{PROG} dit UPM

Programación
Tema 9: Bibliotecas y Colecciones

Contenidos

- **Bibliotecas**
- Concepto de colección
- Definición y uso de lista (List, ArrayList)
- Recorridos sobre colecciones
- Conjunto (Set, HashSet)
- Clases auxiliares-Arrays

Bibliotecas

- Biblioteca (*library*): concepto genérico de programación para referirse a un agrupamiento y encapsulación de código que facilita la reutilización de software.
- En java, las bibliotecas se denominan paquetes (*packages*):
  - Agrupan clases útiles relacionadas entre sí.

{PROG} dit UPM

Bibliotecas

- Las bibliotecas de java:
  - Incorporan miles de clases y decenas de miles de métodos
  - Un programador Java competente debe ser capaz de trabajar con las bibliotecas.
    - Conocer algunas clases importantes por su nombre
    - Saber cómo encontrar otras clases útiles cuando se necesiten.
    - Basta conocer la interfaz, no la implementación.

{PROG} dit UPM

Bibliotecas

- Biblioteca estándar de java
  (API: Application Programmers' Interface):
  - Documentación en formato HTML
  - Descripción de la interfaz (información pública) de las clases:
    - nombre de la clase
    - descripción general de la clase
    - lista de constructores y métodos
    - resultados y parámetros de constructores y métodos
    - descripción del propósito de cada constructor y método
  - No incluye información de implementación:
    - campos/métodos privados
    - cuerpos (código fuente) de cada método

{PROG} dit UPM

Bibliotecas

- Uso de las bibliotecas de clases:
  - Deben ser importadas con una sentencia `import` (excepto las clases de java.lang).
  - Pueden ser usadas como clases del proyecto actual.
    - Se pueden importar clases concretas:
      `import java.util.ArrayList;`
    - Importar paquetes enteros:
      `import java.util.*;`
    - Static import: permite referirse a atributos/métodos definidos en una clase como `public static` sin especificar la clase

{PROG} dit UPM

Bibliotecas

```
public class Circulos {
    public static void main(String[] args) {
        System.out.println(”Un círculo de 5 cm de radio, tiene:");
        System.out.println(”Long.Circunf.: " + (2 * Math.PI * 5) + " cm");
        System.out.println(”Área: “ +(Math.PI * Math.pow(5,2))+ ” cm2");
    }
}

import static java.lang.Math.*;
import static java.lang.System.out;
public class HelloWorld {
    public static void main(String[] args) {
        out.println(”Un círculo de 5 cm de radio, tiene:");
        out.println(”Long.Circunf.: " + (2 * PI * 5) + " cm");
        out.println(”Área: “ +(PI * pow(5,2))+ ” cm2");
    }
}
```

{PROG} dit UPM

Contenidos

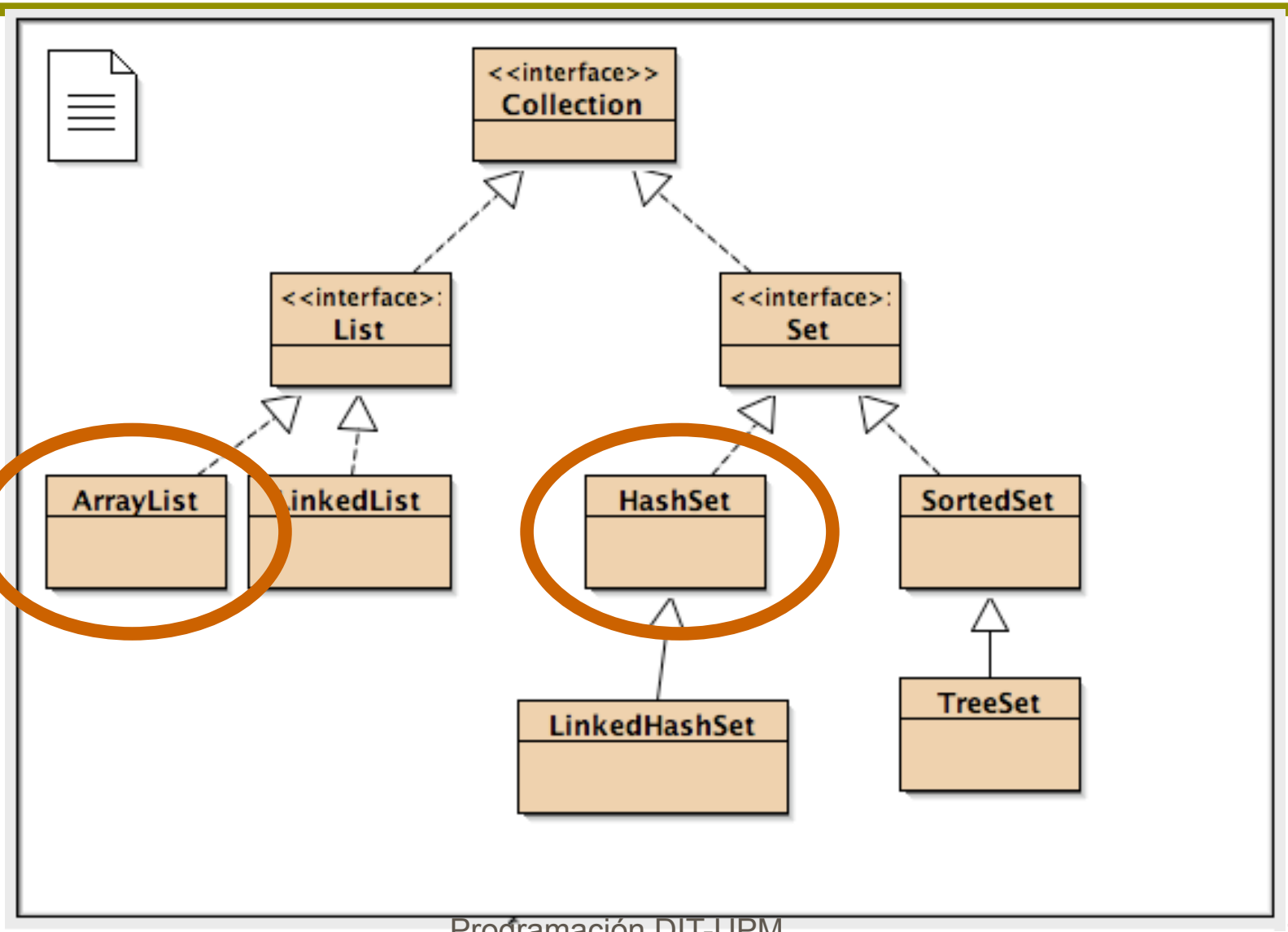
- Bibliotecas
- **Concepto de colección**
- Definición y uso de lista (List, ArrayList)
- Recorridos sobre colecciones
- Conjunto (Set, HashSet)
- Clases auxiliares-Arrays

{PROG} dit UPM

Colecciones

- Los programas manejan gran cantidad de datos del mismo tipo
- **Los arrays** permiten hacerlo, pero **tienen tamaños fijos**
- **Las colecciones** son clases predefinidas que **permiten almacenar datos del mismo tipo**
  - **El tamaño puede variar** –no predeterminado
  - Se pueden manejar estos datos de formas diferentes (Lista, Conjunto, Mapa)
  - **Trabajan sólo con objetos**
    - Importante: Clases envoltorio para tipos primitivos

Jerarquías de colecciones



Colecciones genéricas

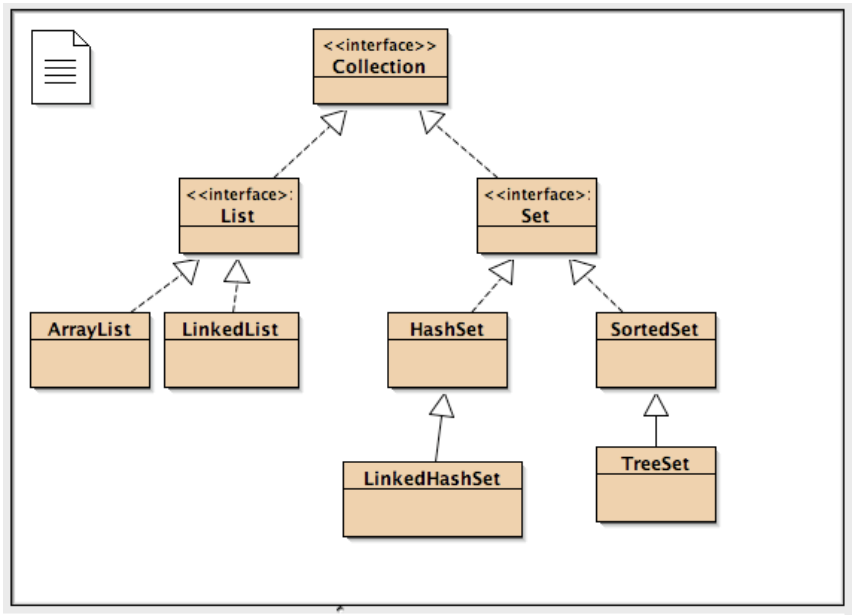
- Las clases e interfaces de colecciones son **genéricas**: valen para cualquier tipo de objetos
- Pero hay que **indicar el tipo concreto** de objetos: el tipo de objetos es un parámetro especial (se marca entre ángulos)
  - **List<E>** // **Lista de elementos de tipo E**
  - **List<Punto>** camino = null; // **lista de puntos**
- El compilador se asegura de que en una colección **sólo se manejan objetos del tipo indicado**

Colecciones más importantes

- package java.util
  - interface List<E>
    - class ArrayList<E> implements List<E>
  - interface Set<E>
    - class HashSet<E> implements Set<E>
  - interface Map<K, V>
    - class HashMap<K,V> implements Map<K,V>

Métodos de la interfaz Collection<E>

- boolean add(E elemento)
- void clear()
- boolean contains(E elemento)
- boolean equals (Object x)
- boolean isEmpty()
- Iterator<E> iterator()
- boolean remove(E elemento)
- int size()
- Object[] toArray()
- …



{PROG} dit UPM

Contenidos

- Bibliotecas
- Concepto de colección
- **Definición y uso de lista** (List, ArrayList)
- Recorridos sobre colecciones
- Conjunto (Set, HashSet)
- Clases auxiliares-Arrays

interface List<E>

- Lista de objetos de tipo E
  - **se respeta el orden** en el que se insertan
  - **admite duplicados**
  - los datos están **indexados**
  - el **tamaño se adapta dinámicamente** a lo que sea necesario
    - parecido a un array de tamaño dinámico

interface List<E>

- boolean add(E elemento)
- void add(int posicion, E elemento)
- void clear()
- boolean contains(E elemento)
- boolean equals(Object x)
- E get(int posicion)
- int indexOf(E elemento)
- boolean isEmpty()
- Iterator<E> iterator()
- E remove(int posicion)
- boolean remove(E elemento)
- E set(int posicion, E elemento)
- int size()
- Object[] toArray()

[En rojo: pueden lanzar excepciones]

Implementaciones de List<E>

- ArrayList<E>
  - array dinámico
  - Usa poca memoria
  - Recorridos rápidos, inserción y borrado lento
- LinkedList<E>
  - lista encadenada
  - Usa memoria extra
  - Recorrido lineal rápido, otros lentos
  - inserciones y eliminaciones internas rápidas

class ArrayList<E> implements List<E>

- Crear una lista de Strings

```
ArrayList<String> notes = new ArrayList<String>();
```

- Con upcasting

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

- Necesario especificar:
  - **el tipo de la colección**: **ArrayList**
  - **el tipo de los objetos** que contendrá: **<String>**

Uso de la colección

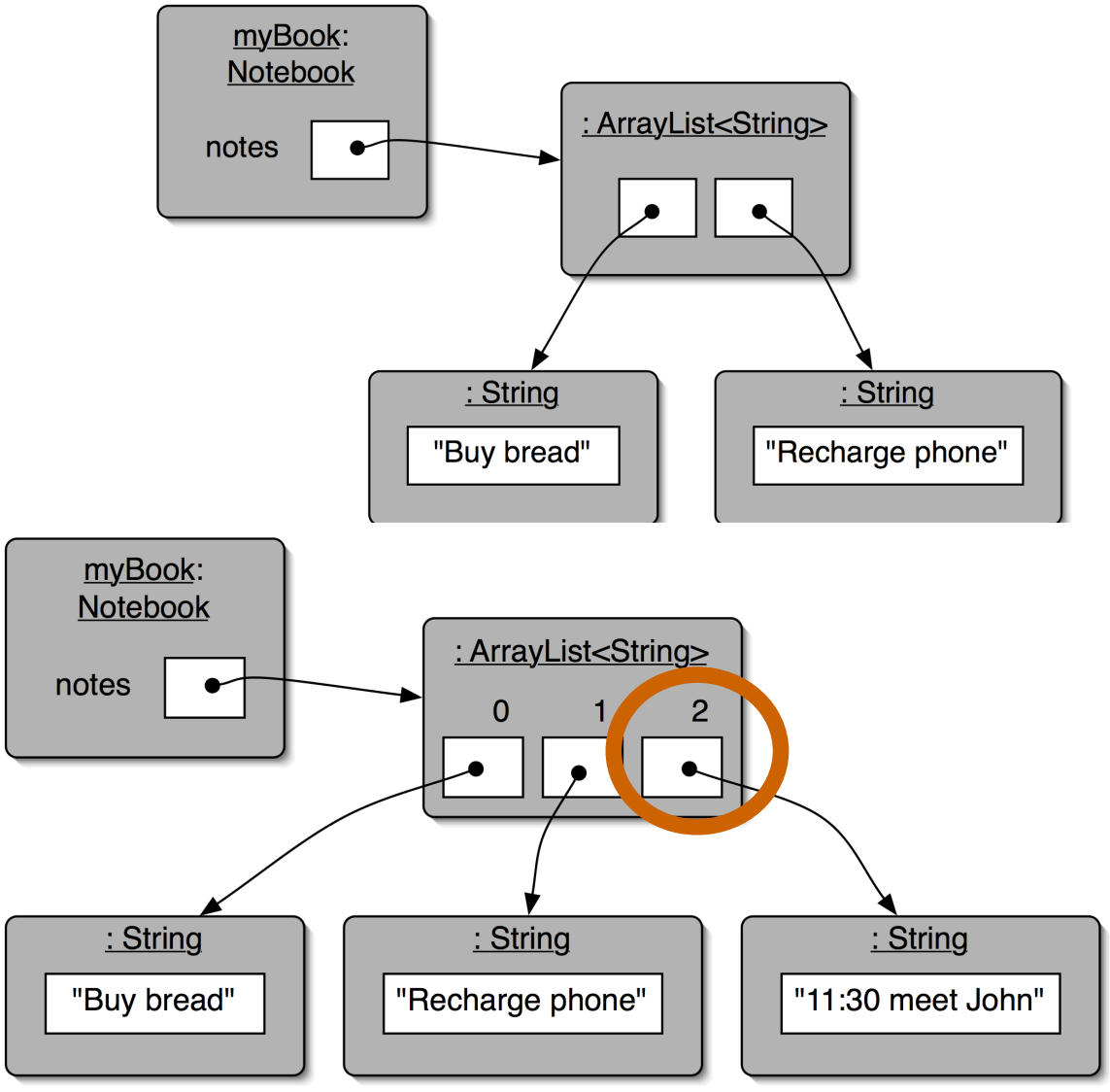
```
public class Notebook {
    private ArrayList<String> notes;
    …
    public Notebook () {
        notes = new ArrayList<String>();
    }
    public void almacenaNota(String nota) {
        notes.add(nota);
    }
    public int numeroDeNotas() {
        return notes.size();
    }
    …
}
```

Creación de la lista

Añadiendo una nota nueva

Devuelve el número de notas

Objetos de la colección



Recuperación de objetos

```
public void muestraNota(int numeroNota)
{
    if(numeroNota < 0) {
        // Esto no es un número de nota válido.
    }
    else if(numeroNota < numeroDeNotas ()) {
        System.out.println(notes.get(numeroNota));
    }
    else {
        // Esto no es un número de nota válido.
    }
}
```

Comprueba la validez del índice

Recupera e imprime la nota

Borrado de objetos

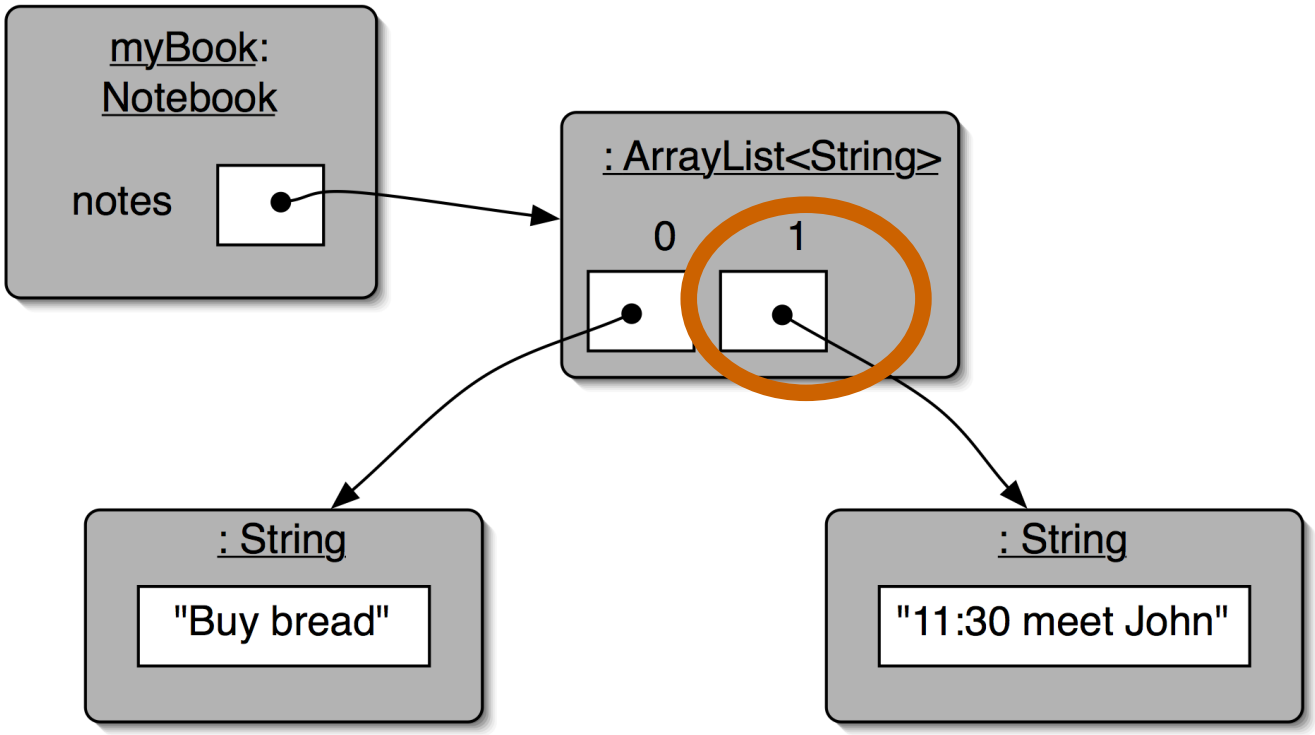
```
public void eliminaNota (int posicion) {

        if (posicion <0 ) {
                // posicion incorrecta, no hace nada

        } else if (posicion < numeroDeNotas() ) {
                notes.remove(posicion);
        } else {
                // posicion incorrecta, no hace nada
        }
}
```

Comprueba la validez del índice

Elimina el objeto por su índice

remove: reasigna índices!

remove(índice)



Ejemplos

```
List <Integer> lista= new ArrayList <Integer> () ;
lista.add(1);
lista.add(9);
lista.add(1, 5);
System.out.println(lista.size());          // 3
System.out.println(lista.get(0));          // 1
System.out.println(lista.get(1));          // 5
System.out.println(lista.get(2));          // 9
for (int n: lista)
        System.out.print(n);               // 1 5 9
```

{PROG} dit UPM

Listas - arrays

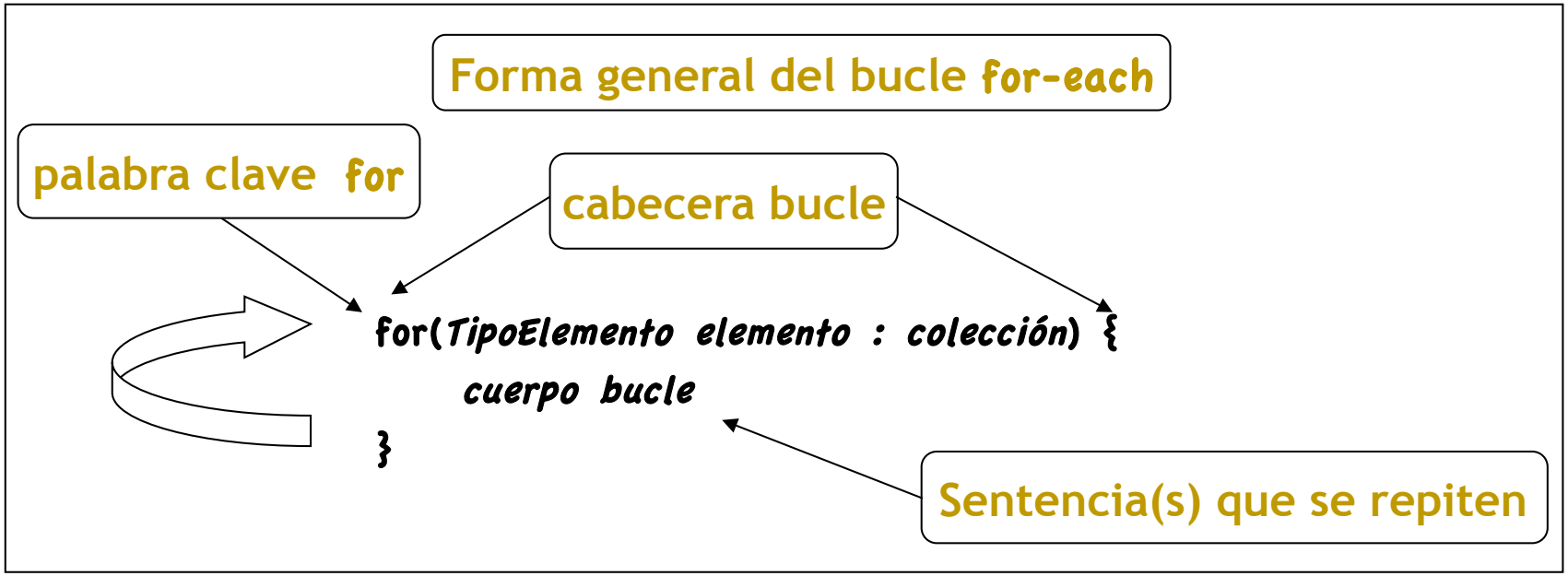
Listas	Arrays
`ArrayList<E> x;`	`E[] x;`
`x = new ArrayList<E>()`	`x = new E[n]`
`x.add(valor)`	-
`x.size()`	`x.length`
`x.remove(n)`	-
`x.get(n)` `x.set(n,v)`	`x[n]` `x[n] = v`
Tamaño variable	Tamaño fijo: *n*

{PROG} dit UPM

Contenidos

- Bibliotecas
- Concepto de colección
- Definición y uso de lista (List, ArrayList)
- **Recorridos sobre colecciones**
- Conjunto (Set, HashSet)
- Clases auxiliares-Arrays

Bucle **for-each**



- Para cada ***elemento*** de una ***colección*** de objetos de ***TipoElemento***,
  haz las acciones del ***cuerpo del bucle***

Recorridos de colecciones

```
// Para recorrer cualquier colección
// for-each
public void listaNotas(){
     for(String nota : notes) {
          System.out.println(nota); }
}
```

```
// Para colecciones indexadas
public void listaNotasForNormal(){
     for (int i = 0; i < notes.size(); i ++) {
          System.out.println( notes.get(i) ); }
}
```

```
// Para colecciones indexadas también bucle while
public void listaNotasWhile() {
     int i = 0;
     while (i< notes.size()) {
          System.out.println( notes.get(i) );
          i++; }
}
```

{PROG} dit UPM

Contenidos

- Bibliotecas
- Concepto de colección
- Definición y uso de lista (List, ArrayList)
- Recorridos sobre colecciones
- **Conjunto** (Set, HashSet)
- Clases auxiliares-Arrays

interface Set<E>

- **No permite elementos repetidos** (un elemento pertenece o no al conjunto)
- **No tienen orden**
- El **tamaño se adapta** dinámicamente

interface Set<E>

- boolean add(E elemento)
- void clear()
- boolean contains(E elemento)
- boolean equals(Object x)
- boolean isEmpty()
- Iterator<E> iterator()
- boolean remove(E elemento)
- int size()

[En rojo: pueden lanzar excepciones]

Implementaciones de Set<E>

- class HashSet<E> implements Set<E>
  - económica en tiempo y memoria
- class TreeSet<E> implements Set<E>
  - más lenta y voluminosa
  - cuando se recorre los elementos salen ordenados

Ejemplo de Set<E>

```
Set<Integer> conjunto = new HashSet<Integer> () ;
conjunto.add(1);
conjunto.add(9);
conjunto.add(5);
conjunto.add(9);
System.out.println(conjunto.size());          // 3
for (int n: conjunto)
   System.out.println(n);     // 9 1 5 (en cualquier orden)
```

Elemento Integer duplicado no se incluye de nuevo

Bucle for-each es el único que puede usarse

{PROG} dit UPM

Contenidos

- Bibliotecas
- Concepto de colección
- Definición y uso de lista (List, ArrayList)
- Recorridos sobre colecciones
- Conjunto (Set, HashSet)
- **Clases auxiliares-Arrays**

interface Map<K, V>

- Un mapa es una **colección de pares de valores**
- El primer elemento del par es la clave (key) y el segundo el valor (value)
- **La clave de tipo K no puede estar repetida**
- Para **acceder a un valor** hay que **dar la clave**
- El tamaño se adapta dinámicamente
- También se llaman **diccionarios**

interface Map<K, V>

- void clear()
- boolean containsKey(Object clave)
- boolean containsValue(Object valor)
- boolean equals(Object x)
- V get(Object clave)
- boolean isEmpty()
- Set<K> keySet()
- V put(K clave, V value)
- V remove(Object clave)
- int size()
- Collection<V> values()

[En rojo: pueden lanzar excepciones]

Implementaciones de Map<K, V>

- HashMap
  - económica
- LinkedHashMap
  - respeta el orden de inserción
  - voluminosa
- TreeMap
  - claves ordenadas
  - lenta y voluminosa

Ejemplo

```
Map <String, String> mapa= new HashMap <String, String> () ;
mapa.put("uno", "one");
mapa.put("dos", "two");
mapa.put("tres", "three");
mapa.put("cuatro", "four");
mapa.put("tres", "33");
System.out.println(mapa.size());
for (String clave: mapa.keySet()) {
     String valor= mapa.get(clave);
     System.out.println(clave + " -> " + valor);
}
```

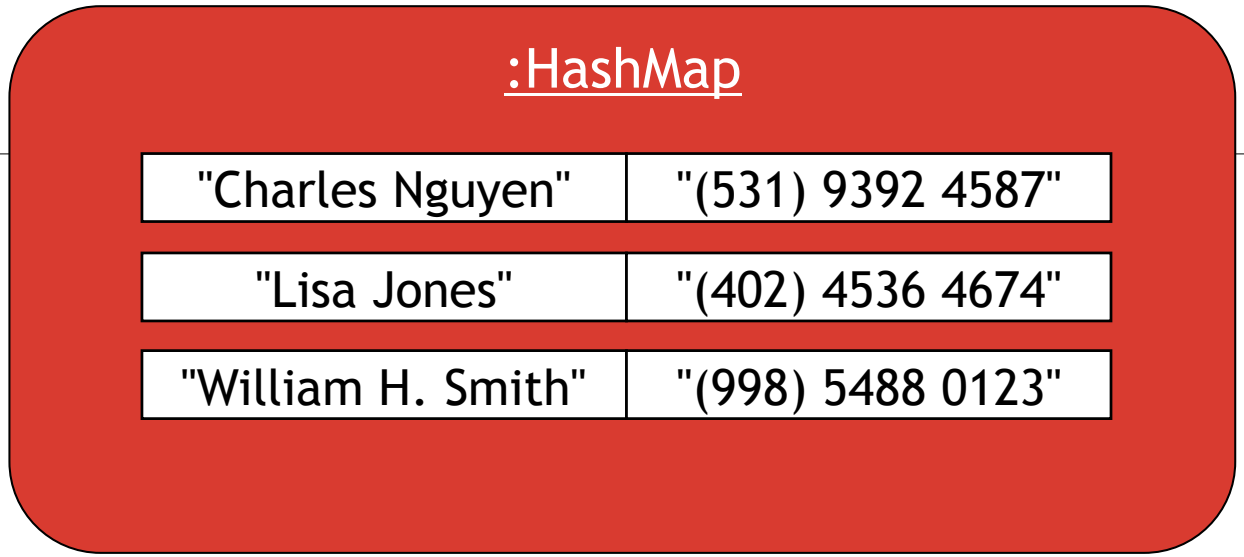
Elemento "tres" duplicado no se incluye de nuevo PERO cambia su valor!

```
4
cuatro -> four
tres -> 33
uno -> one
dos -> two
```

{PROG} dit UPM

Uso de mapas

```
HashMap <String, String> phoneBook
                        = new HashMap<String, String>();
phoneBook.put("Charles Nguyen", "(531) 9392 4587");
phoneBook.put("Lisa Jones", "(402) 4536 4674");
phoneBook.put("William H. Smith", "(998) 5488 0123");

String phoneNumber = phoneBook.get("Lisa Jones");
System.out.println(phoneNumber);
```



class Arrays

- Clase auxiliar con métodos para trabajar sobre arrays de Objects
- Las colecciones tienen un método toArray
- Los elementos deben implementar interface Comparable
  - int compareTo (E e) // -1, 0, 1
- Métodos útiles

```
static int binarySearch (Object[] a, Object key)        //equals
void fill (Object[] a, Object val);
static void sort (Object[] a, int indexPrimer, int indexPenul);
```