

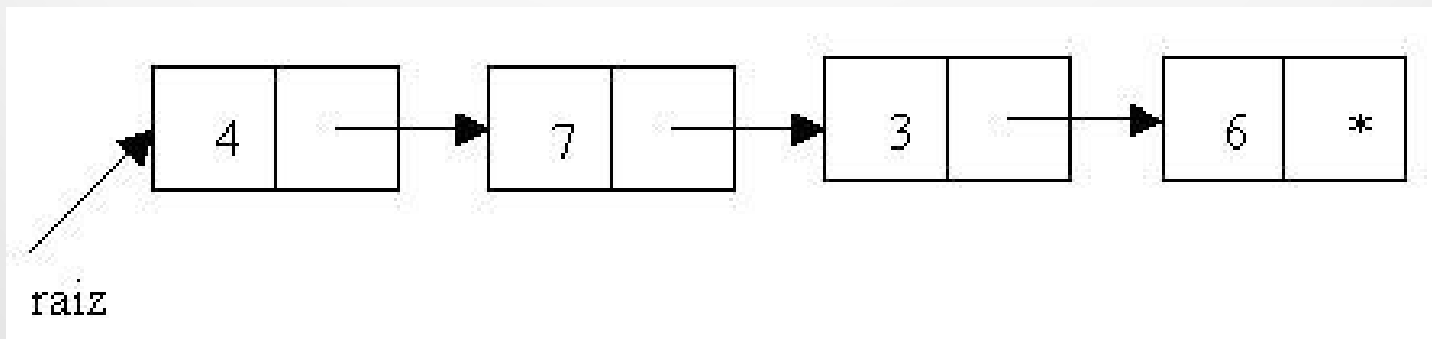


UD9.- Estructures de dades II

Programació – 1r DAW

CONTINGUTS

- Estructures estàtiques *versus* dinàmiques
- Listas
 - Declaració i creació
 - Inserció, modificació i eliminació d'elements
 - Recorregut



Estructures estàtiques *versus* dinàmiques

- En pràcticament tots els llenguatges de programació existeixen estructures per emmagatzemar col·leccions de dades:
 - Un conjunt de dades identificats per un únic nom.
- Estructures **estàtiques**
 - S'ha de saber el nombre d'elements que formaran part de la col·lecció en temps de compilació.
 - Exemple: arrays
- Estructures **dinàmiques**
 - El nombre d'elements es decideix i modifica en temps d'execució.
 - El nombre d'elements és il·limitat.
 - Són clàssiques de la programació i són les coles, les piles, les llistes enllaçades, els arbres, els grafs, etc.
 - En java son implementades a partir de la interfície Collection

Interfícies (I)

- És un element de Java que indica què s'oferix, però no com es fa. Definix un COMPORTAMENT.
- Una interfície proporciona:
 - Valors constants, que són variables “public static final”.
 - Mètodes, que són “public”.
 - NO inclou constructors.

Interfícies (II)

- Quan emprar interfícies?
 - Quan sabem el que volem però no sabem (encara) com fer-ho.
 - Ho farà un altre.
 - Ho farem de varies formes.

Interfícies (III)

- Una classe implementa una interfície quan proporciona codi concret per als mètodes definits en la interfície.
 - D'una mateixa interfície poden derivar-se vàries implementacions.
 - Una mateixa classe pot implementar vàries interfícies (implementació múltiple).
 - Si una classe no implementa tots els mètodes definits en una interfície sinó sols una part, el resultat és una classe abstracta (implementació parcial).

Col·leccions (I)

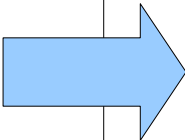
- Interfícies principals:

Collection	Conté la definició de tots els mètodes genèrics que han d'implementar les col·leccions.
-------------------	---

List	Col·lecció d'objectes amb una seqüència determinada.
Set	Col·lecció d'objectes on no s'admeten duplicats.
Map	Emmagatzema parelles d'objectes (clau-valor).

Col·leccions (II)

- Interfícies i implementacions

			Implementacions				
			Tabla Hash	Array redimensionable	Arbre balancejat	Llistes enllaçades	Tabla Hash + Llistes enllaçades
Interface	Collection	Set	HashSet		TreeSet		LinkedHashSet
		List	 ArrayList			LinkedList	
		Map	HashMap		TreeMap		LinkedHashMa p

La classe *ArrayList*

- La classe *ArrayList* permet l'emmagatzemament de dades en memòria de forma semblant als *arrays* convencionals però amb un gran avantatge: la quantitat d'elements que pot guardar és dinàmica.
 - La quantitat d'elements d'un *array* convencional està limitat pel número indicat en el moment de la seua creació o inicialització.
 - Els *ArrayList* poden guardar un número variable d'elements sense estar limitat per un número prefixat.
- Forma part del paquet `java.util.ArrayList`

Declaració d'un ArrayList

- De forma genèrica: `ArrayList nomLlista;`
 - De esta forma no s'especifica el tipus de dades. Això ens permet llistes heterogènies, però és necessari fer un *casting* en la recuperació dels elements.
- És recomanable especificar el tipus de dades que contindrà la llista. Així s'utilitzaran les operacions i mètodes adequats per al tipus de dades concret.
- Per especificar el tipus de dades: `ArrayList<nomClasse> nomLlista;`
 - En cas de guardar dades d'un tipus primitiu de Java (char, int, double, etc...), s'ha d'especificar el nom de la classe associada (*wrapper class*): Character, Integer, Double, etc.
 - Exemples:
 - `ArrayList<String> paisos;`
 - `ArrayList<Integer> edats;`

Creació d'un objecte *ArrayList* (I)

- En dos passos:

```
ArrayList<NomClasse> nomLlista;  
nomLlista = new ArrayList<NomClasse>();
```

- Es pot declarar la llista a la vegada que es crea:

```
- ArrayList<nomClasse> nomLlista = new ArrayList<nomClasse>();
```

- Exemples:

```
ArrayList<String> països = new ArrayList<String>();  
ArrayList<Alumne> curs6J = new ArrayList<Alumne>();
```

Creació d'un objecte *ArrayList* (II)

- Té 3 constructors:
 - `ArrayList()`: constructor per defecte. Crea una llista buida.
 - `ArrayList(int capacitatInicial)`: crea una llista amb una capacitat inicial indicada.
 - `ArrayList(Collection c)`: crea una llista a partir dels elements de la col·lecció indicada.

Afegir elements al final de la llista

- `boolean add(Object elementAInsertar)`
 - Els elements que es van afegir es col·loquen després de l'últim element.
 - El primer element es col·locarà en la posició 0.
- Exemple:

```
ArrayList<String> països = new ArrayList<String>();
països.add("Espanya"); //Ocupa la posició 0
països.add("França"); //Ocupa la posició 1
països.add("Portugal"); //Ocupa la posició 2

//Es poden crear ArrayList per a guardar dades numèriques
ArrayList<Integer> edats = new ArrayList<Integer>();
edats.add(22);
edats.add(31);
edats.add(18);
```

Afegir elements en una posició determinada

- `void add(int posicio, Object elementAInsertar)`
 - Inserta l'element en la posició indicada i desplaça tots els elements una posició a la dreta.
 - Si s'intenta insertar en una posició que no existeix, es produirà l'excepció `IndexOutOfBoundsException`.
- Exemple:

```
ArrayList<String> paisos = new ArrayList<String>();  
paisos.add("España");  
paisos.add("Francia");  
paisos.add("Portugal");
```

```
//L'ordre fins ara és: España, Francia, Portugal
```

```
paisos.add(1, "Italia");
```

```
//L'ordre ara és: España, Italia, Francia, Portugal
```

Consultar un element d'una llista

- `Object get(int posició)`
 - Permet obtindre l'element guardat en una determinada posició.
 - Exemple:

```
System.out.println(paisos.get(3));  
//Seguint amb l'exemple anterior, mostraria: Portugal
```

Modificar un element de la llista

- `Object set(int posició, Object nouElement)`
 - Permet modificar un element que prèviament ha estat guardat en la llista.
 - El primer paràmetre indica la posició que ocupa l'element a modificar.
 - El segon paràmetre indica el nou element que substituirà a l'anterior.
 - Exemple:

```
paisos.set(1,"Alemania");
```


Buscar un element

- `int indexOf(Object elementBuscat)`
 - Retorna la posició de l'element buscat.
 - Si l'element es troba més d'una vegada, indicarà la posició de la primera aparició.
 - El mètode `lastIndexOf` obté la posició de l'últim element trobat.
 - Si l'element no es troba, retornarà -1
 - Exemple:

```
String paisBuscat = "Francia";  
int pos = paisos.indexOf(paisBuscat);  
if(pos!=-1){  
    System.out.println(paisBuscat + " en la posició: " + pos);  
} else {  
    System.out.println(paisBuscat + " no s'ha trobat");  
}
```

Recórrer una llista (I)

- `size()`
 - Retorna el nombre d'elements de la llista.
 - Exemple:

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

```
for(String pais:paisos) {  
    System.out.println(pais);  
}
```

Recórrer una llista (II)

- També es poden recórrer utilitzant un iterador (en el package java.util).
 - Exemple:

```
Iterator iter = paisos.iterator( );  
while ( iter.hasNext( ) ) { //True si n'hi han més elements  
    System.out.println(iter.next( )); //retorna elemento  
}                               //i apunta al següent
```

Altres mètodes

- `boolean remove(Object o)`
 - Elimina de la col·lecció l'object indicat.
- `void clear()`
 - Borra tot el contingut de la llista.
- `Object clone()`
 - Retorna una còpia de la llista.
- `boolean contains(Object o)`
 - Retorna true si l'element es troba en la llista i false en cas contrari.
- `boolean isEmpty()`
 - Retorna true si la llista està buida.
- `Object[] toArray()`
 - Convertix la llista a un array.

Array versus ArrayList

<i>arrays</i>	<i>List</i>
<code>String[] x;</code>	<code>List<String> x;</code>
<code>x = new String[1000];</code>	<code>x = new ArrayList<String>();</code>
<code>... = x[20];</code>	<code>... = x.get(20);</code>
<code>x[20] = "1492";</code>	<code>x.set(20, "1492");</code>
	<code>x.add("2001");</code>