Tecnología Java

El propósito de esta clase es aplicar los conceptos de TAD y Complejidad en Java.

A continuación se mencionan algunas herramientas que nos ayudaran a aplicar estos y otros conceptos que se explicaran a lo largo de materia.

String Builder, String Buffer y String

La clase String es una clase no modificable. Esto quiere decir que cuando se modifica un String se crea un nuevo objeto String modificado a partir del original y el recolector de basura es el encargado de eliminar de la memoria el String original.

Java proporciona las clases StringBuffer y StringBuilder a partir de Java 1.5

En general decidiremos cuando usar String, StringBuilder o StringBuffer según lo siguiente:
- Usaremos String si la cadena de caracteres no va a cambiar.
- Usaremos StringBuilder si la cadena de caracteres puede cambiar y solamente tenemos un hilo de ejecución.
- Usaremos StringBuffer si la cadena de caracteres puede cambiar y tenemos varios hilos de ejecución.

En Programación II utilizaremos StringBuilder, por ser un poco más rápida que StringBuffer, pero más que nada porque no veremos temas de sincronización[1].

Constructores de la Clase StringBuilder
Un objeto de tipo StringBuilder gestiona automáticamente su capacidad
– Se crea con una capacidad inicial.
– La capacidad se incrementa cuando es necesario.

[1] La diferencia entre StringBuffer y StringBuilder es que los métodos de StringBuffer están sincronizados y los de StringBuilder no lo están Por este motivo StringBuilder ofrece mejor rendimiento que StringBuffer y la utilizaremos cuando la aplicación tenga un solo hilo de ejecución.

La clase StringBuilder proporcionan varios constructores, algunos de ellos son:

Constructor	Descripción y ejemplo
StringBuilder ()	Crea un StringBuilder vacío. StringBuilder sb = new StringBuilder ()
StringBuilder(int n)	Crea un StringBuilder vacío con capacidad para n caracteres.
StringBuilder(String s)	Crea un StringBuilder y le asigna el contenido del String s String s = "ejemplo" StringBuilder sb = new StringBuilder (s)

La clase StringBuilder proporcionan métodos para acceder y modificar la cadena de caracteres. Algunos de ellos son:

Método	Descripción
insert(posicion, X)	Inserta X en la posición indicada. X puede ser de cualquier tipo.
setCharAt(posicion, c)	Cambia el carácter que se encuentra en la posición indicada, por el carácter c.
indexOf('caracter')	Devuelve la posición de la primera aparición de carácter
substring(n1,n2)	Devuelve la subcadena (String) comprendida entre las osiciones n1 y n2 - 1. Si no se especifica n2, devuelve desde n1 hasta el final.
delete(inicio, fin)	Elimina los caracteres desde la posición inicio hasta fin.
reverse()	
toString()	

Los pueden consultar todos en la API de Java:
http://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html

Iteradores

Es una herramienta de java (y un patrón de diseño[2]) utilizado para recorrer las colecciones. En java es una ***interfaz***[3] denominada ***Iterator***. Está formado por 3 métodos:

boolean hasNext(): retorna true en caso de haber más elementos y false en caso de llegar al final de iterator
Object next(): retorna el siguiente elemento en la iteración
void remove(): remueve el último elemento devuelto por la iteración

En las clases Vector, ArrayList, HashSet y TreeSet un iterador se consigue a través del método: Iterator iterator(). Se utiliza de la siguiente manera:

```
List lasPersonas = new ArrayList();
lasPersonas.add("Juan");
lasPersonas.add("Pepe");

Iterator it = lasPersonas.iterator();

While(it.hasNext){
String unaPersona = (String) it.next();
```

En las clases HashMap y TreeMap se utiliza de la siguiente manera:

```
SortedMap sm=new TreeMap();
sm.put(3, "Intruccion");
sm.put(2, 5);
Iterator iterator = sm.entrySet().iterator();

while (iterator.hasNext()) {

Object claveValor = iterator.next();
System.out.println("Clave = Valor : "+claveValor;
```

Las clases de usuario también pueden tener iteradores, mas adelante veremos como hacerlo.

[2] No veremos "patrones de diseño" en la materia, para programación II utilizaremos este concepto, como sinónimo de "buena práctica".

[3] Cuando lleguemos a la parte objetos definiremos formalmente que es una interfaz, por ahora asumiremos que es una convención que nos sirve para definir cierto comportamiento esperado.

Iterator vs ForEach

```
ArrayList<String> lista = new ArrayList<String>();
lista.add("Pedro");
lista.add("Olga");
lista.add("Miguel");
lista.add("Antonio");
lista.add("Pedro");

Iterator<String> it = lista.iterator();
while (it.hasNext()) {
	System.out.println(it.next());
}
```

Esto nos presentaría la lista de elementos por pantalla :

```
Pedro
Olga
Miguel
Antonio
Pedro
```

A partir de Java 1.5 existe otra forma de recorrer una lista que es mucho más cómoda y compacta, el uso de bucles ***foreach***[4]. Un bucle foreach se parece mucho a un bucle for con la diferencia de que no hace falta una variable "i":

```
for (String nombre : lista) {
	System.out.println(nombre);
}
```

Sin embargo, tenemos que tener algunas precauciones para el uso del foreach:

```
for (String nombre : lista) {
 if (nombre.equals("Pedro")) {
	lista.remove("Pedro");
}
```

El código no funciona y lanza una excepción :

```
Exception in thread "main" java.util.ConcurrentModificationException
	at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:819)
	at java.util.ArrayList$Itr.next(ArrayList.java:791)
	at com.arquitecturajava.Principal.main(Principal.java:44)
```

[4] http://www.arquitecturajava.com/java-iterator-vs-foreach/

La interface Iterator dispone de un método adicional que permite eliminar objetos de una lista mientras la recorremos (el método remove) :

```
Iterator<String> it= lista.iterator();
while(it.hasNext()) {

String nombre= it.next();
if (nombre.equals("Pedro")) {
       it.remove();
}
```

El resultado será:

```
Olga
Miguel
Antonio
```

Test de unidad

Cuando probamos un programa, lo ejecutamos con unos datos de entrada (casos de prueba) para verificar que el funcionamiento cumple los requisitos esperados. Definimos **prueba unitaria** como la prueba de uno de los módulos que componen un programa.

En los últimos años se han desarrollado un conjunto de herramientas que facilitan la elaboración de pruebas unitarias en diferentes lenguajes. Dicho conjunto se denomina *XUnit*. De entre dicho conjunto, **JUnit**[5] es la herramienta utilizada para realizar pruebas unitarias en Java.

Todas las instalaciones de Eclipse incluyen una copia de la biblioteca, y el menú para agregarla se encuentra en: `Project -> Build Path -> Add libraries -> JUnit -> Version 4.`

Todos los **trabajos prácticos** de programación II contendrán un test obligatorio en formato Junit.

Nota: Que nuestra implementación supere todos los casos de prueba no quiere decir que sea correcta; Solo quiere decir funciona correctamente para los casos de prueba que hemos diseñado

Etiqueta Before

Es probable que en varias de las pruebas implementadas se utilicen los mismos datos. Para evitar tener código repetido en los diferentes métodos de *test*, podemos la etiqeta @Before:
Este método se ejecutará antes de cada prueba (antes de ejecutar cada uno de los métodos marcados con @Test).

[5] https://es.wikipedia.org/wiki/JUnit

Ejemplo[6]:

Imaginemos que tenemos el TAD ColaMensajes al que se pueden añadir una serie de mensajes de texto hasta llegar a un límite de capacidad. Cuando se rebase dicho límite, el mensaje más antiguo será eliminado. Para probar los métodos de esta clase en muchos casos nos interesará tener como entrada una cola llena.

```
public class ColaMensajesTest {

  ColaMensajes colaLlena3;

  @Before
  public void setUp() throws Exception {
    colaLlena3 = new ColaMensajes(3);
    colaLlena3.insertarMensaje("1");
    colaLlena3.insertarMensaje("2");
    colaLlena3.insertarMensaje("3");
  }

  @Test
  public void testInsertarMensaje() {
    List<String> listaEsperada = new ArrayList<String>();
    listaEsperada.add("2");
    listaEsperada.add("3");
    listaEsperada.add("4");

    colaLlena3.insertarMensaje("4");
    assertEquals(listaEsperada, colaLlena3.obtenerMensajes());
  }

  @Test
  public void testNumMensajes() {
    assertEquals(3, colaLlena3.numMensajes());
  }

  @Test
  public void testExtraerMensaje() {
    assertEquals("1", colaLlena3.extraerMensaje());
  }
}
```

Durante el TP explicaremos las particularidades de cada test.

Para obtener más información de cómo hacer los test, ver la API de JUnithttp://junit.sourceforge.net/javadoc/org/junit/package-summary.html

[6] http://www.jtech.ua.es/j2ee/publico/lja-2012-13/sesion04-apuntes.html