PROGRAMACIÓN ORIENTADA A OBJETOS

TEMA3: Collections e Interfaces

Manel Guerrero

PROGRAMACIÓN ORIENTADA A OBJETOS

Antes que nada veamos W3H1.

[W1H3] Array de elementos atomicos

```
// Como ya vimos en W1H3
// Declaration
int[] myIntArray = new int[3];
int[] myIntArray = {1,2,3};
int[] myIntArray = new int[]{1,2,3};
// print
Arrays.toString(myIntArray);
    [1, 2, 3]
// are equals?
Arrays.equals(a1, a2)
// sorting
Arrays.sort(a1);
```

[W3H2] Array de objetos

```
[Opiskelija toteuttaa Ficha.java]
// Declaration
Ficha[] arrayFichas = new Ficha[MIDA];
// for
for (int i=0;i<arrayFichas.length;i++) {
    arrayFichas[i] = new Ficha(a,b);
}
// foreach
for (Ficha f : arrayFichas) {
    System.out.print(f + ", ");
}
```

[W3H2] Array de tamaño variable

- Java Collections nos ofrece unas clases de arrays de tamaño variable.
- Con ellas podemos empezar con un array vacio e ir añadiendo y eliminando elementos.
- Las Collections tienen que ser de objetos. No pueden ser de tipos primitivos (int o double).

[W3H2] ArrayList

```
// Declaration: Si es un atributo el new mejor en el constructor
private final ArrayList<Ficha> fichas; // atributo de objeto
fichas = new ArrayList<>(); // new en el constructor
fichas = new ArrayList<Ficha>(); // o mas redundante
// Methods:
fichas.add(new Ficha(a,b));
fichas.add(position, object);
fichas.get(position);
fichas.remove(pos); // A parte de eliminar retorna el objeto
fichas.size()
fichas.isEmpty()
// foreach:
for (Ficha f : fichas) { ... }
```

[W3H2] LinkedList

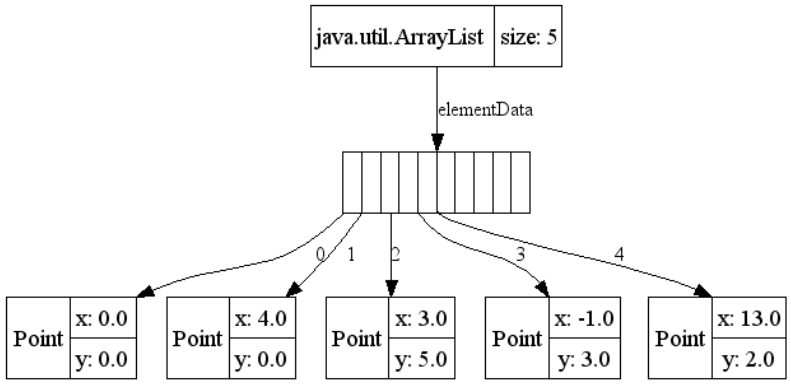
```
// Declaration: Si es un atributo el new mejor en el constructor
private List<Ficha> mesa; // atributo de objeto
pila = new LinkedList<>();  // new en el constructor
pila = new LinkedList<Ficha>(); // o mas redundante
// Methods:
mesa.add(new Ficha(6, 6));
fichas.add(position, object);
pila.get(pos);
fichas.remove(pos); // A parte de eliminar retorna el objeto
fichas.size()
fichas.isEmpty()
// foreach:
for (Ficha f : fichas) { ... }
```

[W3H2] ArrayList vs LinkedList

- LinkedList tiene estos métodos:

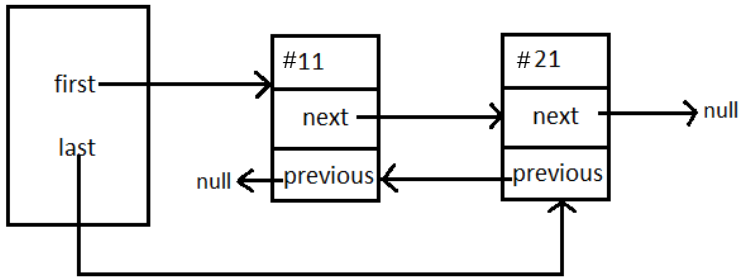
  addFirst(), addLast(), getFirst(), getLast()
- Por cómo están implementados:
  - ArrayList es más eficiente accediendo a una posición del Array.
  - LinkedList es más eficiente insertando y eliminando elementos.

ArrayList



http://www.softwero.com/2012/10/cuando-usar-linkedlist-sobre-arraylist.html

LinkedList



http://www.javamadesoeasy.com/2015/04/arraylist-vs-linkedlist-similarity-and.html

[W3H3] Iterators

```
private List<Ficha> pila = new LinkedList<>();
String s = "";
// El Iterator no se crea, se pide.
Iterator<Ficha> it = pila.iterator();
while(it.hasNext()) { // mientras queden
    s += it.next(); // devuelve el siguiente
}
s += "\n";
// ¿Es feo? ¿Es necesario? Ya tengo el for-each
```

¿Cuando usar Iterators?

http://stackoverflow.com/questions/16000282/why-do-we-need-to-use-iterator-on-arraylist-in-java

> "As you have stated **iterator is used when you want to remove stuff whilst you iterate over the array contents**. If you don't use an iterator but simply have a for loop and inside it use the remove method you will get exceptions because the contents of the array changes while you iterate through. e.g: you might think array size is 10 at the start of the for loop but it wont be the case once you remove stuff.. so when u reach the last loops probably there will be IndexOutofBoundsException etc."

[W3H3] Iterators (2)

```
Iterator<Ficha> it = pila.iterator();
while(it.hasNext()) {
   Ficha f = it.next();
   if (condicion) {
      it.remove(); // remove sobre el iterador
      // (no sobre la pila. pila.remove(f) petaría)
      // y ya no da java.util.ConcurrentModificationException
   }
}
```

¿Cuando usar iterators? (2)

http://www.xyzws.com/javafaq/what-is-the-advantage-of-using-an-iterator-compared-to-the-getindex-method/19

- get(index) en una LinkedList es lento.
- Implementaciones con ArrayList pueden pasar más tarde a usar LinkedList.
- For-each usa un Iterator internamente.
- Mi recomendación: Usar siempre for-each excepto cuando borramos elementos que usaremos Iterator.
- Modificar atributos de un elemento no da problemas de excepciones ya que las referencias no se modifican.

[W3H4] CompareTo Strings

https://www.tutorialspoint.com/java/number_compareto.htm

```
public class Test {
   public static void main(String args[]) {
      String str1 = "Strings are immutable";
      String str2 = new String("Strings are immutable");
      String str3 = new String("Integers are not immutable");
      int result = str1.compareTo( str2 );
      System.out.println(result);
      result = str2.compareTo( str3 );
      System.out.println(result);
   } // Como strcmp() de C: s1.compareTo(s2)>0 -> s1 > s2
} // Imprimira 0 y 10 (puntos frikis: 10 es 'S' – 'I')
```

[W3H4] CompareTo and Interfaces

http://www.javapractices.com/topic/TopicAction.do?Id=10

- The compareTo method is the sole member of the Comparable interface, and is not a member of Object.

- Implementing Comparable allows:
  - calling Collections.sort and Collections.binarySearch
  - calling Arrays.sort and Arrays.binarySearch
  - using objects as keys in a TreeMap
  - using objects as elements in a TreeSet

The compareTo method needs to satisfy the following conditions

These conditions have the goal of allowing objects to be fully sorted, much like the sorting of a database result set on all fields.

- **anticommutation** : x.compareTo(y) is the opposite sign of y.compareTo(x)
- **exception symmetry** : x.compareTo(y) throws exactly the same exceptions as y.compareTo(x)
- **transitivity** : if x.compareTo(y)>0 and y.compareTo(z)>0, then x.compareTo(z)>0 (and same for less than). IMPORTANT!!!
- **consistency with equals** is highly recommended, but not required : x.compareTo(y)==0, if and only if x.equals(y) ; consistency with equals is required for ensuring sorted collections (such as TreeSet) are well-behaved. boolean x.equals(y) { return (x.compareTo(y) == 0); }

compareTo es un metodo de Comparable

java.lang.String
public **final** class String
extends Object
implements Comparable<String>, ...

java.util.ArrayList<E>
public class ArrayList<E>
extends AbstractList<E>
implements Iterable<E>, Collection<E>, List<E>, ...

java.util.LinkedList<E>
public class LinkedList<E>
extends AbstractSequentialList<E>
implements Iterable<E>, Collection<E>, Deque<E>, List<E>, ...

- “final”: Strings inmutables.
- “extends”: String es un tipo de Object.
- “implements”: String implementa los métodos de la interficie Comparable.
- Iterable es la interficie con los métodos de los iteradores.
- Más adelante hablaremos de las interficies con más detalle.

PROGRAMACIÓN ORIENTADA A OBJETOS

Veamos ahora el resto de los códigos W3.

PROGRAMACIÓN ORIENTADA A OBJETOS

Preguntas