



# *ARREGLOS Y COLECCIONES*

## *Capítulo 7*

*Milton Labanda*

# ***Arreglos: Declaracion e Inicialización***

---

- Un arreglo contiene varios valores (primitivos u objetos) del mismo tipo
- Los Arreglos necesitan ser declarados:
  - Especificando un tamaño fijo:
    - La longitud es fijada cuando el array es creado
    - Debe ser especificado por una expresión entera
  - Opcionalmente pueden ser inicializados
  - Sus elementos tienen valores por defecto dependiendo del tipo del arreglo.
  - El primer elemento siempre será el elemento cero , ej. array[0]
- Examples:

```
int MAX = 5;
boolean bit[] = new boolean[MAX];
float[] value = new float[2*3];
int[] number = {10, 9, 8, 7, 6};
System.out.println(bit[0]); // imprime "false"
System.out.println(value[3]); // imprime "0.0"
System.out.println(number[1]); // imprime "9"
```

# Clases Genéricas

- Permiten implementar el Polimorfismo paramétrico (parametrizar Tipos)
- Permiten al compilador asegurar la validez de los tipos en tiempo de compilación.
- Evitan el uso de casting en muchas ocasiones

```
//Ejemplo de Clase genérica
class Cosa<T>{

    T nombre;

    public Cosa( T nombre){
        this.nombre = nombre;
    }

    public void setNombre(T nom){
        this.nombre = nom;
    }

    public T getNombre(){
        return nombre;
    }
}

...
//Uso de clase genérica
Cosa <String> c = null;
c = new Cosa<String>("piedra");
String nombre = c.getNombre();
c.setNombre(
    new Float(1.4)); //Error
```

# Qué es una Colección?

---

• **Una colección es un objeto contenedor que agrupa múltiples elementos de un tipo determinado en sólo una unidad y proporciona una forma particular de organizarlos.**

• Las colecciones típicamente agrupan items que forman una agrupación natural, así por ejemplo:

- Un juego de pocker
  - Una colección of cartas
- Una carta del correo
  - Una colección letras
- Un directorio telefónico
  - Una colección de pares nombre-número telefónico

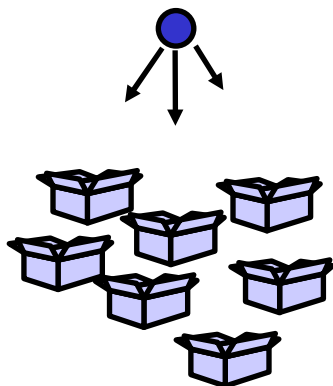


# Representan Estructuras de Datos

arreglo-vector

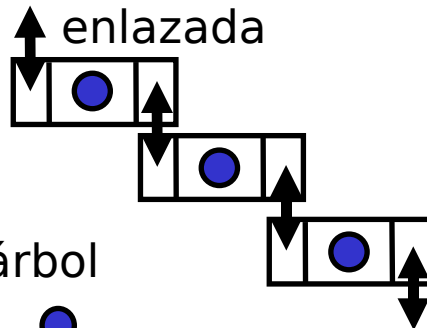


mapa - hash

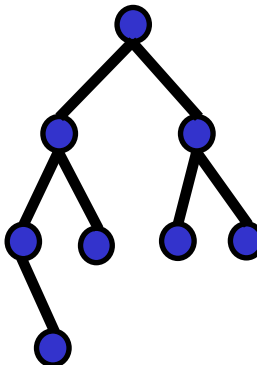


lista

enlazada



árbol



# ***Tipos específicos de Colecciones***

---

- Set
  - No pueden contener elementos duplicados
    - e.j., empleados, libros de una biblioteca, procesos corriendo en una máquina
- List
  - Colección ordenada, puede contener duplicados
    - e.j., historial de páginas web, registro de estudiantes
- Map
  - Objetos que mapean llaves (keys) a valores, no se permiten llaves duplicadas.
    - e.j., diccionario, atributos de una clase

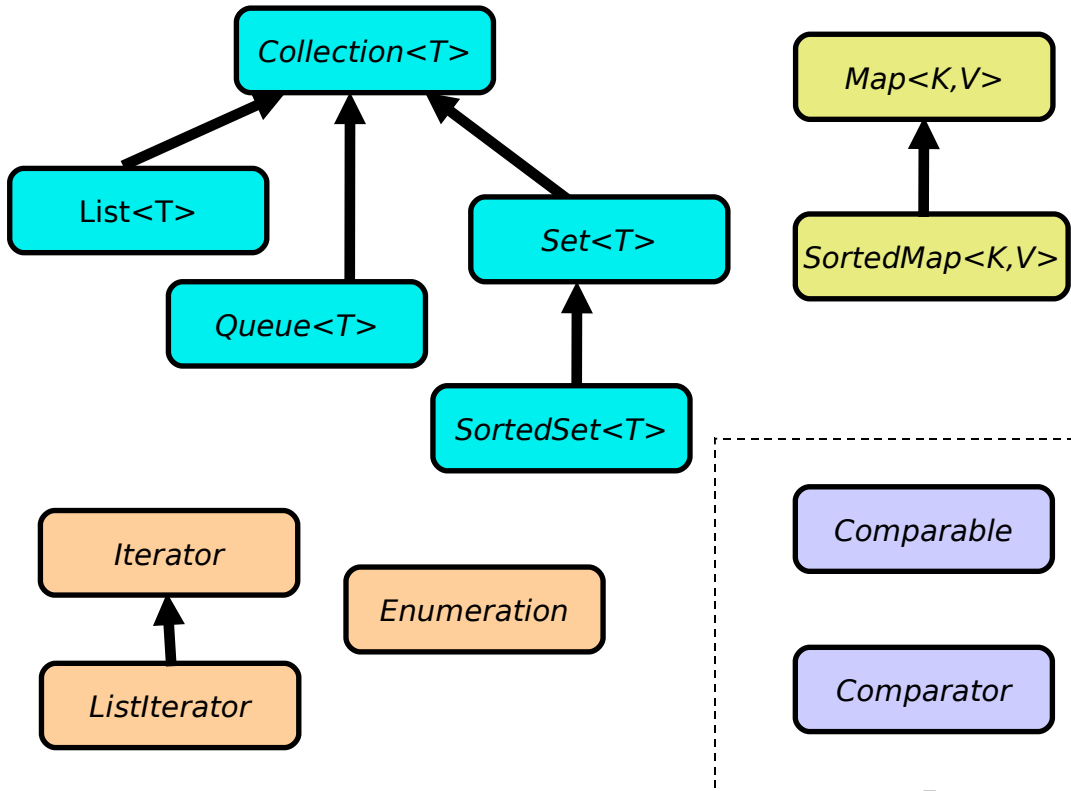
Los Arreglos son considerados también colecciones, aunque ellos no son parte del Framework de Colecciones

# ***El Framework de Colecciones de Java***

---

- El Framework de Colecciones es una arquitectura unificada de elementos para representar y manipular colecciones:
- Dicho Framework se compone de 3 cosas:
  - **Interfaces:** Tipos de datos abstractos que representan colecciones.
    - Permiten manipular colecciones independientemente de los detalles de sus representaciones
    - Pueden crearse nuevos tipos de colecciones, proveyendo de diferentes implementaciones del mismo método.
  - **Implementaciones:** Tipos concretos de las interfaces de Colecciones.
    - Se constituyen en estructuras de datos reusables
  - **Algoritmos:** Métodos que realizan operaciones muy útiles en los objetos que implementan las interfaces de colección.
    - e.j. búsquedas y ordenaciones

# Interfaces



# La Interface Collection

- Usada para cambiar colecciones y pasarlas de un método a otro.

Por ejemplo:

- Añadir o remover de una colección

- Definir métodos para

- Facilitar la iteración a través de colecciones

- Convertir colecciones a arreglos

## **Collection**

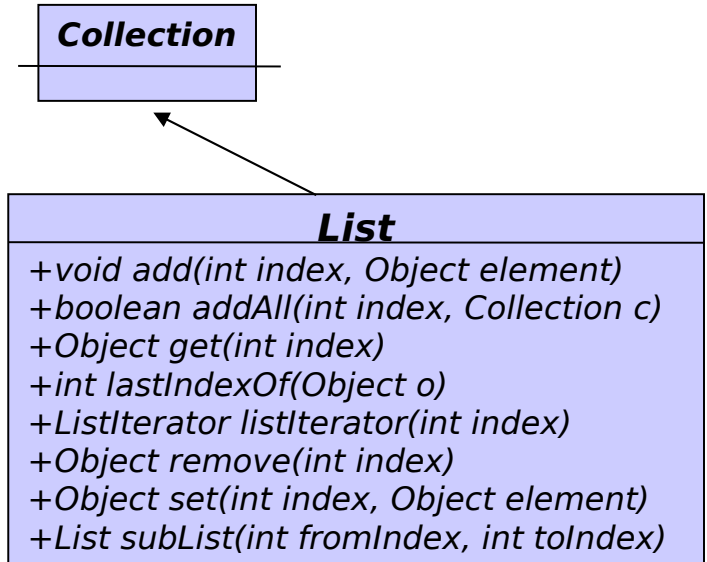
```
+boolean add(Object o)
+boolean addAll(Collection c)
+void clear()
+boolean contains(Object o)
+boolean containsAll(Collection c)
+boolean equals(Object o)
+int hashCode()
+boolean isEmpty()
+iterator iterator()
+boolean remove(Object o)
+boolean removeAll(Collection c)
+boolean retainAll(Collection c)
+int size()
+Object[] toArray()
+Object[] toArray(Object[] a)
```

# La Interface List

---

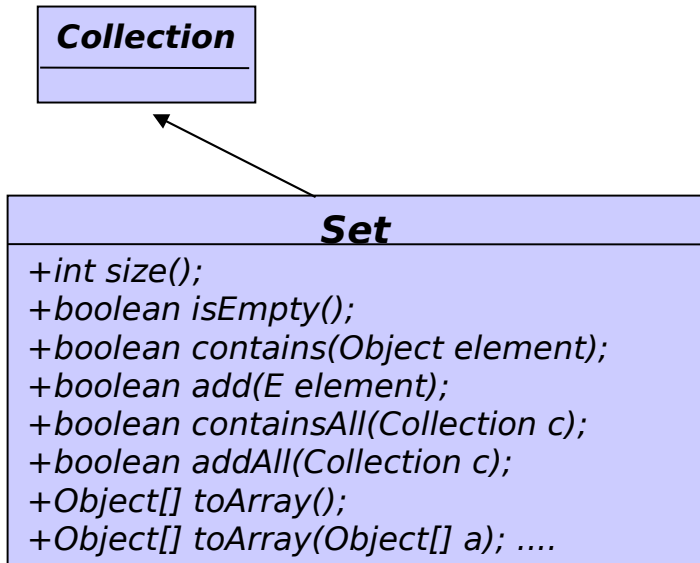
- Una Lista (List) representa un grupo ordenado de elementos:

- Se permiten elementos duplicados
- Permiten acceder a los elementos vía índices similares a los Arreglos
- Permiten realizar búsquedas
- Permiten extraer sublistas



# La Interface Set

- Un Set o conjunto representa un conjunto de elementos sin orden
- No admiten duplicados
- Su organización es similar a un bolsillo en donde se guardan elementos de distintos tipos y en cualquier orden



# La Interface Map

- Mapean claves a valores
  - No puede contener claves duplicadas
- Definen la interface necesaria para manipular los elementos tal como una colección:
  - Añadir/Eliminar un par clave-valor
  - Dada una clave, obtener el valor
- El contenido de un Mapa puede ser visualizado en una de tres **vistas de colección**:
  - Un set of claves
  - Una collection de valores
  - Un set de equivalencias claves-valores

## Map

```
void clear()
boolean containsKey(Object key)
boolean containsValue(Object value)
Set entrySet()
boolean equals(Object o)
Object get(Object key)
int hashCode()
boolean isEmpty()
Set keySet()
Object put(Object key)
void putAll(Map t)
Object remove(Object key)
int size()
Collection values()
```

# ***Comparación de Objetos***

---

- Para comparar dos objetos de una colección primeramente debe existir una forma de imponer una ***orden*** en los items

- Para cualquier par de dos items en la collection, debe ser posible comparar los objetos y determinar sin ambigüedades si:

- El Objeto A viene antes del objeto B
- El Objeto B viene antes del objeto A
- El Objeto A y el objeto B son iguales

- Existen dos formas de ordenar objetos

- La interface Comparable
- La interface Comparator

# Comparación de Objetos

---

- La Interface **Comparable**
  - Provee una **ordenación natural** para los objetos de las clases más conocidas : String, Integer, Date, etc, etc,
  - Las clases más conocidas la implementan por defecto
- The **Comparator** interface
  - Permite escribir comparadores personalizados implemantandola en una clase específica
  - Los mismos dos objetos pueden compararse de manera usando diferentes comparadores.

## Comparable

*int compareTo(Object o)*

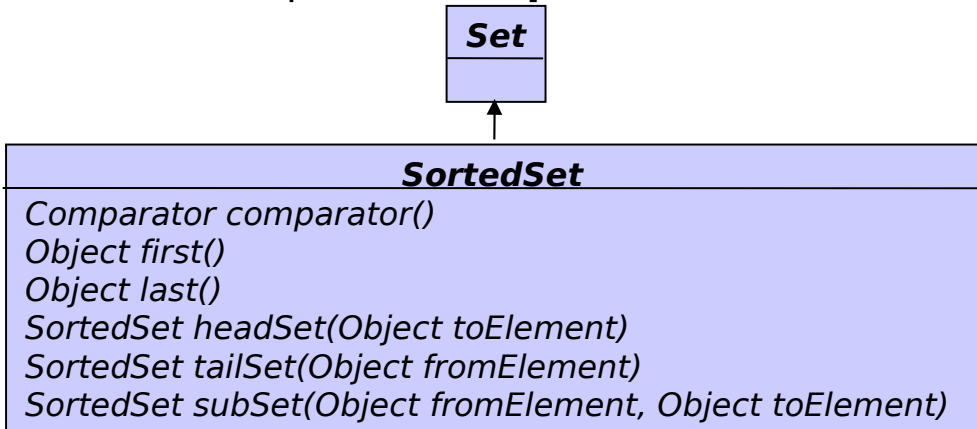
## Comparator

*int compare(Object o1, Object o2)*

# Colecciones Ordenadas

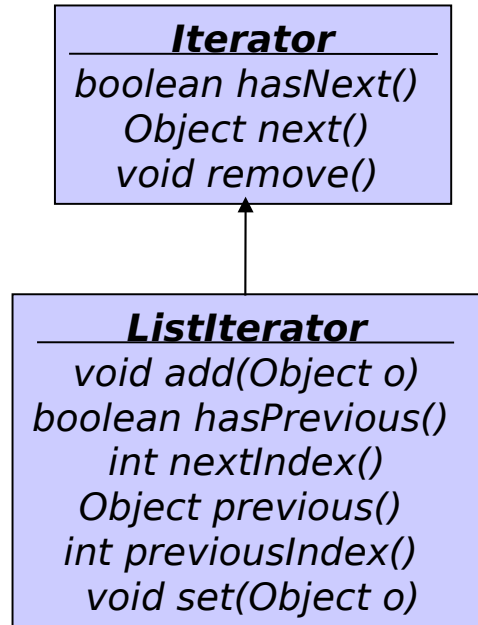
---

- **SortedSet** es un **Set** con un intrínseco (y automáticamente mantenido) orden
  - Los métodos adicional en esta subclase exponen este orden
- **SortedMap** es un **Map** con propiedades propiedades similares, pero basadas en el orden de la clave (key)
- En ambos casos, el orden puede ser determinado por una orden natural o por un **Comparator**



# Iteradores

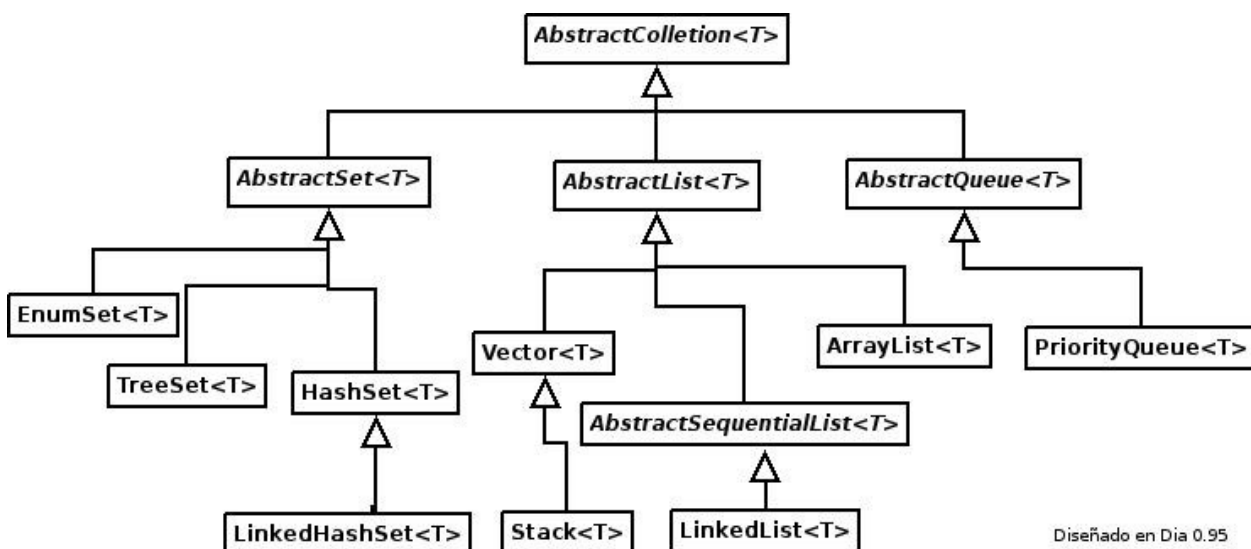
- Proveen una forma conveniente de iterar o recorrer todos los elementos de una colección.
- **ListIterator** añade métodos que permiten manipular de manera más específica colecciones del tipo **SequentialList**
- Permiten Añadir y remueven elementos de la colección asociada
- Iteradores de colecciones ordenadas iteran a través de la colección de acuerdo al orden establecido



# ***Interfaces e Implementaciones***

		<b>IMPLEMENTATIONS</b>				
		Hash Table	Resizable Array	Balanced Tree	Linked List	Legacy
<b>I N T E R F A C E S</b>	Set	HashSet		TreeSet		
	List		ArrayList		LinkedList	Vector, Stack
	Map	HashMap		TreeMap		HashTable, Properties

# Clases de Colecciones



# Como elegir las Implementaciones

---

## • Set / Map

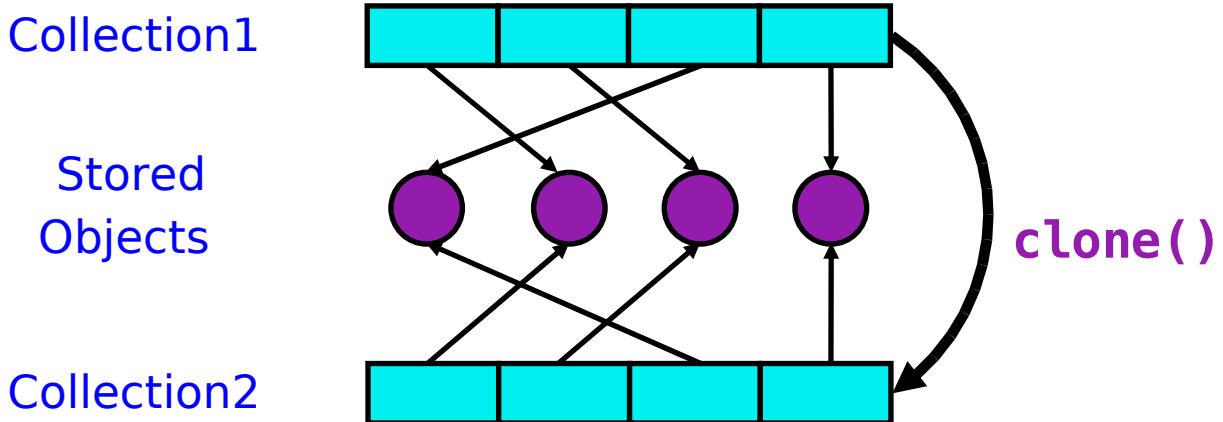
- HashSet / HashMap
  - Muy rápido, no ordenados
  - Elección de la *capacidad inicial* y *factor* de carga importante para el rendimiento
- TreeSet / TreeMap
  - Mantiene un árbol balanceado, bueno para iteraciones ordenadas
- HashTable
  - Sincronizado
  - Use la interface **Map**

## • List

- ArrayList
  - Muy rápida
  - Puede usar el método nativa **System.arraycopy**
- LinkedList
  - Bueno para colecciones volátiles, o añadir al frente (ej., colar)
- Vector
  - Sincronizado
  - Use la interface **List**

# Clonando Colecciones

- Se pueden hacer copias de la mayoría de colecciones con el método `clone()`
  - Este método crea una nueva colección pero no clona los elementos almacenados en la colección (llamada copia superficial)



# ***La Clase Collections***

---

- `java.util.Collections` consiste exclusivamente de métodos estáticos que operan sobre o retornan colecciones. Contiene:
  - Algoritmos polimórficos que operan sobre colecciones, e.j.,
    - `binarySearch`
    - `copy`
    - `min and max`
    - `replace`
    - `reverse`
    - `rotate`
    - `shuffle`
    - `sort`
    - `swap`
  - “Wrappers” – retorna una nueva colección
    - Colecciones sincronizadas
    - Colecciones no modificables

# Ejemplos: Set

```
import java.util.*;

public class BuscaDups {
    public static void main(String[] args) {
        Set<String> s = new HashSet<String>();
        for (String a : args)
            if (!s.add(a))
                System.out.println("Duplicado: " + a);

        System.out.println(s.size() + " distintos: " + s);
    }
}
```

Ahora si se ejecuta el programa con:

```
java BuscaDups Yo vengo yo veo yo salgo
```

La salida producida sería.

```
Duplicado: Yo
```

```
Duplicado: yo
```

```
4 palabras distintas: [Yo, salgo, veo, vengo]
```

# Ejemplos: ArrayList

```
import java.util.*;
public class ListExample {
    public static void main(String args[]) {
        List<String> lista = new ArrayList<String>();
        lista.add("Bernadine");
        lista.add("Elizabeth");
        lista.add("Gene");
        lista.add("Elizabeth");
        lista.add("Clara");
        System.out.println(list);
        System.out.println("2: " + list.get(2));
        System.out.println("0: " + list.get(0));
    }
}
```

El resultado de la ejecución:

**[Bernadine, Elizabeth, Gene, Elizabeth, Clara]**

**2: Gene**

**0: Bernadine**

# ***Ejemplos: LinkedList***

```
import java.util.*;
public class ListExample2 {
    public static void main(String args[]) {
        LinkedList <String> cola = new LinkedList<String>();
        cola.addFirst("Bernadine");
        cola.addFirst("Elizabeth");
        cola.addFirst("Gene");
        cola.addFirst("Elizabeth");
        cola.addFirst("Clara");
        System.out.println(cola);
        cola.removeLast();
        cola.removeLast();
        System.out.println(cola);
    }
}
```

El resultado de ejecutar el programa sería:  
**[Clara, Elizabeth, Gene, Elizabeth, Bernadine]**  
**[Clara, Elizabeth, Gene]**

# Ejemplos: Queue

```
import java.util.*;

public class CuentaRegresiva {
    public static void main(String[] args){
        int tiempo = 9;
        Queue<Integer> cola = new LinkedList<Integer>();
        for (int i = tiempo; i >= 0; i--)
            cola.add(i);
        while (!cola.isEmpty()) {
            System.out.println(cola.remove());
        }
    }
}
```

Salida:

5  
4  
3  
2  
1  
0

# Ejemplos: Map

```
import java.util.*;
public class Freq {
    public static void main(String[] args) {
        Map<String,Integer> m = new HashMap<String,Integer>();
        for (String k : args) {
            Integer f = m.get(k);
            if (f == null) f = 1;
            else f++;
            m.put(k, f);
        }
        System.out.println(m.size() + " palabras distintas:");
        System.out.println(m);
        Map<String,Integer> o = new TreeMap<String,Integer>(m);
        System.out.println(o);
    }
}
```

Al ejecutar: `java Freq if it is to be it is up to me to delegate`

El resultado sería:

**8 palabras distintas:**

`{to=3, delegate=1, be=1, it=2, up=1, if=1, me=1, is=2}`

`{be=1, delegate=1, if=1, is=2, it=2, me=1, to=3, up=1}`