UT07. 04-COLECCIONES EN JAVA. LIST

Programación de 1DAW

C.I.F.P. Carlos III - Cartagena

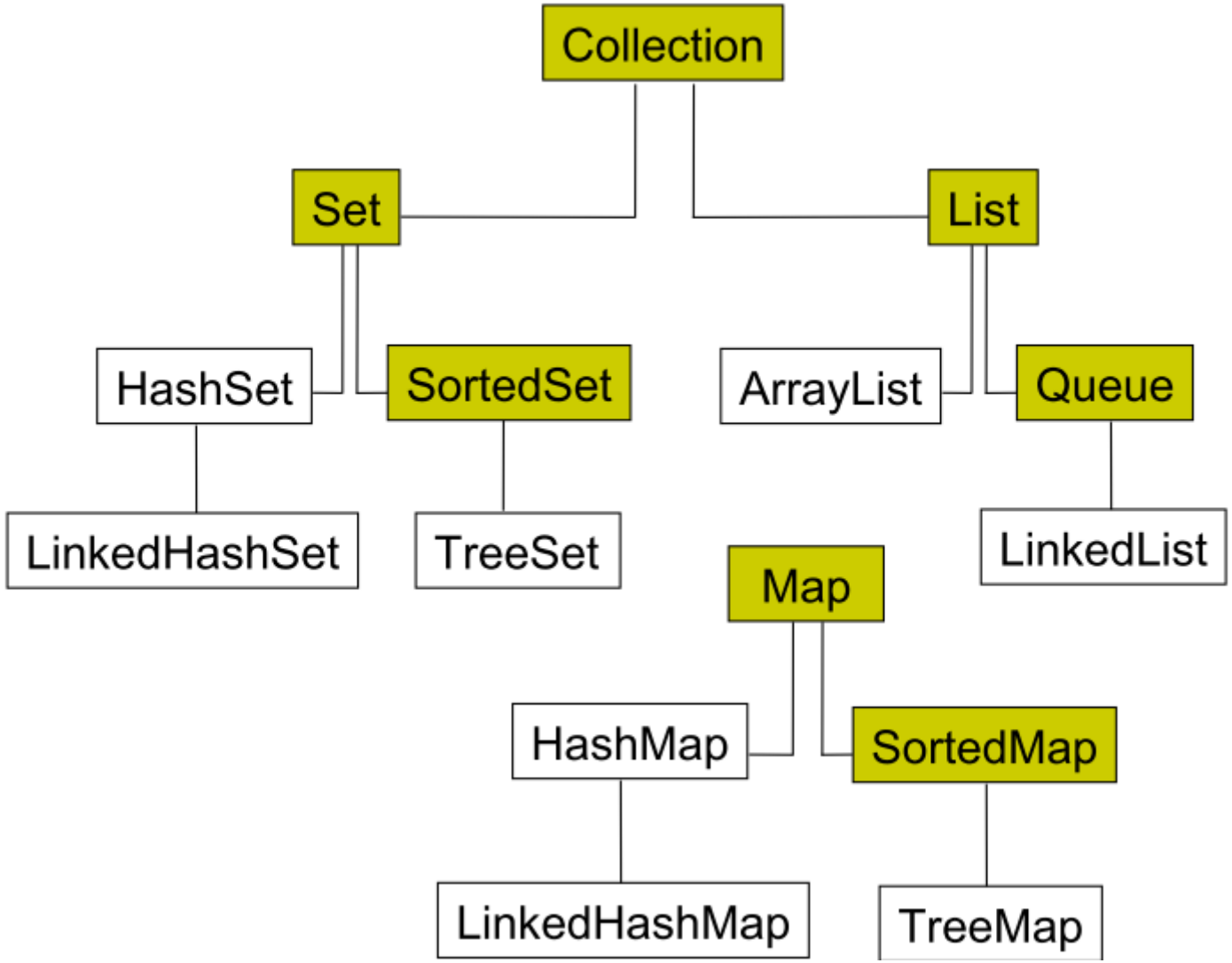
Introducción

- Listas
  - Listas enlazadas
  - Listas doblemente enlazadas
  - Listas circulares
- Pilas (LIFO)
- Colas (FIFO)
- Árboles
- Tablas Hash
- Grafos

Colecciones

- Una **colección** es un objeto que almacena un conjunto de referencias a otros objetos.
- Características.
  - Son estructuras **dinámicas**
    - No tienen tamaño fijo
    - Pueden añadir y eliminar objetos en tiempo de ejecución.
  - Incluidas en el paquete **java.util**.
  - Proporcionan **métodos** para realizar las siguientes operaciones:
    - **Añadir** objetos.
    - **Eliminar** objetos.
    - **Obtener** un objeto.
    - **Localizar** un objeto.
    - **Iterar** a través de la colección.

Java Collections Framework



Colecciones

- **Set**
  - No puede contener elementos duplicados ➔ **no duplicados**
  - Sus elementos **pueden o no estar ordenados** ➔ **sí/no ordenados**
- **List**
  - Colección ordenada de forma secuencial ➔ **ordenado secuencial**
  - Puede contener elementos duplicados ➔ **sí duplicados**
  - Se tiene un control preciso sobre la posición que ocupa cada objeto, pudiendo acceder a cada elemento mediante su índice ➔ **acceso por índice**
- **Map**
  - Colección en la que se hace corresponder claves con valores ➔ **clave-valor**
  - No puede contener claves duplicadas, pero sí valores duplicados ➔

  no claves duplicadas- sí valores duplicados
  - Sus elementos pueden o no estar ordenados ➔ **sí/no ordenados**

Interfaz Collection<E>

- java.util.**Collection** ➔ interfaz fundamental
- Elementos de la colección: clase **Object** ➔ **casting**, excepto con **genéricos.**

Método	Descripción
boolean **add** (E element)	**Añade** el objeto a la colección. Devuelve true si se pudo completar la operación. Si no cambió la colección como resultado de la operación devuelve false
boolean **remove** (Object o)	**Elimina** al objeto indicado de la colección
int **size**()	Devuelve el **número de objetos** almacenados en la colección
boolean **isEmpty**()	Indica si la colección **está vacía**
boolean **contains** (Object o)	Devuelve true si la **colección contiene al objeto** indicado en o
void **clear**()	**Elimina todos** los elementos de la colección
boolean **addAll** (Collection<? extends E> c)	**Añade todos** los elementos de la colección **c** a la colección actual
boolean **removeAll** (Collection<?> c)	**Elimina todos** los objetos de la colección actual que estén en la colección **c**
boolean **retainAll** (Collection <?> c)	**Conserva** sólo los elementos de esta lista que se encuentran en la colección **c**
boolean **containsAll** (Collection <?> c)	Indica si la colección contiene todos los elementos de **c**
Object [] **toArray**()	Devuelve **la colección en un array** de objetos
Iterator<E> **iterator**()	Obtiene el **objeto iterador** de la colección.

Interfaz Iterator<E>

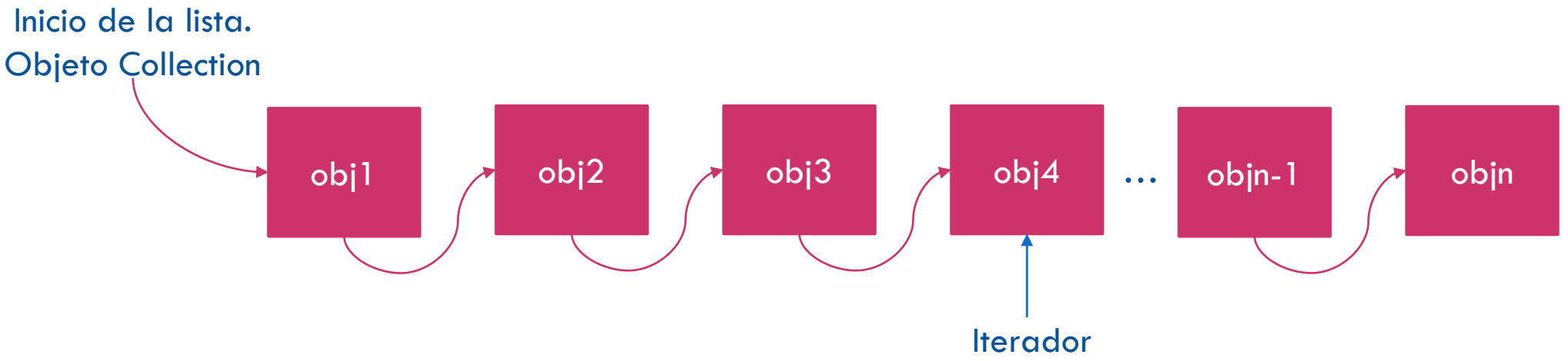
- **Iteradores**. Define **objetos que permiten recorrer** los elementos de una colección y borrarlos.

Método Iterator<E>	Descripción Iterator<E>
E **next** ()	Obtiene el **siguiente objeto** de la colección. Si se ha llegado al final de la colección y se intenta seguir, da lugar a una excepción de tipo: **NoSuchElementException** (derivado de **RunTimeException**)
boolean **hasNext**()	Indica **si hay un elemento siguiente** (evita la excepción)
void **remove**()	**Elimina** el último elemento devuelto por **next.**

```
//suponiendo que colecciónString es una colección de textos
Iterator it = colecciónString.iterator();
while(it.hasNext()){
    String s = (String)it.next();   //casting necesario porque devuelve un tipo Object
    System.out.println(s);
}
```

Collection.Iterator

- La interfaz **Collection** junto con los iteradores de **Iterator** permiten crear listas simplemente enlazadas
- Los elementos se añaden solo al final de la lista → **add**
- Se recorren de izquierda a derecha
- Se pueden borrar elementos del interior de la lista→ **remove(Object)**



List<E>

- **List** es una **interfaz** utilizada para definir listas doblemente enlazadas
- Importa la posición de los objetos y se pueden recolocar
- Deriva de **Collection** con lo que dispone de todos sus métodos y añade otros más potentes

Interfaz List<E>

- **Interfaz List** (java.util). Deriva de **Collection**.
- Los objetos de esta lista se pueden recolocar.

Método	Uso
void **add** (int indice, E elemento)	**Añade** el elemento indicado en la posición de la lista
void **remove** (int indice)	**Elimina** el elemento cuya posición en la colección coincide con el parámetro **índice**
E **set** (int indice, E elemento)	**Sustituye** el elemento número **índice** por uno nuevo. **Devuelve además el elemento antiguo**
E **get** (int indice)	Obtiene el **elemento almacenado** en la colección en la posición que indica el índice
int **indexOf** (Object elemento)	Devuelve la **posición del elemento** o -1 si no lo encuentra
int **lastIndexOf** (Object elemento)	Devuelve la **posición del elemento comenzando a buscarle por el final** o -1 si no lo encuentra
void **addAll** (int indice, **Collection**<? extends **E**> c)	Añade todos los elementos de una colección a una posición dada.
ListIterator **listIterator**()	Obtiene el iterador de lista que permite recorrer los elementos de la lista.
ListIterator **listIterator** (int indice)	Obtiene el iterador de la lista que permite recorrer los elementos de la lista, que se coloca inicialmente apuntando al elemento cuyo índice en la colección es el indicado
List **subList** (int desde, int hasta)	Obtiene una lista con los elemento que van de la posición desde a la posición hasta

Interfaz ListIterator<E>

- **ListIterator**. Hereda de **Iterator**.
- Pueden recorrer la lista en las dos direcciones.

Método	Uso
void **add** (E elemento)	**Añade** el elemento delante de la posición actual del iterador
void **set** (E elemento)	**Sustituye** el elemento señalado por el iterador, por el elemento indicado.
E **previous**()	**Obtiene** el elemento previo al actual. Si no lo hay provoca exception **NoSuchElementException**
boolean **hasPrevious**()	Indica si hay elemento anterior al actualmente señalado por el iterador
int **nextIndex**()	Obtiene el índice del elemento siguiente
int **previousIndex**()	Obtiene el índice del elemento anterior

List<E>

- Clases **ArrayList** y **LinkedList**
  - Representan una colección **basada en índices**, en la que cada objeto tiene asociado un número (índice) según la posición que ocupa dentro de la colección, siendo **0 la posición del primer elemento**.

```
ArrayList<T> coleccion = new ArrayList<T>();
LinkedList<T> colección = new LinkedList<T>();
```

Uso de genéricos <T>

  - **ArrayList**: lista enlazada implementada utilizando un **array de dimensión modificable**.
  - **LinkedList**: implementación de una **lista doblemente enlazada**. Útil para implementar colas y pilas:
    - getFirst(), getLast(), removeFirst(), removeLast(), addFirst(), addLast()

List<E>

- Ejemplo de **ArrayList**

```
ArrayList<String> listaString = new ArrayList<String>(20);
for (int i=0; i<20; i++){
   listaString.add("Cadena número " + i);
}
System.out.println("La colección tiene " + listaString.size() + " elementos");
```

List<E>

- **for each**. Nueva versión de **for** para colecciones:

```
for  ( Object objeto: nombreColección) {
    …
}
```

```
//suponiendo que colecciónString es una colección de textos
for  ( Object o: colecciónString ) {
    String s = (String)o;
    System.out.println(s);
}
```

ArrayList

- Implementa la **interfaz List.**
- Para listas **simple/doblemente enlazadas.**
- **Definición:**

```
ArrayList<tipo> nombre= new ArrayList<tipo>();
ArrayList nombre =new ArrayList();
```

- **Constructores:**

```
ArrayList (): crea un ArrayList vacío
ArrayList (int capacidadInicial)
ArrayList (Collection c)
```

```
ArrayList() a = new ArrayList ();
a.add("Hola");
a.add("Adiós");
a.add("Hasta Luego");
a.add(0, "Buenos días");
for (Object o:a){
   System.out.println(o);
}
```

```
package pruebacollection;
import java.awt.Color;
import java.util.*;

public class PruebaCollection {
  private static final String colores[]={"rojo", "blanco", "azul"};

   public PruebaCollection() {
      ArrayList lista=new ArrayList();    //o List lista=new ArrayList();
      lista.add(Color.MAGENTA);           //agregar objetos
      for (int cuenta=0; cuenta< colores.length; cuenta++){
        lista.add (colores[cuenta]);
       }
      lista.add(Color.CYAN);
      System.out.println("\nArrayList:"); //mostrar contenido de lista
      for (int cuenta=0; cuenta< lista.size(); cuenta++){
         System.out.print (lista.get(cuenta)+ " ");
       }
      eliminarObjetosString(lista);       //eliminar los elementos String
      System.out.println("\nArrayList despues de llamar a eliminar ObjetosString:");
                                          //mostrar contenido de lista
      for (int cuenta=0; cuenta< lista.size(); cuenta++)
         System.out.print (lista.get(cuenta)+ " ");
   }
```

```
private void eliminarObjetosString(Collection coleccion){
    Iterator iterator = coleccion.iterator();
/*
 * Obtener iterator para acceder a los elementos de la colección.
 * Iterar mientras coleccion tenga elementos
 * hasNext determina si el objeto Collection contiene más elementos, devuelve true si
 * existe otro elemento y devuelve false en caso contrario.
 * next permite obtener una referencia al siguiente elemento, después de la instancia
 * instanceof para determinar si el objeto es de tipo String, de ser así llama al método
 * remove de Iterato para eliminar dicho elemento de la colección.
 */
    while (iterator.hasNext())
      if (iterator.next() instanceof String)
        iterator.remove();              //elimina objeto String
  }

  public static void main(String[] args) {
     PruebaCollection pruebaCollection = new PruebaCollection();
  }
}
```

ArrayList

- **Ejercicio1**: Construir una lista de números utilizando la **clase ArrayList**, realizando la típicas operaciones (alta, baja, búsqueda…)
- **Ejercicio2**: Construir la **guía de teléfonos usando la clase ArrayList**, realizando las siguientes operaciones: alta, baja, búsqueda de un elemento, vaciarla.

LinkedList <E>

- Heredera de las anteriores e implementa.
- **Listas dobles**: permiten insertar por el principio y el final.
- Con esta lista se implementan **colas** y **pilas**.

Método	Uso
E **getFirst**()	Obtiene el primer elemento de la lista (E)
E **getLast**()	Obtiene el último elemento de la lista
void **addFirst**()	Añade el objeto al principio de la lista
void **addLast**()	Añade el objeto al final de la lista
E **removeFirst**()	Borra el primer elemento
E **removeLast**()	Borra el último elemento

LinkedList<E>