

Curso de Programación en Java



Juan Francisco Maldonado León
Arquitecto de Software



Java

Collections



Juan Francisco Maldonado León
Arquitecto de Software



Collections Frameworks

Una colección es simplemente un grupo de objetos de varios elementos representado en una sola unidad.

Las colecciones se utilizan para almacenar, recuperar, manipular y comunicar los datos agregados.



Colección de Contactos
Telefónicos.

Componentes Principales del Framework

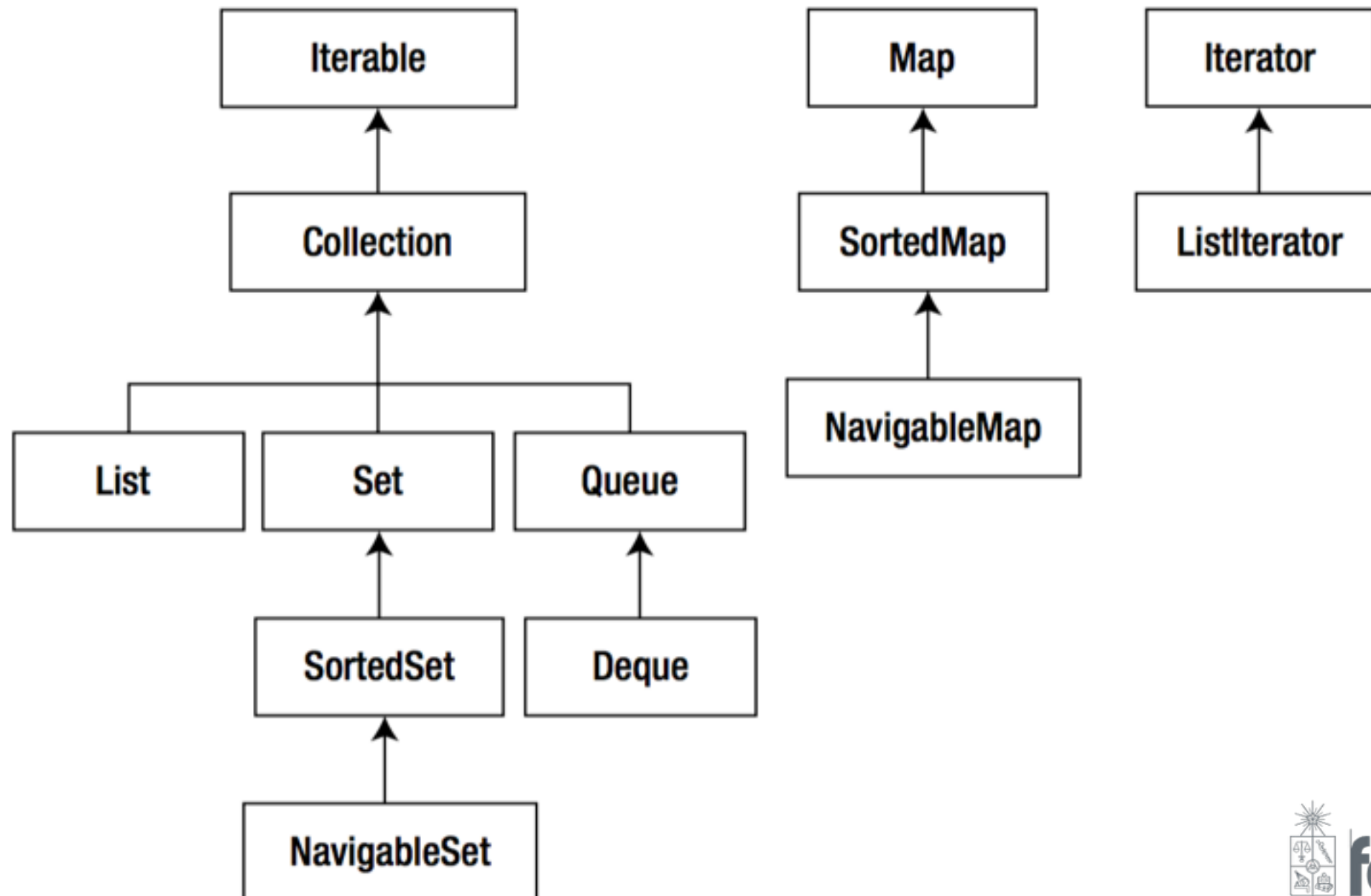
- **Clases abstractas e interfaces:** El Framework de la colecciones tiene muchas clases abstractas e interfaces que proporcionan funcionalidad general. (List, Set, Map, etc)
- **Clases concretas:** Estos son los casos reales de contenedores que va a utilizar en los programas.
- **Algoritmos:** funcionalidades como la clasificación, búsqueda, etc.



Introducción a Java

Collections Framework

Interfaces





Introducción a Java

Collections Framework

Abstract Class/Interface	Short Description
Iterable	A class implementing this interface can be used for iterating with a <code>for each</code> statement.
Collection	Common base interface for classes in the collection hierarchy. When you want to write methods that are very general, you can pass the <code>Collection</code> interface. For example, <code>max()</code> method in <code>java.util.Collections</code> takes a <code>Collection</code> and returns an object.
List	Base interface for containers that store a sequence of elements. You can access the elements using an index, and retrieve the same element later (so that it maintains the insertion order). You can store duplicate elements in a <code>List</code> .
Set, SortedSet, NavigableSet Queue, Deque	<p>Interfaces for containers that don't allow duplicate elements. <code>SortedSet</code> maintains the set elements in a sorted order. <code>NavigableSet</code> allows searching the set for the closest matches.</p> <p><code>Queue</code> is a base interface for containers that holds a sequence of elements for processing. For example, the classes implementing <code>Queue</code> can be LIFO (last in, first out— as in stack data structure) or FIFO (first in, first out—as in queue data structure). In a <code>Deque</code> you can insert or remove elements from <i>both</i> the ends.</p>
Map, SortedMap, NavigableMap	Base class for containers that map keys to values. In <code>SortedMap</code> , the keys are in a sorted order. A <code>NavigableMap</code> allows you to search and return the closest match for given search criteria. Note that <code>Map</code> hierarchy does <i>not</i> extend the <code>Collection</code> interface.



Introducción a Java

Collections Framework

Interfaz Collection

Method	Short description
<code>boolean add(Element elem)</code>	Adds elem into the underlying container.
<code>void clear()</code>	Removes all elements from the container.
<code>boolean isEmpty()</code>	Checks whether the container has any elements or not.
<code>Iterator<Element> iterator()</code>	Returns an <code>Iterator<Element></code> object for iterating over the container.
<code>boolean remove(Object obj)</code>	Removes the element if obj is present in the container.
<code>int size()</code>	Returns the number of elements in the container.
<code>Object[] toArray()</code>	Returns an array that has all elements in the container.



Introducción a Java

Collections Framework

Clases Concretas

Concrete Class	Short Description
ArrayList	Internally implemented as a resizable array. This is one of the most widely used concrete classes. Fast to search, but slow to insert or delete. Allows duplicates.
LinkedList	Internally implements a doubly-linked list data structure. Fast to insert or delete elements, but slow for searching elements. Additionally, LinkedList can be used when you need a stack (LIFO) or queue (FIFO) data structure. Allows duplicates.
HashSet	Internally implemented as a hash-table data structure. Used for storing a set of elements—it does not allow storing duplicate elements. Fast for searching and retrieving elements. It does <i>not</i> maintain any order for stored elements.
TreeSet	Internally implements a red-black tree data structure. Like HashSet, TreeSet does not allow storing duplicates. However, unlike HashSet, it stores the elements in a sorted order. It uses a tree data structure to decide where to store or search the elements, and the position is decided by the sorting order.
HashMap	Internally implemented as a hash-table data structure. Stores key and value pairs. Uses hashing for finding a place to search or store a pair. Searching or inserting is very fast. It does <i>not</i> store the elements in any order.
TreeMap	Internally implemented using a red-black tree data structure. Unlike HashMap, TreeMap stores the elements in a sorted order. It uses a tree data structure to decide where to store or search for keys, and the position is decided by the sorting order.

List

Collections

Un objeto List (conocido como secuencia) es una Colección ordenada que puede contener elementos duplicados.

Al igual que los índices de arreglos, los índices de objetos List empiezan desde cero



Introducción a Java

Collections Framework

ArrayList

Collections

ArrayList es una colección de tipo lista. Un ArrayList es una colección de tamaño dinámico y pertenece al package java.util

Las posiciones de un ArrayList van desde el 0 hasta la cantidad de elementos - 1



[0] [1] [..]



Inicializar un ArrayList

Collections

Inicializar por defecto

```
ArrayList enteros = new ArrayList();
```

Inicializar por defecto indicando el tipo

```
ArrayList<Integer> enteros = new ArrayList<Integer>();
```

Inicializar indicando la cantidad de elementos

```
ArrayList<Integer> enteros = new ArrayList<Integer>(3);
```



Introducción a Java

Collections Framework

Inicializar un ArrayList

Collections

Un ArrayList solo puede contener tipo de datos complejos, no primitivos.

```
List<int> enteros = new ArrayList<int>();
```


Inicializar un ArrayList

Collections

```
ArrayList<Integer> enteros = new ArrayList<Integer>();
```



Indica el tipo de objeto que contendrá cada posición de la lista



Agregar un elemento

Collections

```
List<String> enteros = new ArrayList<String>();
```

```
enteros.add("Diez");
```

Posición 0

```
enteros.add("Once");
```

Posición 1

Si no indicamos la posición en el método add, añadirá el elemento en la siguiente posición disponible.



Agregar un elemento

Collections

Utilizando metodo add sobrecargado

```
List<String> enteros = new ArrayList<String>();  
enteros.add("Diez");  
enteros.add("Once");  
enteros.add(2, "Doce");
```

Indica la
posición

Indica el
elemento ha
agregar



Introducción a Java

Collections Framework

Agregar un elemento

Collections

```
List<String> enteros = new ArrayList<String>();  
enteros.add("Diez");           posición 1  
enteros.add("Once");           posición 2  
enteros.add(0, "Doce");        posición 0
```




Obtener un elemento ArrayList

Collections

```
List<String> enteros = new ArrayList<String>();  
enteros.add("Diez");  
enteros.add("Once");  
enteros.get(0);
```

Obtiene el
elemento según
la posición
indicada



Obtener un elemento ArrayList

Collections

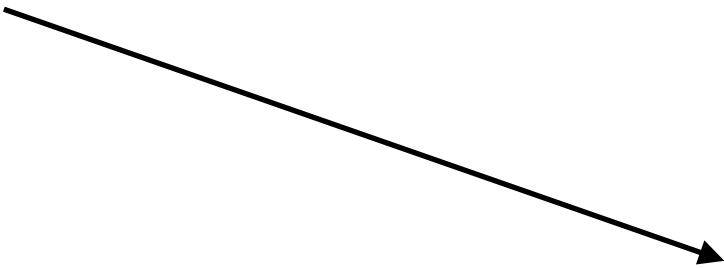
```
List<String> enteros = new ArrayList<String>();  
enteros.add("Diez");  
enteros.add("Once");  
enteros.get(4);
```

[java.lang.IndexOutOfBoundsException](#)

Quitar un elemento ArrayList

Collections

```
ArrayList<String> cadenas = new ArrayList<String>();  
cadenas.add("Hola");  
cadenas.add("Chao");  
cadenas.remove(0);
```



Elimina el
elemento según
la posición
indicada



Introducción a Java

Collections Framework

Iterar un ArrayList

Collections

El método `size` indica la cantidad de elemento de un ArrayList

```
for( int i = 0; i < cadenas.size() ; i++ )  
{  
    System.out.println( cadenas.get(i) );  
}
```




Introducción a Java

Collections Framework

Iterar un ArrayList

Collections

```
for( String cadena : cadenas )  
{  
    System.out.println(cadena);  
}
```



fcfm

Ciencias de la
Computación
FACULTAD DE CIENCIAS
FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE



Introducción a Java

Collections Framework

Iterar un ArrayList

Collections

```
Iterator<String> iterador = cadenas.iterator();

while( iterador.hasNext() )
{
    System.out.println( iterador.next() );
}
```



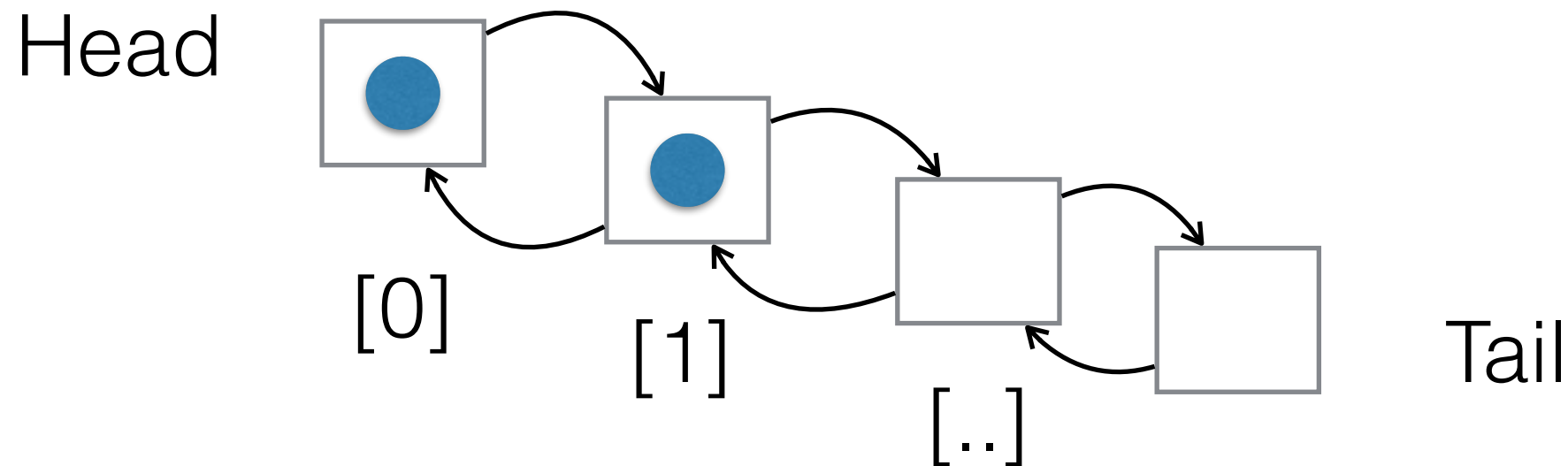
fcfm

Ciencias de la
Computación
FACULTAD DE CIENCIAS
FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

LinkedList

Collections

LinkedList representa un colección de tipo lista doblemente enlazada, es decir cada elemento conoce al siguiente elemento y al anterior.





Introducción a Java

Collections Framework

LinkedList

Collections

Método	Descripción
E getFirst()	Obtiene el primer elemento
E getLast()	Obtiene el ultimo elemento
E removeFirst()	Elimina el primer elemento
E removeLast()	Elimina el ultimo elemento
void addFirst(E)	Agrega un elemento al inicio
void addLast(E)	Agrega un elemento al final



Introducción a Java

Collections Framework

LinkedList

Collections

```
Persona per1 = new Persona("juan", "16.942.333-6");  
Persona per2 = new Persona("juan", "16.942.333-6");  
Persona per3 = new Persona("juan", "16.942.333-6");
```

```
List<Persona> personas = new LinkedList<Persona>();  
personas.add(per1);  
personas.add(per2);  
personas.add(per3);
```

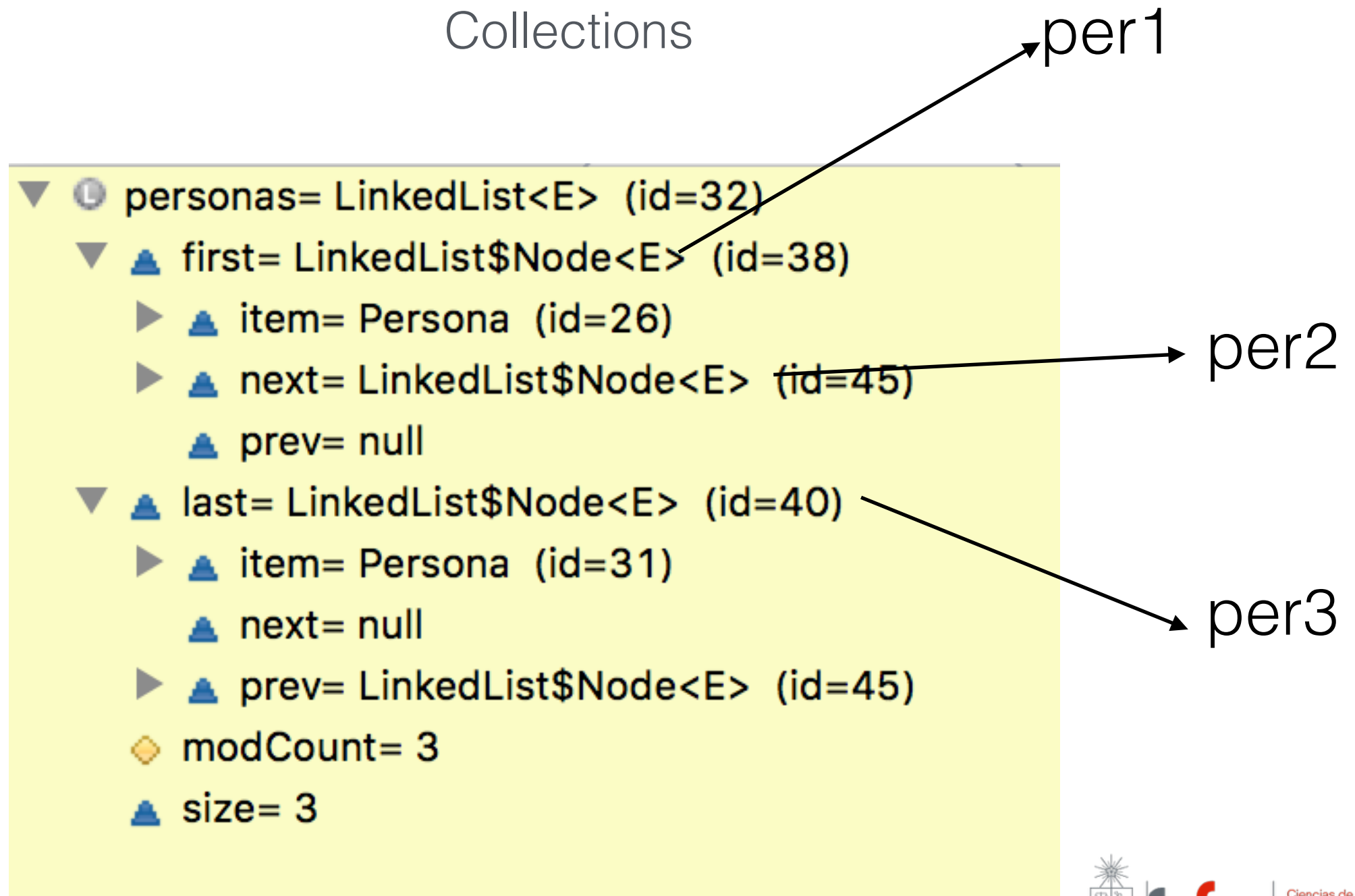


Introducción a Java

Collections Framework

LinkedList

Collections



Ordenar una Lista

Collections

Hay dos formas de ordenar una lista utilizando el framework de colecciones. Una de ellas es implementar la interfaz Comparable.

Por ejemplo si quisiéramos ordenar una lista de tipo Persona, deberíamos implementar la interfaz Comparable en la clase Persona.

```
List<Persona> personas = new ArrayList<Persona>();
```

Ordenar una Lista

Collections

```
public class Persona implements Comparable<Persona>
```

Comparable es una interfaz genérica, por ende debemos indicarle el tipo de Objeto que queremos comprar, en este caso, Lo haremos con Persona

Ordenar una Lista

Collections

Al implementar la interfaz comparable debemos implementar el método `compareTo`, este método retorna un entero que puede ser 1, -1 ó 0

```
@Override  
public int compareTo(Persona o) {  
    return 0;  
}
```



Introducción a Java

Collections Framework

Ordenar una Lista

Collections

Retorna 1 si el objeto es mayor al objeto recibido.

Retorna -1 si el objeto es menor al objeto recibido.

Retorna 0 si el objeto es igual al objeto recibido.



Ordenar una Lista

Collections

Para ordenar una lista de personas según su edad

```
@Override
public int compareTo(Persona o) {
    if( this.getEdad() > o.getEdad() )
        return 1;
    if( this.getEdad() < o.getEdad() )
        return -1;
    if( this.getEdad() == o.getEdad() )
        return 0;
    return 0;
}
```



Ordenar una Lista

Collections

```
List<Persona> personas = new ArrayList<Persona>();  
Persona per1 = new Persona("juan", "16.942.333-6", 10);  
Persona per2 = new Persona("francisco", "16.942.333-6", 20);  
Persona per3 = new Persona("pablo", "16.942.333-6", 3);  
personas.add(per1);  
personas.add(per2);  
personas.add(per3);
```

```
Collections.sort(personas);
```

El método estático sort de la clase
java.util.Collections
realizara el orden según la implementación del
método compareTo

Ordenar una Lista

Collections

El contra de utilizar este método para ordenar una lista, es que quedamos limitados, a ordenar una colección solamente por un único termino. Es decir si la colección de personas, ademas de ordenar según su edad, quisiéramos ordenar también por nombre, no podríamos realizarlo, dado que solamente podemos escribir una vez el método `compareTo` en la clase `persona`.

Ordenar una Lista

Collections

La segunda forma de ordenar una colección es utilizando la interfaz Comparator, en el método sort sobrecargado.

`Collections.sort(List<E>, Comparator)`



Lista a ordenar



Clase que implemente la interfaz comparator

Ordenar una Lista

Collections

La segunda forma de ordenar una colección es utilizando la interfaz Comparator, en el método sort sobrecargado.

`Collections.sort(List<E>, Comparator)`



Lista a ordenar



Clase que implemente la interfaz comparator



Introducción a Java

Collections Framework

Ordenar una Lista

Collections

```
public class OrdenarPorNombre implements Comparator<Persona> {  
  
    @Override  
    public int compare(Persona o1, Persona o2) {  
        return o1.getNombre().compareTo(o2.getNombre());  
    }  
}
```

Al igual que el método compareTo, el método compare retorna un entero indicando si un objeto es mayor, menor o igual que otro



Introducción a Java

Collections Framework

Ordenar una Lista

Collections

```
public class OrdenarPorNombre implements Comparator<Persona> {  
  
    @Override  
    public int compare(Persona o1, Persona o2) {  
        return o1.getNombre().compareTo(o2.getNombre());  
    }  
  
}
```

```
Collections.sort(personas, new OrdenarPorNombre());
```



fcfm

Ciencias de la
Computación
FACULTAD DE CIENCIAS
FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE



Ordenar una Lista

Collections

Para evitar crear una clase que tenga un solo método, podemos crear una clase anónima de la siguiente forma.

```
Collections.sort(personas, new Comparator<Persona>() {  
  
    @Override  
    public int compare(Persona o1, Persona o2) {  
        return o1.getNombre().compareTo(o2.getNombre());  
    }  
});
```

Algoritmos de Collections

Collections

Algoritmo	Descripción
sort	Ordena los elementos de un objeto <code>List</code> .
binarySearch	Localiza un objeto en un objeto <code>List</code> .
reverse	Invierte los elementos de un objeto <code>List</code> .
shuffle	Ordena al azar los elementos de un objeto <code>List</code> .
fill	Establece cada elemento de un objeto <code>List</code> para que haga referencia a un objeto especificado.
copy	Copia referencias de un objeto <code>List</code> a otro.
min	Devuelve el elemento más pequeño en un objeto <code>Collection</code> .
max	Devuelve el elemento más grande en un objeto <code>Collection</code> .
addAll	Anexa todos los elementos en un arreglo a una colección.
frequency	Calcula cuántos elementos en la colección son iguales al elemento especificado.
disjoint	Determina si dos colecciones no tienen elementos en común.