

Introd. a Co **Metodologías de Programación I**

Introducción a Objetoslecciones.

Tipos de Colecciones .
Protocolo de Iteración.
Colecciones en Java

Introd. a Colecciones

Una Colección es toda estructura que permite almacenar objetos.

Se puede recorrer (o “iterar”) y se puede saber su tamaño.



- Una *colección* es simplemente un objeto que agrupa varios elementos en una sola unidad.
- Las colecciones son utilizadas para almacenar, recuperar y manipular los datos agregados.
- Por lo general, representan elementos de datos que forman un grupo natural, ej: un mazo de cartas (una colección de cartas).



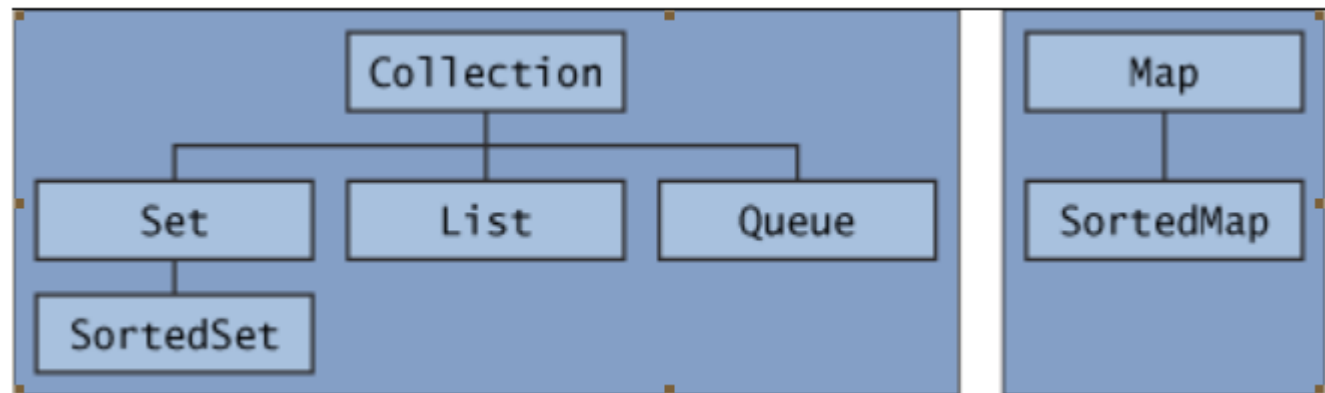
Introd. a Colecciones

Un *framework* (*armazón*) de colecciones es una arquitectura unificada para representar y manipular colecciones. Todos los *frameworks* de las colecciones contienen:

- Interfaces: Estos son los tipos de datos abstractos que representan colecciones. Las interfaces permiten manipular las colecciones de forma independiente de los detalles de su representación. En lenguajes orientados a objetos, las interfaces generalmente forman una jerarquía.
Una interface declara un marco abstracto de comportamiento.
- Implementaciones: Estas son las implementaciones concretas de las interfaces de Collection. En esencia, son estructuras de datos reutilizables.
- Algoritmos: Estos son los métodos que realizan cálculos útiles, tales como la búsqueda y la selección, en los objetos que implementan interfaces de la colección. Los algoritmos se dice que son *polimórficos*, es decir, el mismo método se puede utilizar en muchas implementaciones dependiendo de las diferentes de la interfaces que hayamos elegido. En esencia, los algoritmos son funcionalidad reutilizable.

Introd. a Colecciones

La Figura muestra la jerarquía del núcleo de interfaces de la JCF. Estas interfaces permiten manipular de forma independiente los detalles de su representación.



Set es un tipo especial de Collection, SortedSet es un tipo especial de Set, y así sucesivamente. Notar que la jerarquía se compone de dos árboles.

Introd. a Colecciones

- Las colecciones se organizan en:
 - Interfaces: Manipulan los datos independientemente de los detalles de implementación.
 - Clases: Implementan las interfaces.

Las interfaces y las clases están relacionadas en un armazón (framework) de colecciones para facilitar su uso.

- Para programar con colecciones se debe:
 - Elegir una interfaz adecuada a la funcionalidad requerida.
 - Elegir una clase que implemente la interfaz
 - Extender la clase si fuera necesario.

Tipos de colecciones

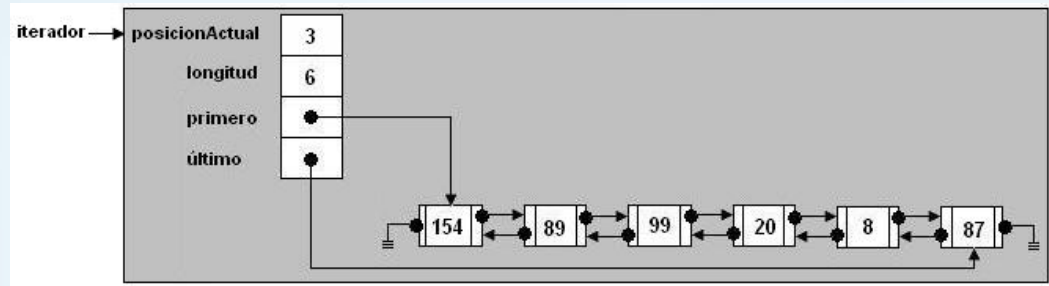
- Existen distintos tipos de colecciones; ordenadas, sin orden, indexadas, que permiten elementos repetidos, que no permiten elementos repetidos, que almacena un elemento compuesto (clave, valor)

Protocolo de Iteración

Protocolo para "manipular" esos objetos/elementos.

Utilizaremos un iterador para el recorrido de cualquier colección y eliminar un elemento.

Interface Iterator



Interface Iterator{

boolean hasNext();

/ Devuelve true si la iteración tiene mas elementos */*

Object next();

/ Devuelve el siguiente elemento de la iteración*/*

void remove();

/ Elimina el último elemento devuelto por la iteración*/*

}

Iteración

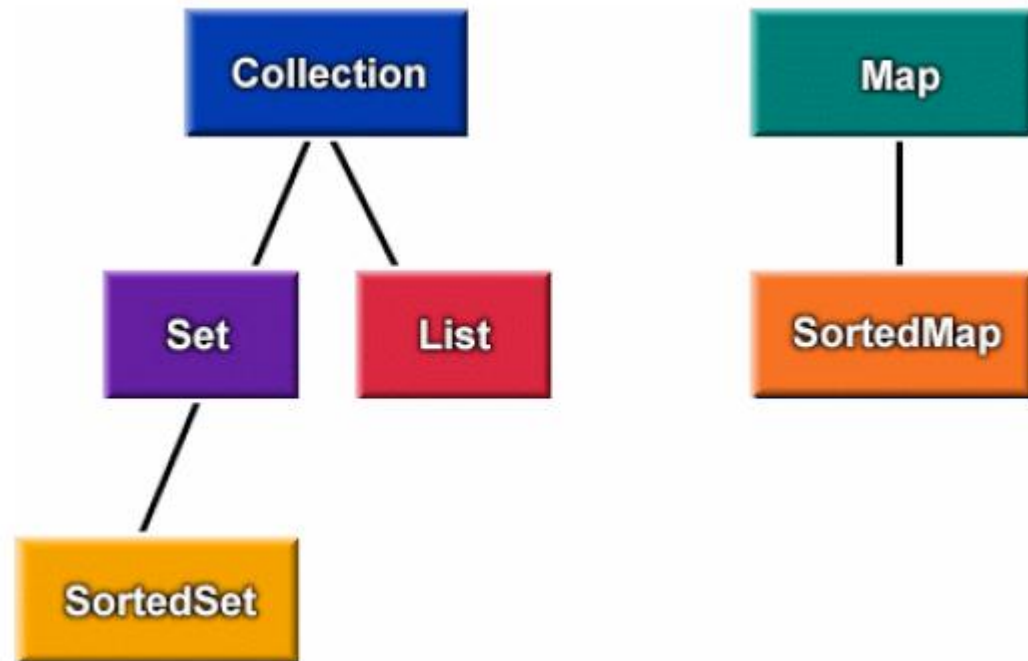
- La interfaz **ListIterator** extiende a **Iterator** y maneja un objeto **List** ordenado. Permite iterar hacia delante y hacia atrás.

Ejemplo de uso de Iteradores

- Ejemplo, en el que el iterador nos permite recorrer los elementos de un *Vector*:

```
Vector vec = new Vector();  
vec.add( new String( "hola" ) );  
vec.add( new String( "adios" ) );  
Iterator it = vec.iterator();  
while ( it.hasNext() )  
    System.out.println( (String) it.next() );  
)
```

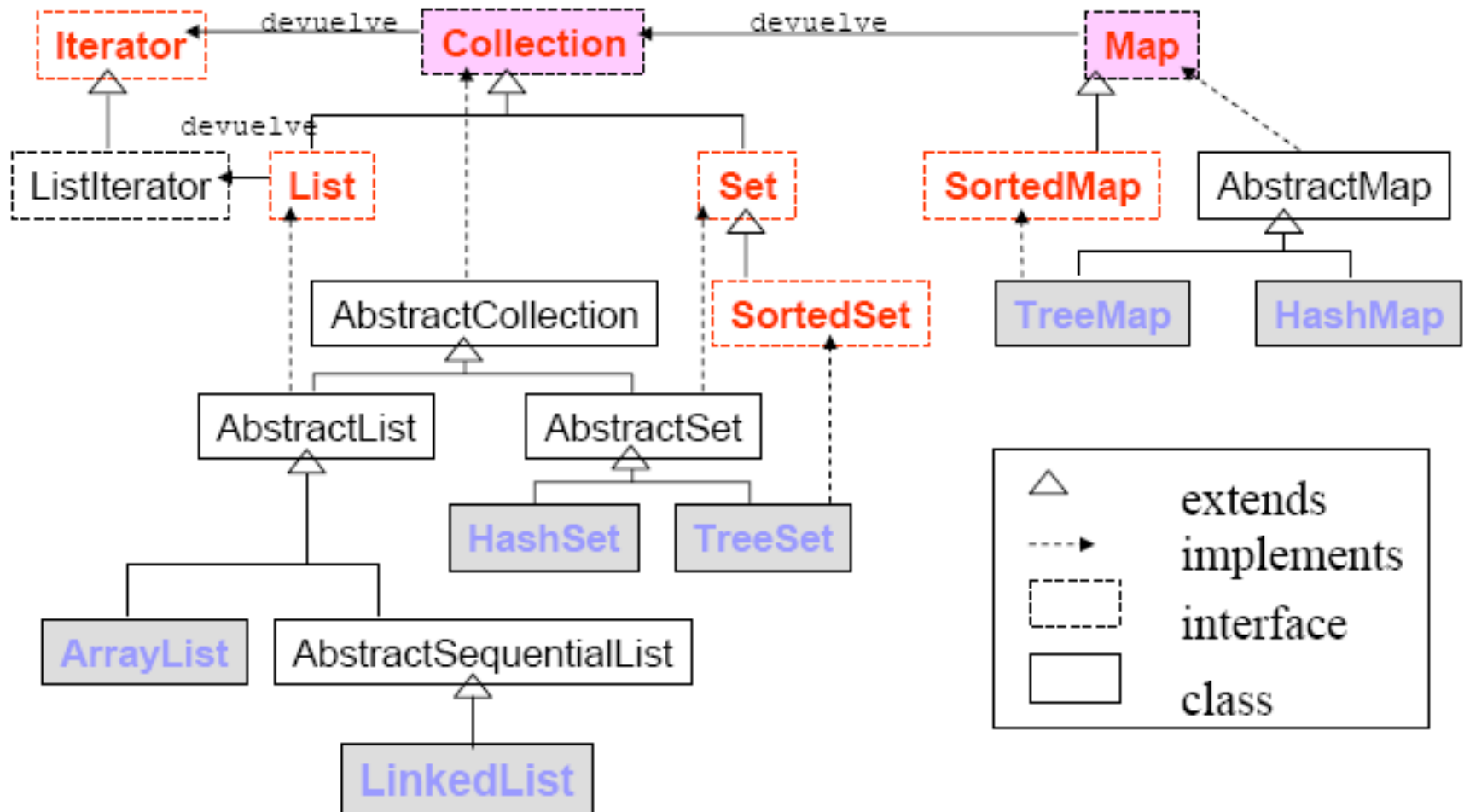
Jerarquía de interfaces de colecciones



Jerarquía de interfaces de colecciones

- Collection: Representa un grupo de objetos. Sin implementaciones directas, agrupa la funcionalidad general que todas las colecciones ofrecen.
- Set: Colección que no puede tener objetos duplicados.
- SortedSet : Set que mantiene los elementos ordenados
- List: Colecciones ordenadas (secuencias) en las que cada elemento ocupa una posición identificada por un índice. El primer índice es el 0. Las listas admiten duplicados.
- Map. Son pares de datos(clave, valor). No puede haber claves duplicadas y cada clave se corresponde con al menos un valor.
- SortedMap: Map que mantiene las claves ordenadas.
- Con la aparición de las nuevas versiones: Interface Queue(colas) y LinkedList (Listas Enlazadas)

Jerarquía de interfaces de colecciones



Ejemplo de clases implementadas

- Las interfaces List, Set y SortedSet son descendientes de la interface Collection

El concepto de Polimorfismo aplica para todas las clases que implementan estas interfaces.

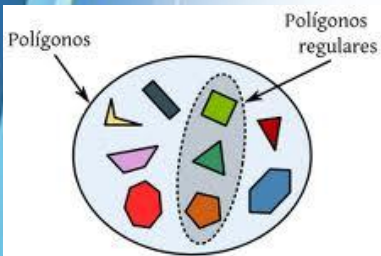
- Las clases que implementan la interface List son: ArrayList y LinkedList
- Las clases que implementan la interface Set son: HashSet y LinkedHashSet
- La clase que implementa la sub-interface SortedSet es: TreeSet.

Ejemplo de clases implementadas

- Las clases que implementan la interfase Set son: HashSet y LinkedHashSet

Set


- Por sobre lo que es una collection, un set agrega una sola restricción: No puede haber duplicados.
- Por lo general en un set el orden no es dato. Si bien es posible que existan sets que nos aseguren un orden determinado cuando los recorremos (por ejemplo obtener strings en orden alfabético), ese orden no es arbitrario y decidido por nosotros, ya que la interfaz Set no tienen ninguna funcionalidad para manipularlo (como si lo admite la interfaz List).
- La ventaja de utilizar Sets es que preguntar si un elemento ya está contenido mediante “contains()” suele ser muy eficiente. Entonces es conveniente utilizarlos cada vez que necesitemos una colección en la que no importe el orden, pero que necesitemos preguntar si un elemento está o no.



HashSet

- Existen varias implementaciones de Set. La más comúnmente usada es HashSet.
- Los Sets (y los Maps) aprovechan una característica de Java: Todos los objetos heredan de Object, por lo tanto todos los métodos de la clase Object están presentes en todos los objetos. Dicho de otra manera, hay ciertas cosas que todo objeto en Java sabe hacer.
- Dos de estas cosas son:
 1. Saber si es igual a otro, con su método equals().

HashSet

- 
2. Devolver un número entero de modo tal que si dos objetos son iguales ese número también lo será (se conoce esto como un *hash*). Esto todo objeto lo sabe hacer con su método `hashCode()`.
 - La clase `HashSet` aprovecha la segunda de las funciones. A cada objeto que se añade a la colección se le pide que calcule su “hash”. Este valor será un número entre -2147483647 y 2147483648. Basado en ese valor se lo guarda en una tabla. Más tarde, cuando se pregunta con `contains()` si un objeto *x* ya está, habrá que saber si está en esa tabla.

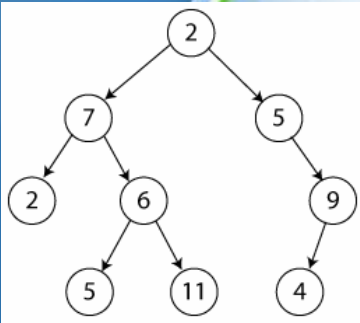
¿En qué posición de la tabla está? `HashSet` puede saberlo, ya que para un objeto determinado, el hash siempre va a tener el mismo valor.

Ejemplo de clases implementadas

- La clase que implementa la sub-interfase SortedSet es: TreeSet.

TreeSet

- TreeSet usa una técnica completamente diferente a la explicada para HashSet. Construye un árbol con los objetos que se van agregando al conjunto. Un árbol es una forma en computación de tener un conjunto de cosas todo el tiempo en orden, y permitir que se agreguen más cosas y que el orden se mantenga.



- Una ventaja de TreeSet es que el orden en el que aparecen los elementos al recorrerlos es el orden natural de ellos. Una desventaja es que mantener todo ordenado tiene un costo, y esta clase es un poquito menos eficiente que HashSet.

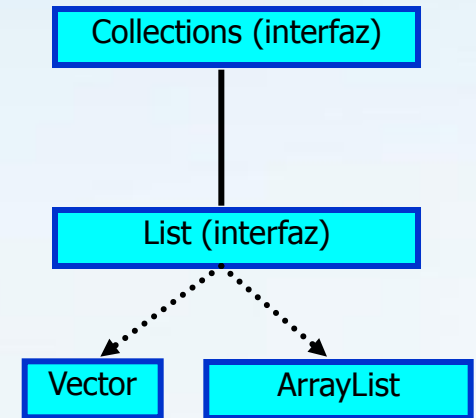
Map

- Un Map representa lo que en otros lenguajes se conoce como “diccionario” y que se suele asociar a la idea de “tabla hash” (aunque no se implemente necesariamente con esa técnica).
- Un Map es un conjunto de valores, con el detalle de que cada uno de estos valores tiene un objeto extra asociado. A los primeros se los llama “claves” o “keys”, ya que nos permiten acceder a los segundos.

Clases

- Ahora veremos las clases mas conocidas que implementan las interfaces y se encuentran en el paquete `java.util`

Colecciones basicas



Esto indica que *List* es una *interface* que hereda de la interface *Collection*.

Podemos encontrar dos clases que implementan su comportamiento: *ArrayList* y *Vector*.

Colecciones básicas :Vector

La clase **Vector** es parte del paquete **java.util** de la librería estándar de clases de Java. Ofrece un servicio similar a un arreglo, ya que se pueden almacenar y acceder valores y referencias a través de un índice.

Mientras un arreglo tiene un tamaño dado, un objeto de tipo **Vector** puede *dinámicamente crecer y decrecer conforme se vaya necesitando*.

Un elemento puede insertarse y eliminarse de una posición específica a través de la invocación de un sólo método.

A diferencia de un arreglo, un **Vector** no está declarado para ser de un tipo particular.

Un objeto de tipo **Vector** maneja una lista de referencias a la clase **Object**.

Métodos de la clase Vector

Vector () Constructor: crea un vector inicialmente vacío

void addElement (Objet obj) Inserta el objeto especificado al final del vector

void setElementAt (Object obj, int índice) Inserta el objeto especificado en el vector en la posición especificada

Object remove (int índice) Elimina el objeto que se encuentra en la posición especificada y lo regresa

boolean removeElement (Object obj) Elimina la primera ocurrencia del objeto especificado en el vector

void removeElementAt (int índice) Elimina el objeto especificado en el índice del vector

Métodos de la clase **Vector**

void clear () Elimina todos los objetos del vector

boolean contains (Object obj) Retorna verdadero si el objeto dado pertenece al vector

int indexOf (Object obj) Retorna el índice del objeto especificado. Regresa -1 si no fue encontrado el objeto

Object elementAt (int índice) Retorna el componente en el índice especificado

boolean isEmpty () Retorna verdadero si el vector no contiene elementos

int size () Retorna el número de elementos en el vector

Ejemplo de Vector

```
import java.util.*;
```

```
class Coche {  
    private int numCoche;  
    Coche( int i ) {  
        numCoche = i;  
    }  
    void print() {  
        System.out.println(  
            "Coche #" + numCoche );  
    }  
}
```

```
class Barco {  
    private int numBarco;  
    Barco( int i ) {  
        numBarco = i;  
    }  
    void print() {  
        System.out.println(  
            "Barco #" + numBarco );  
    }  
}
```

```
public class PruebaVector {  
  
    public static void main( String  
        args[] ) {  
        Vector coches = new Vector();  
        for( int i=0; i < 7; i++ )  
            coches.addElement( new Coche( i  
                ) );  
  
        // No hay problema en añadir un  
        // barco a los coches  
  
        coches.addElement( new Barco( 7  
            ) );  
  
        for( int i=0; i < coches.size(); i++ )  
            (( Coche )coches.elementAt( i )  
                ).print(); //  
  
        El barco solamente es detectado  
        en tiempo de ejecucion  
  
    }  
}
```

Otro Ejemplo de Vector

- ```
import java.util.Vector;
/**
 * Demuestra el uso de un objeto de la clase Vector
 */

public class Beatles
{
 public static void main ()
 {
 Vector band = new Vector ();
 band.addElement ("Paul");
 band.addElement ("Pete");
 band.addElement ("John");
 band.addElement ("George");

 System.out.println (band);

 band.removeElement ("Pete");

 System.out.println (band);
 System.out.println ("En la posición 1 está: " + band.elementAt (1));

 band.insertElementAt ("Ringo", 2);

 System.out.println ("Tamaño de la banda: " + band.size ());
 for (int i = 0; i < band.size (); i++)
 System.out.print (band.elementAt (i) + " ");
 }
}
```

# Clase java.util.Array

Los Arrays son el tipo más sencillo de colección

```
Point2D[] puntos;
puntos = new Point2D [100];
for (int i=0; i <= puntos.length(); i++)
 puntos[i]= new Point2D();
```

El tipo de los objetos del array debe declararse en tiempo de compilación.

Hay dos características que diferencian a los arrays de cualquier otro tipo de colección:

**eficiencia y tipo.**



# Clase java.util.Array

El array es la forma más eficiente que Java proporciona para almacenar y acceder a una secuencia de objetos.

El array es una simple secuencia lineal, que hace que el acceso a los elementos sea muy rápido, pero el precio que hay que pagar por esta velocidad es que cuando se crea un array su tamaño es fijado y no se puede cambiar a lo largo de la vida del objeto.

Se puede sugerir la creación de un array de tamaño determinado y luego, ya en tiempo de ejecución, crear otro más grande, mover todos los objetos al nuevo y borrar el antiguo. Esto es lo que hace que la clase **Vector** sea menos eficiente que un array, en cuestiones de velocidad.

# Clase Collections

- Ordenar y buscar en Colecciones:  
Collections
  - La clase Collections (que no es la interface Collection) nos permite ordenar y buscar elementos en listas.

Se usaran los métodos:

- sort y binarySearch
- equals, hashCode y compareTo.