable, Collection, List, Queue, Set…
- Collections:

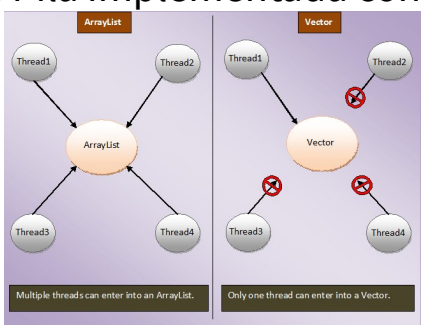
```
boolean add(E e)
void    clear()
boolean isEmpty()
boolean remove(Object o)
default Stream<E>    parallelStream()
int size()
Object[]    toArray()
```

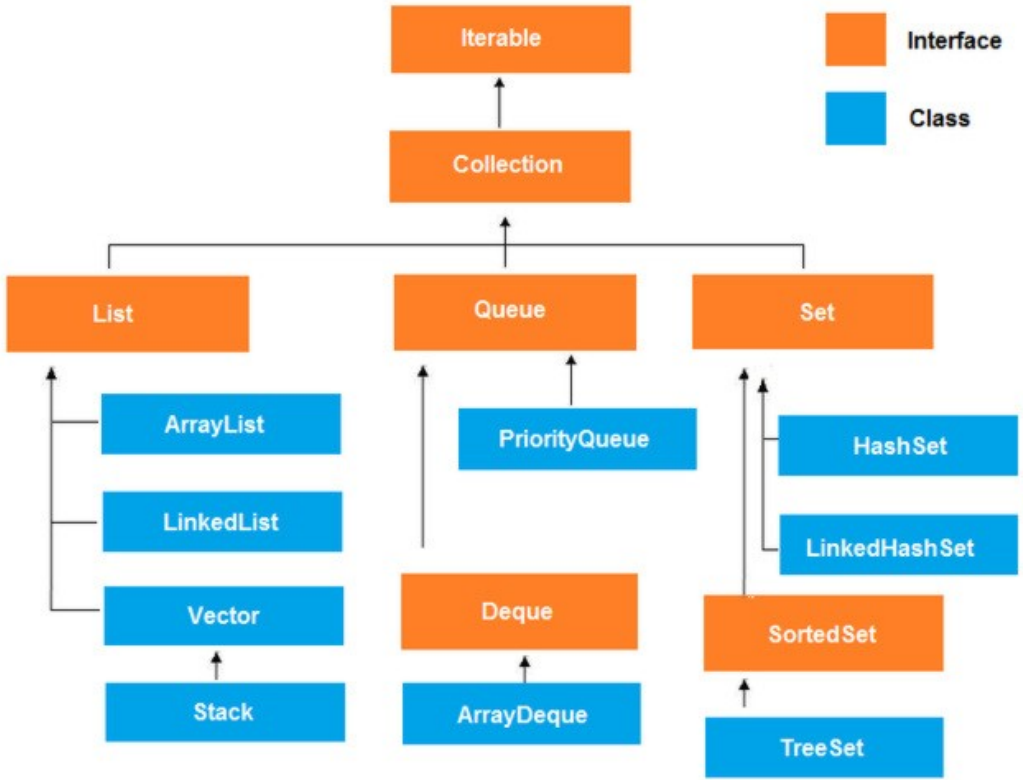


Estructuras de datos avanzadas y POO. Colecciones en Java.

Jerarquía de clases, interfaces.

- **List**. Interfaz. Lista. Internamente con otras estructuras.
  - ArrayList. Vectores.
  - LinkedList. Lista enlazada.
  - Vector. Similar a arraylist pero acceso sincronizadc (hilos).
    - Stack. Pila implementada con vector.

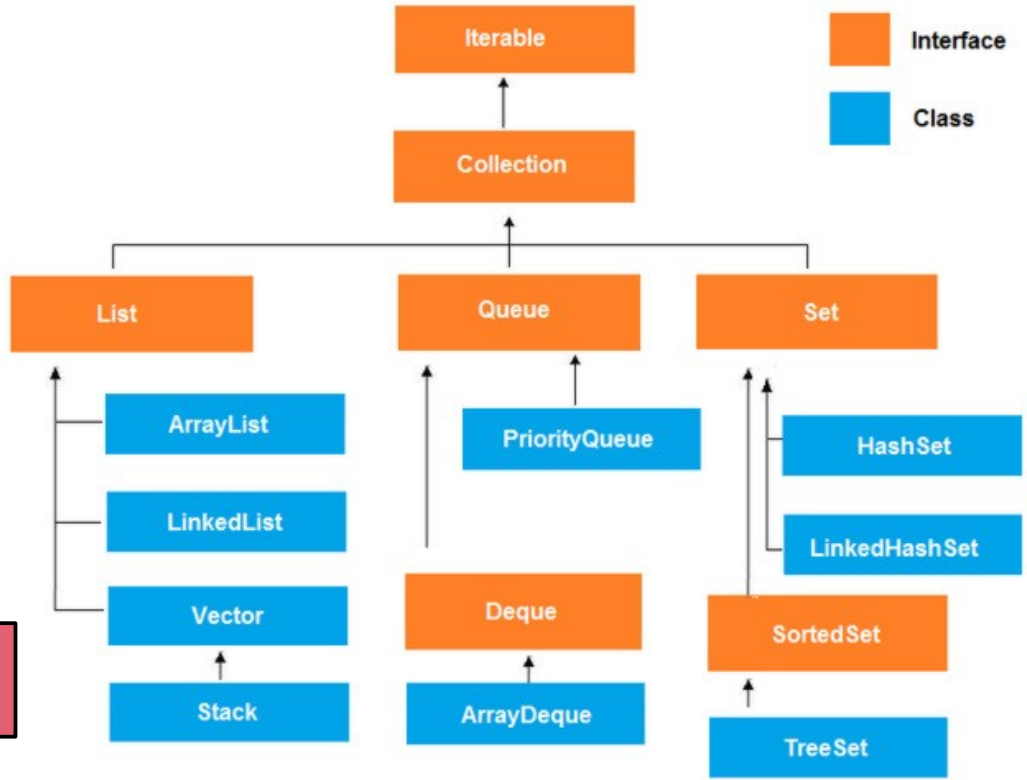




Estructuras de datos avanzadas y POO. Colecciones en Java.

Jerarquía de clases, interfaces.

- **Queue.** Interfaz. Cola.
  - Gran variedad, con características propias.
    - PriorityQueue.
    - Deque. Interfaz .Cola con dos extremos.
      - ArrayDeque.
      - ConcurrentLinkedDeque.
      - LinkedBlockingDeque. Limitada en tamaño.




```
PriorityQueue<String> queue=new PriorityQueue<String>();
queue.add("Amit");
queue.add("Vijay");
queue.add("Karan");
Iterator itr=queue.iterator();
while(itr.hasNext()){
      System.out.println(itr.next());
}
```

¿Qué palabra tiene mayor prioriodad?
¿Como cambiar la prioridad?

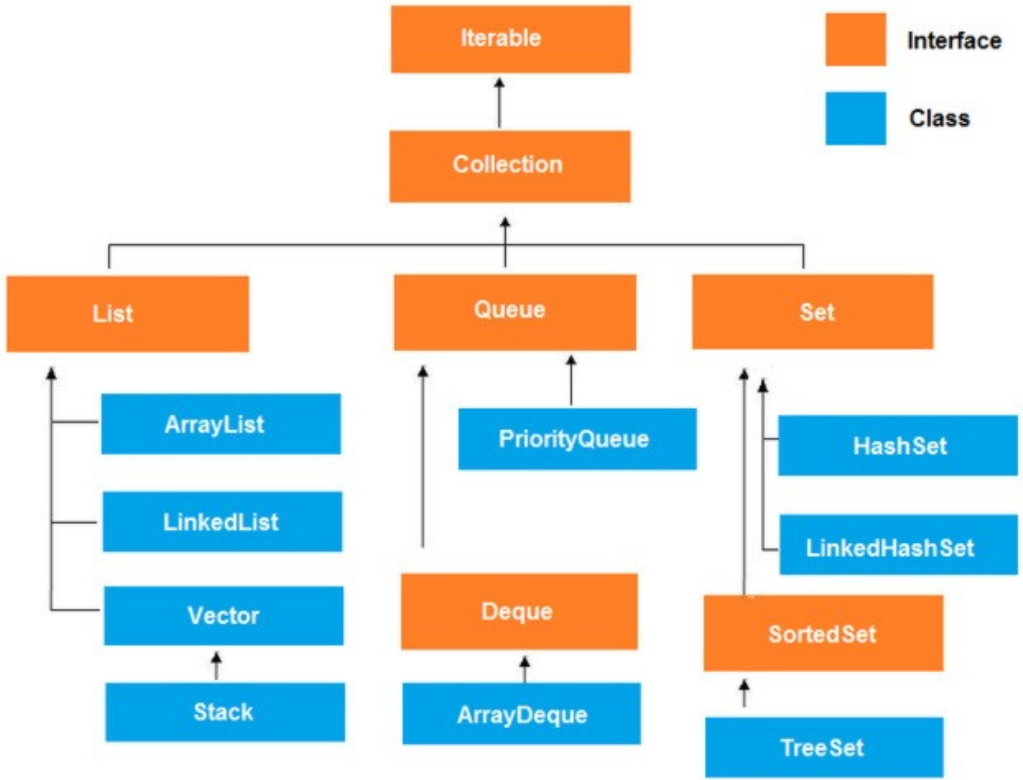
Estructuras de datos avanzadas y POO. Colecciones en Java.

Jerarquía de clases, interfaces.

- **Set.** Interfaz. Conjunto de elementos desordenados **que no pueden tener duplicados.**
- Conjunto matemático.
- Implementaciones:
  - HashSet. Tabla hash.
  - TreeSet. Árbol rojo-negro.
  - EnumSet. Enumerados. Internamente vector bits.

Existen más tipos de colecciones como:

- ConcurrentSkipListSet.
- LinkedTransferQueue...



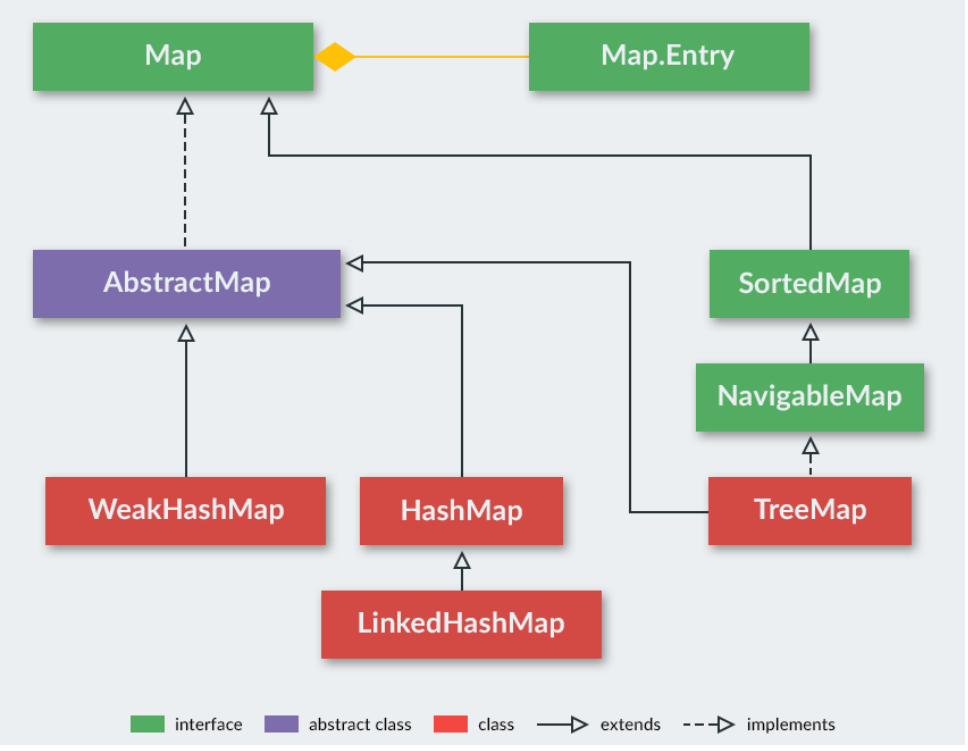
Estructuras de datos avanzadas y POO. Colecciones en Java.

Jerarquía de clases, interfaces.

- **Tabla Hash.** En Java se conoce como Map.
- Definición de:
  - Interfaces: Map, sortedmap, navigableMap.
  - Clases abstractas: AbstractMap.
  - Clases concretas.
    - WeakHashMap.
    - HashMap.
    - LinkendHashMap.
    - TreeMap.

Existen más tipos de colecciones como:

- ConcurrentSkipListSet. (Ordenado, no nulos, sincro)
- LinkedTransferQueue...(paso de mensajes)



Estructuras de datos avanzadas y POO. Colecciones en Java.

Jerarquía de clases, interfaces.

- Selección de la colección o estructura en función de las características del problema.
- Tener en cuenta las complejidades temporales en las operaciones más comunes.

Listas y conjuntos

Estructura	get	add	remove	contains
ArrayList	O(1)	O(1)	O(n)	O(n)
LinkedList	O(n)	O(1)	O(1)	O(n)
HashSet	O(1)	O(1)	O(1)	O(1)
LinkedHashSet	O(1)	O(1)	O(1)	O(1)
TreeSet	O(log n)	O(log n)	O(log n)	O(log n)

Mapas:

Estructura	get	put	remove	containsKey
HashMap	O(1)	O(1)	O(1)	O(1)
LinkedHashMap	O(1)	O(1)	O(1)	O(1)
TreeMap	O(log n)	O(log n)	O(log n)	O(log n)

Estructuras de datos avanzadas y POO. Colecciones en Java.

Jerarquía de clases, interfaces.

- **Generalización.**
  - Estructuras se comportan igual almacenen enteros, coches o naves espaciales.
  - Tener n estructuras que hacen lo mismo cambiando la clase que se almacena.
  - Solución: Generics. Se especifica en la definición de la estructura la clase de los objetos que almacenará.
    - La estructura puede almacenar objetos de la clase que se define y de los que implemente o hereden de esta clase.

```
List myIntList = new LinkedList(); // 1
myIntList.add(new Integer(0)); // 2
Integer x = (Integer)
myIntList.iterator().next(); // 3
```

```
List<Integer> myIntList = new LinkedList(); // 1'
myIntList.add(new Integer(0)); // 2'
Integer x = myIntList.iterator().next(); // 3'
```

¿Qué diferencia se observa en los dos fragmentos anteriores? ¿Qué sucede si se intenta insertar en el segundo fragmento una cadena?

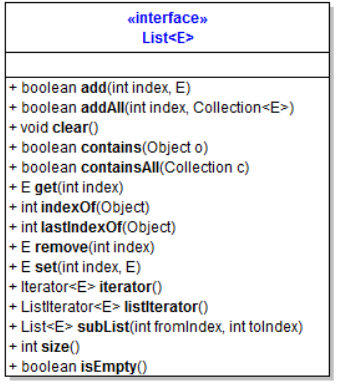
Iteradores.

- Es una interfaz.
- Permiten recorrrer las colecciones.
- Los métodos son:
  - **void foreach(Consumer <? super T> action)**: Permite recorrer la estructura sin necesidad de un for, además de almacenar el elemento en un tipo de dato concreto.
  - **Iterator<T> iterator()**: Devuelve un iterador del tipo indicado.
  - **default Spliterator<T> spliterator()**. Similiar al anterior, pero con la característica de que pueden procesarse en paralelo. Pudiendo obtener información del proceso de procesado, por ejemplo cuantos quedan por procesar un conjunto de imágenes (cambiar el tamaño) en la que cada tarea es independiente.

Estructuras de datos avanzadas y POO. Colecciones en Java.

List.

- Interfaz. **Impone un orden y puede tener duplicados.**
- Operaciones:
  - Acceso por posición: get, set, add, addAll, remove.
  - Búsquedas: indexOf y lastIndexOf.
  - Iteración.
  - Rango, extraer sublistas con sublist.



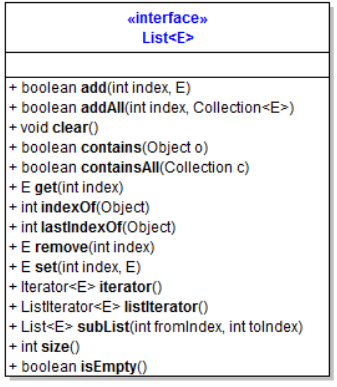
Clase estática **Collections** posee algoritmos y operaciones comunes (también para Set y Queue).

- Sort. (Ordena)
- Suffle. (Permuta aleatoriamente)
- Swap. (Intercambia).
- Replace. (Reemplaza tods las apariciones de un valor).
- Fill. Sobreescribe ciertos elementos.
- Binarysearch. Búsqueda en colección **ordenada.**
- IndexOfSublist.
- LastIndexOfSublist.

Estructuras de datos avanzadas y POO. Colecciones en Java.

List.

- Interfaz. **Impone un orden y puede tener duplicados.**
- Operaciones:
  - Acceso por posición: get, set, add, addAll, remove.
  - Búsquedas: indexOf y lastIndexOf.
  - Iteración.
  - Rango, extraer sublistas con sublist.



Clase estática **Collections** posee algoritmos y operaciones comunes (también para Set y Queue).

- Sort. (Ordena)
- Suffle. (Permuta aleatoriamente)
- Swap. (Intercambia).
- Replace. (Reemplaza tods las apariciones de un valor).
- Fill. Sobreescribe ciertos elementos.
- Binarysearch. Búsqueda en colección **ordenada.**
- IndexOfSublist.
- LastIndexOfSublist.

https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html

Estructuras de datos avanzadas y POO. Colecciones en Java.

List.

- Las clases que implementan la interfaz:
  - **ArrayList**. Se implementa con un array, en caso de quedarse sin espacio es necesario redimensionar el array, es posible indicar la capacidad del array en uno de los constructores personalizados. **Una de las colecciones más utilizadas.**
  - **CopyOnWriteArrayList**. Utilizada para procesos concurrentes, en el que se pueden tomar insantaneas de la lista en cada momento de forma que no se vea afecta por otras operaciones. Por ejemplo se obtiene un iterador y a continuación se borran elementos. En un arrayList, estos cambios afectan al iterador, en caso de CopyOnWriteArrayList el iterador se encuentra apuntando a la instantanea y no se ve afectado por los cambios.
  - **LinkedList**. Se implementa con referencias doblemente enlazadas. No es una implementanción sincronizada.
  - **Stack.** La implementación de una típica pila, internamente es una extensión de la clase vector, posee las operaciones clásicas de push y pop.
  - **Vecto**r. Similar al ArrayList, pero con sincronización al acceder de forma concurrente, siendo por tanto más segura pero más lenta.

List.

- Ejemplo:

```
public class Alumno implements Comparable<Object> {
    private String nombre;
    private String apellidos;
    private String curso;
    public Alumno() {}
    public Alumno(String nombre, String apellidos, String curso) {
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.curso = curso;
    }
    @Override
    public int compareTo(Object o) {
        if (!(o instanceof Alumno)) {
            throw new UnsupportedOperationException("Not supported yet.");
        }
        return (this.nombre.compareTo(((Alumno) o).apellidos));
    }...
```

List.

- Ejemplo:

```
long inicio, fin;
long tiempo;
int tam = 1500000;
List<Alumno> alumnos = new ArrayList<Alumno>();
for (int index = 0; index < tam; index++) {
    alumnos.add(new Alumno("Paco" + Math.random() * 1000000, "Uno" + index, "1DAW" + index));
}
Collections.sort(alumnos);
alumnos = new Vector<Alumno>();
for (int index = 0; index < tam; index++) {
     alumnos.add(new Alumno("Paco" + Math.random() * 1000000, "Uno" + index, "1DAW" + index));
}
alumnos = new LinkedList<Alumno>();
for (int index = 0; index < tam; index++) {
     alumnos.add(new Alumno("Paco" + Math.random() * 1000000, "Uno" + index, "1DAW" + index));
}
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

List.

- Ejemplo:
  Al iterar y ordenar las listas anteriores se obtienen los siguientes resultados:

un arraylist tarda en hacer las operaciones 1058 milisegundos
un vector tarda en hacer las operaciones 1040 milisegundos
un lista doblemente enlazada tarda en hacer las operaciones 1541 milisegundos

En el caso del vector es muy simliar al arraylist ya que no existen otros hilos accediendo concurrentemente, pero en el caso de la lista doblemente enlazada es significativamente mayor. ¿ofrece alguna ventaja?

Ya sea el ArrayList, el vector o la lista doblemente enlazada el tipo de la variable que los referencia es de tipo List ¿Cómo es posible? ¿Qué principio de la POO se utiliza?

Queue.

- Es una interfaz, con los métodos:

```
public interface Queue<E> extends Collection<E> {
   E element();
   boolean offer(E e);
   E peek();
   E poll();
   E remove();
}
```

Hereda de Collection, por tanto los que implementes Queue han de implementar los métodos de Collection.
Existen multitud de colas cada una con unas características y uso concreto.

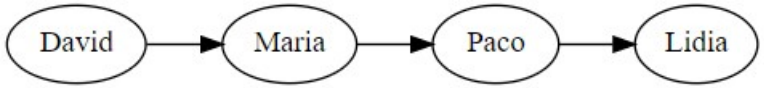
AbstractQueue, ArrayBlockingQueue, ArrayDeque, ConcurrentLinkedDeque, ConcurrentLinkedQueue, DelayQueue, LinkedBlockingDeque, LinkedBlockingQueue, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, SynchronousQueue

Queue.

- Ejemplo. Cola con prioridad:

```
public class Person implements
Comparable {
   private String name;
   private int age;
   public Person(String name, int age) {
      this.name = name;
      this.age = age;
   }
   @Override
   public int compareTo(Object o) {
      if (o instanceof Person) {
         return -1*( this.getAge() - ((Person)
o).getAge());
```

```
public class Principal {
  public static void main(String args[]) {
    PriorityQueue<Person> cola;
    Person last = null;
    cola = new PriorityQueue<Person>();
    cola.add(new Person("Lidia",19));
    cola.add(new Person("Paco", 20));
    cola.add(new Person("David", 76));
    cola.add(new Person("Maria", 22));
  }
}
```



Estructuras de datos avanzadas y POO. Colecciones en Java.

Dequeue.

- Interfaz. Cola doblemente enlazada, inserción y borrado en los 2 extremos.

	First Element (Head)		Last Element (Tail)	
	Throws exception	*Special value*	*Throws exception*	*Special value*
Insert	`addFirst(e)`	`offerFirst(e)`	`addLast(e)`	`offerLast(e)`
Remove	`removeFirst()`	`pollFirst()`	`removeLast()`	`pollLast()`
Examine	`getFirst()`	`peekFirst()`	`getLast()`	`peekLast()`

Esta interfaz es implementada por las clases:
ArrayDeque, ConcurrentLinkedDeque, LinkedBlockingDeque, LinkedList

Estructuras de datos avanzadas y POO. Colecciones en Java.

Set.

- Interfaz. Modela **conjuntos** matemáticos.
- No puede tener elementos duplicados.
- Operaciones con conjuntos

Methods

Modifier and Type	Method and Description
boolean	**add**(**E** e) Adds the specified element to this set if it is not already present (optional operation).
boolean	**addAll**(**Collection**<? extends **E**> c) Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
void	**clear**() Removes all of the elements from this set (optional operation).
boolean	**contains**(**Object** o) Returns `true` if this set contains the specified element.
boolean	**containsAll**(**Collection**<?> c) Returns `true` if this set contains all of the elements of the specified collection.
boolean	**equals**(**Object** o) Compares the specified object with this set for equality.
int	**hashCode**() Returns the hash code value for this set.
boolean	**isEmpty**() Returns `true` if this set contains no elements.
Iterator<**E**>	**iterator**() Returns an iterator over the elements in this set.
boolean	**remove**(**Object** o) Removes the specified element from this set if it is present (optional operation).
boolean	**removeAll**(**Collection**<?> c) Removes from this set all of its elements that are contained in the specified collection (optional operation).
boolean	**retainAll**(**Collection**<?> c) Retains only the elements in this set that are contained in the specified collection (optional operation).
int	**size**() Returns the number of elements in this set (its cardinality).
Object[]	**toArray**() Returns an array containing all of the elements in this set.
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the specified array.

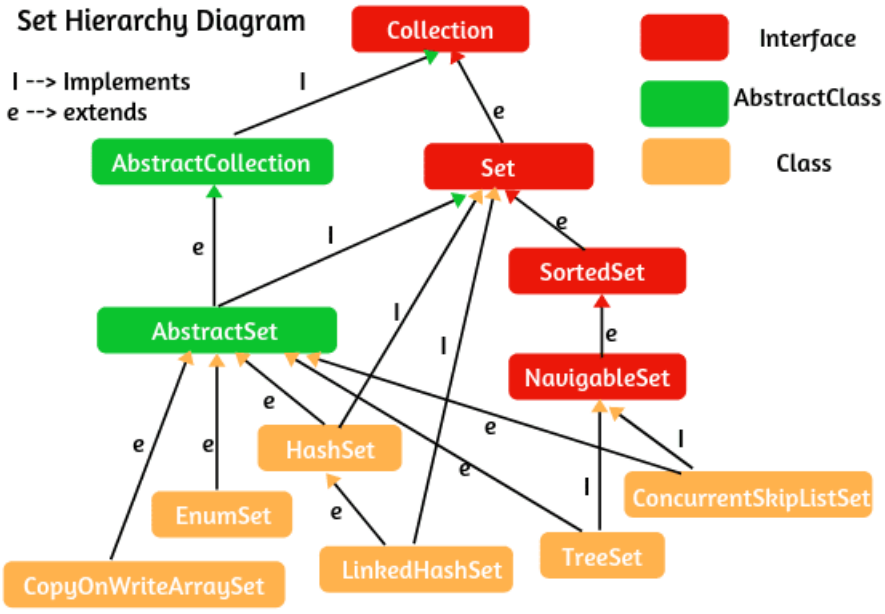
Estructuras de datos avanzadas y POO. Colecciones en Java.

Set.

SortedSet: Ordenado, usando el orden natural o implementando compareTo, siendo el orden ascendente. Define los métodos **comparator, first, headSet, last, subSet y tailSet.**

NavigableSet: Amplia a SorteSet, define los métodos:

- E ceiling(E e): Devuele el elemento menor de los mayores del elemento que se pasa.
- E floor(E e): El mayor de los menores.
- SortedSet<E> headSet(E toElement): Devuelve una vista de la parte de este conjunto cuyos elementos son estrictamente menores que toElement.
- E higher(E e): Devuelve el elemento mínimo de este conjunto estrictamente mayor que el elemento dado, o nulo si no existe tal E pollFirst(): Recupera y elimina el primer elemento (el más bajo), o devuelve un valor nulo si este conjunto está vacío.

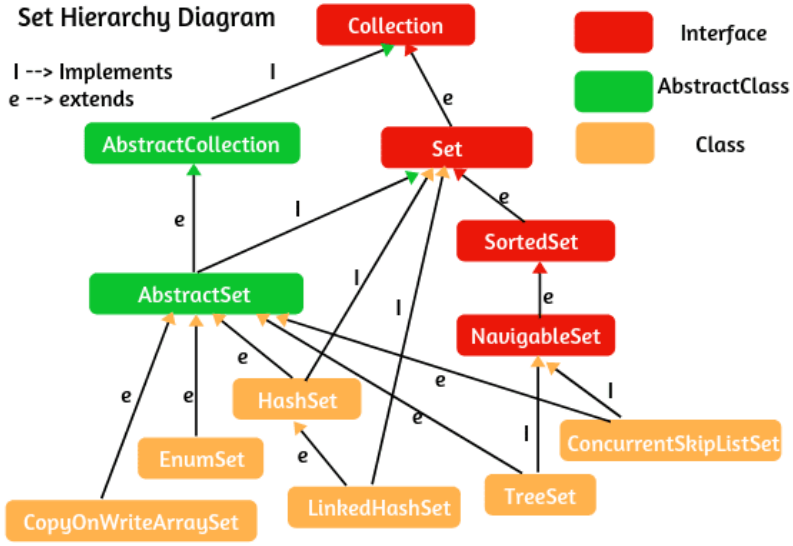


Estructuras de datos avanzadas y POO. Colecciones en Java.

Set.

Implementaciones:

- **HashSet**. Se almacena el conjunto con una tabla hash, no se garantiza el orden de la iteración.
- **TreeSet**. En este caso la estructura interna es un árbol rojo-negro, más lento que HashSet, implementa la interfaz NavigableSet.
- **LinkedHashSet**. En este caso se usa una tabla hash que contiene una lista enlazada en la que se almacena los elementos con colisión, en las listas internas se ordena por orden de llegada.
- Otras menos usadas: ConcurrentSkipListSet, EnumSet, CopyOnWriteArrayListSet, ConcurrentSkipListSet.



Estructuras de datos avanzadas y POO. Colecciones en Java.

Map.

- No es una colección como tal.
- En otros lenguajes se conoce como diccionario, tabla hash…
- Se define una interfaz Map<Key,Value> sobre la que se implementa la jerarquía
- En la práctica es una más.
- **Se accede a los elementos por clave, que puede ser un entero, una cadena, o un objeto complejo entre otros.**

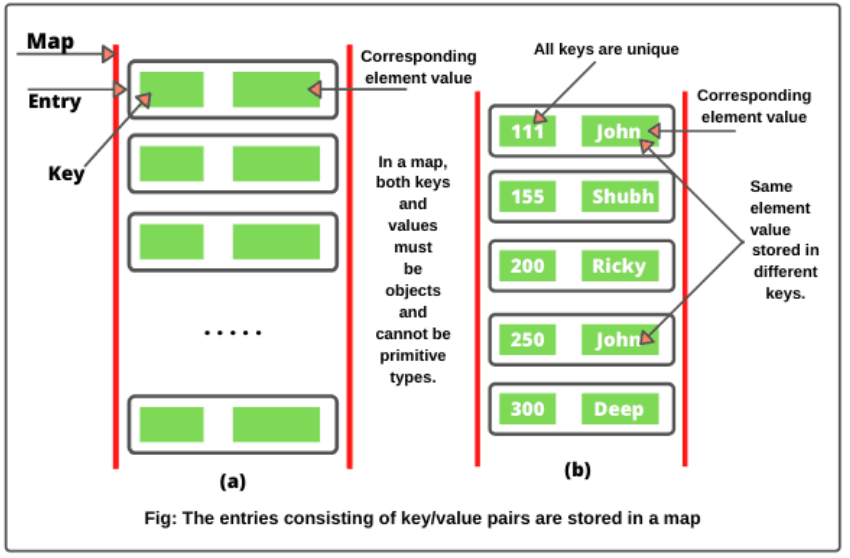


Fig: The entries consisting of key/value pairs are stored in a map

Estructuras de datos avanzadas y POO. Colecciones en Java.

Map.

- Se transforma la clave a un entero con una función hash que indica el índice del elemento.
- Dependiendo de la implementación se puede o no garantizar el orden (árbol sí, tabla hash no).
- Al igual con la existencia de claves nulas, depende de la implementación.
- Algunos métodos de la interfaz:
  - void clear().
  - boolean containsValue(Object value).
  - V get(Object key).
  - V put(K key, V value).
  - V remove(Object key). Elimina el mapeo de una clave de este mapa si está presente.

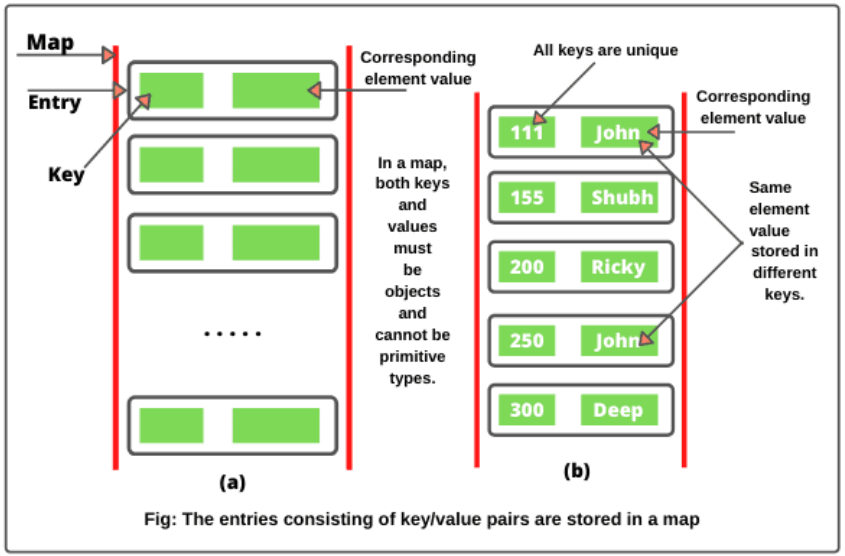


Fig: The entries consisting of key/value pairs are stored in a map

Map.

- A partir de esta interfaz se define una jerarquía de clases compleja:
  - Se definen a partir de esta las interfaces:
    - Bindings.
    - ConcurrentMap<K,V>.
    - ConcurrentNavigableMap<K,V>.
    - LogicalMessageContext.
    - MessageContext.
    - NavigableMap<K,V>.
    - SOAPMessageContext, SortedMap<K,V>
- Sobre estas interfaces a su vez se establece una serie de clases concretas.
  - ConcurrentSkipListMap.
  - **EnumMap.**
  - **HashMap.**
  - **Hashtable.**
  - IdentityHashMap.
  - **LinkedHashMap.**
  - SimpleBindings.
  - TabularDataSupport.
  - **TreeMap.**

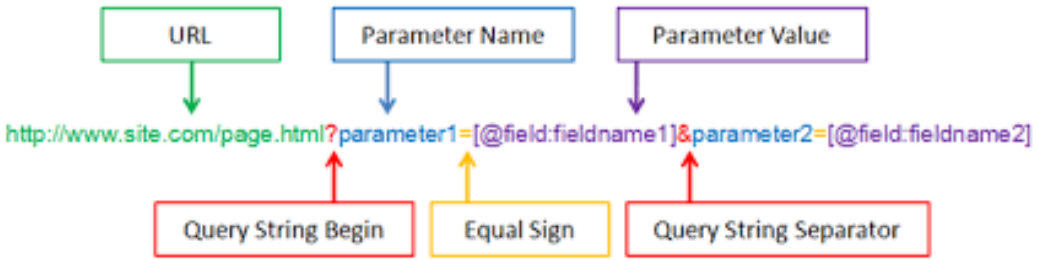
  En los apuntes se encuentran el resto de clases que implementan la interfaz Map

Map.

- **HashMap**. Utiliza una tabla de dispersión para almacenar la información del mapa. Las operaciones básicas (get y put) se harán en tiempo constante siempre que se dispersen adecuadamente los elementos. Es coste de la iteración dependerá del número de entradas de la tabla y del número de elementos del mapa. No se garantiza que se respete el orden de las claves.
- **TreeMap**. Utiliza un árbol rojo-negro para implementar el mapa. El coste de las operaciones básicas será logarítmico con el número de elementos del mapa O(log n). En este caso los elementos se encontrarán ordenados por orden ascendente de clave.
- **Hashtable.** Similar a HashMap. Acceso sincronizado, no se permitirán claves nulas (null). Este objeto extiend e la obsoleta clase Dictionary, ya que viene de versiones más antiguas de JDK.
- **EnumMap.** La clave ha de ser de un tipo de enumerado concreto. No permitiendo claves nulas y no se encuentra sincronizado.

Map.

- Ejemplo:
  - Al realizar una petición HTTP, los parámetros se encuentran en la “querystring” por GET o en el cuerpo por POST (o combinaciones de ambas).
  - Los lenguajes y frameworks web utilizan tablas Hash o Map para ofrecer la información de forma sencilla al desarrollador.



Map.

- Ejemplo:

```
public class Principal {
    public static void main(String[] args) {
        String cadena = "https://example.com/path/to/page?name=ferret&color=purple";
        HashMap<String, String> parameters = new HashMap<String, String>();
        //se obtiene la parte del querystring
        String querystring = cadena.substring(cadena.indexOf('?') + 1);
        //se trocean los parametros
        StringTokenizer st = new StringTokenizer(querystring, "&");
        //temporales necesarios
        String parametro, clave, valor;
        //se recorren los trozos y se inserta en un HashMap
        while (st.hasMoreElements()) {
```

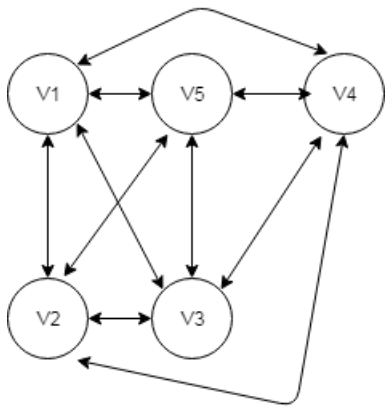
```
Clave:color valorpurple
Clave:name valorferret
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

Otras estructuras. Grafos.

- Basado en vértices (nodos) y aristas(uniones).
- Java no posee de forma nativa grafos, se ha de recurrir a liberías externas.
- Diferentes representaciones, matrices, matrices dispersas, listas de adyacencia.
- Los árboles son una particularización de los grafos.
- Problemas complejos como algoritmo del viajante, o construcción de tablas de enrutamiento.

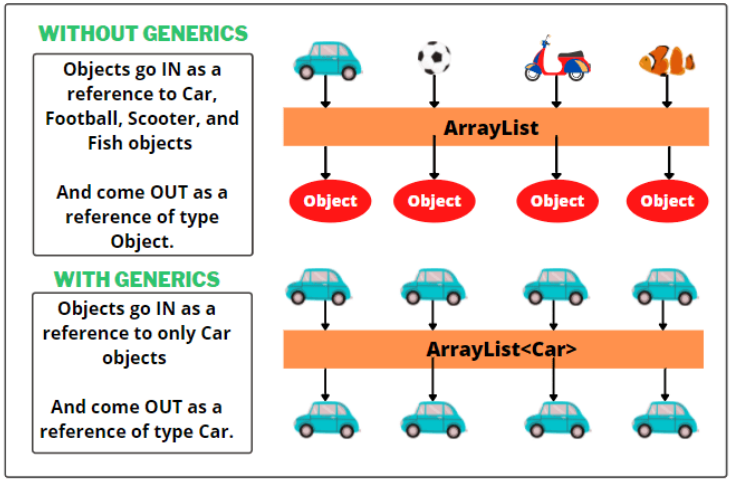
JGraphT



Estructuras de datos avanzadas y POO. Colecciones en Java.

Generalización.

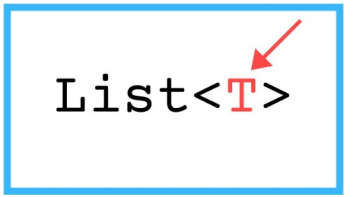
- Por defecto las colecciones almacenan objetos de tipo “Object”.
- **Permite establecer resticcciones a nivel de tipo haciendo que ciertas clases, intefaces o métodos acepten únicaente los tipos estipulados.**
- Se denomina Generics.
- Posible aplicar a clases creadas a posteriori.
- Beneficios:
  - Comprobación de tipos más fuerte en tiempo de compilación.
  - Eliminación de casts aumentando la legibilidad del código.
  - Posibilidad de implementar algoritmos genéricos, con tipado seguro.



Estructuras de datos avanzadas y POO. Colecciones en Java.

Generalización.

- Desarrollo de clases genéricas.
- Indicar argumentos “genéricos” que puede ser cualquier clase.
- Se indicará al definir la clase genérica los tipos “concretos”.
- Lista de parámetros después del nombre de la clase <T1, T2, T3….>
- Conveciones de nombres para parámetros:
  - E: elemento de una colección.
  - K: clave.
  - N: número.
  - T: tipo.
  - V: valor.
  - S, U, V etc: para segundos, terceros y cuartos tipos.



Generic Term	Meaning
`Set<E>`	Generic Type , E is called formal parameter
`Set<Integer>`	Parametrized type , `Integer` is actual parameter here
`<T extends Comparable>`	Bounded type parameter
`<T super Comparable>`	Bounded type parameter
`Set<?>`	Unbounded wildcard
`<? extends T>`	Bounded wildcard type
`<? Super T>`	Bounded wildcards
Set	Raw type
<T extends Comparable<T>>	Recursive type bound

T – used to denote the type
E – used to denote an element
K – keys
V - values
N – for numbers

Estructuras de datos avanzadas y POO. Colecciones en Java.

Generalización.

- Ejemplo: Estructura bolsa para almacenar objetos concretos, internamente una lista

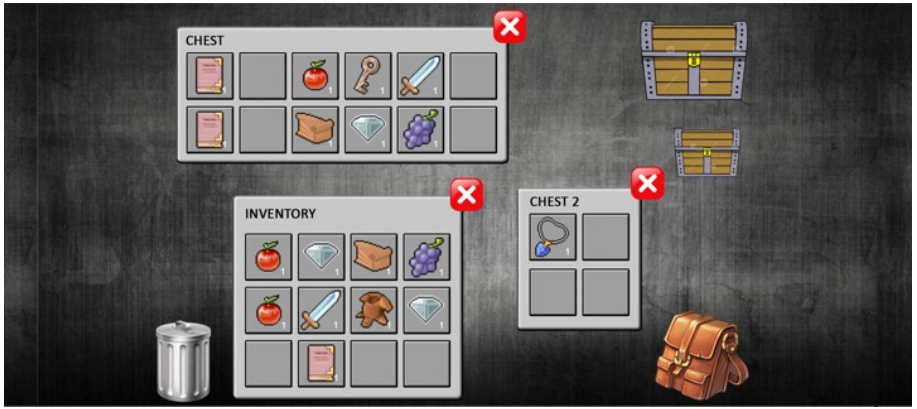
```
public class Bolsa<T> implements Iterable<T>{

	private ArrayList<T> lista= new ArrayList<T>();
	private int tope;
	public Bolsa(int tope) {
		super();
		this.tope = tope;
	}
	public void add(T objeto ) {
		if (lista.size()<=tope) {
			lista.add(objeto);
		}else{
			throw new RuntimeException("bolsa llena");
		}
	}
	public Iterator<T> iterator() {
		return lista.iterator();
	}
}
```

```
Bolsa<Chocolatina> bolsa= new Bolsa<Chocolatina>();
Chocolatina c= new Chocolatina("milka");
bolsa.add(c);
Chocolatina c2= new Chocolatina("ferrero");
bolsa.add(c2);
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

Generalización.

- Ejercicio : Estructura de datos para gestionar inventarios de juego, cada personaje posee armas, vestuario, comida….
  - AddElement
  - GetElement
  - JoinInventory
  - Order
  - GetElementByName
  - RemoveElementByName
  - RemoveElementByIndex



Además se han de crear la clase base Arma, y dos hijas: Pistola y Rifle y la clase base Vestuario, con dos clases hijas: Coraza y CotadeMaya.