

JAVA

String Fechas y Colecciones



TOGETHER. FREE YOUR ENERGIES

Objetivo

- Lograr que aprendamos a programar en Java, hasta el punto de construir aplicaciones reales en la tecnología.



Agenda de hoy

- ✓ Parte 1: Herramientas de base y J2EE

Módulo 2: Java SE




- ✓ Strings
- ✓ Fechas
- ✓ Colecciones – Part I

Strings I



String

Existen diferentes formas de crear Strings, las más comunes son:

-  Constructor por defecto.
String s = new String();
-  Constructor con la cadena como parámetro.
String s = new String("Hola");
-  Asignación de una cadena de caracteres.
String s = "Buen" + "dia";

La primera posición de las letras que componen la cadena es la posición 0.

Una vez creado un objeto String no se pueden cambiar los caracteres que lo componen.

Strings II



String



int compareTo(String other)



boolean startsWith(String prefix)



boolean endsWith(String suffix)



boolean equals(Object other)
boolean equalsIgnoreCase(String other)



String toUpperCase()



String toLowerCase()



int indexOf(String str)



int length()



String trim()

Strings III



StringBuilder

Cuando necesitamos construir un String a partir de Strings más pequeños, es ineficiente utilizar concatenación. Conviene usar para ello un **StringBuilder**.

```
StringBuilder builder = new StringBuilder();
```

```
builder.append(ch); // appends a single character  
builder.append(str); // appends a string
```

```
String completedString = builder.toString();
```

Fechas I



Date

Pertenece al paquete `java.util` y se utiliza para obtener la fecha y hora del sistema.



```
Date miFecha = new Date();
```

Algunos métodos:



boolean after(Date). Si la fecha es posterior a la del parámetro



boolean before(Date). Si la fecha es anterior a la del parámetro.



long getTime(). Obtiene el número de milisegundos desde el 01.01.1970.



void setTime(long). Establece la fecha si indicamos los milisegundos.

Fechas II

La clase **Calendar** es una clase abstracta que contiene métodos para trabajar con fechas, permitiendo obtener datos como el día de la semana, el mes, etc.

También contiene una serie de constantes predefinidas para facilitar su uso.

Algunas de estas constantes son las siguientes:

```
public static final int DATE 5
public static final int MONTH 2
public static final int MONDAY 2
public static final int HOUR 10
public static final int DECEMBER 11
public static final int DAY_OF_WEEK 7
public static final int DAY_OF_MONTH 5
```


Fechas III



GregorianCalendar

Al crear un objeto `GregorianCalendar` sin pasar parámetros a su constructor estamos obteniendo la fecha actual del sistema.

```
GregorianCalendar now = new GregorianCalendar();  
int month = now.get(Calendar.MONTH);  
int weekday = now.get(Calendar.DAY_OF_WEEK);
```

```
now.set(Calendar.YEAR, 2010);  
now.set(Calendar.MONTH, Calendar.APRIL);  
now.set(Calendar.DAY_OF_MONTH, 15);  
  
now.set(2010, Calendar.APRIL, 15);  
now.add(Calendar.MONTH, 3);
```

```
Date time = calendar.getTime();  
calendar.setTime(time);
```

Fechas IV



SimpleDateFormat

DateFormat es una clase abstracta que pertenece al paquete `java.text` y permite dar formato a fechas y horas de acuerdo con parámetros locales.

La clase **SimpleDateFormat** es la única que hereda de `DateFormat` y tiene los métodos `parse(String)` y `format(Date)`, para convertir objetos `java.util.Date` a `String` y al revés.

Fechas V



El método `parse()` puede provocar una excepción si lo que le pasamos no es una fecha en el formato adecuado, por lo que tenemos que capturarla.

```
//Pasamos un Date a String
Date unaFecha = new Date();
DateFormat formateador = new SimpleDateFormat("dd/MM/yyyy");
String fechaString = formateador.format(unaFecha);
System.out.println(fechaString);

//Pasamos un String en a Date
fechaString = "01/09/2006";
try {
    Date fechaDate = formateador.parse(fechaString);
} catch (ParseException e) {
    e.printStackTrace();
}
```

Ejercicio








Ejercicio XI de la guía



Colecciones - Part I I

Colecciones

-  Las colecciones son objetos Java que se utilizan para almacenar y manipular datos.
-  Todas las colecciones se encuentran en el paquete `java.util`
-  La interface `java.util.Collection` es la raíz de todas las colecciones.
-  De esta clase parten diferentes especializaciones que permitirán ordenar o no los elementos, hacer búsquedas, permitir duplicados, etc.
-  Las colecciones no permiten almacenar tipos primitivos, en su lugar tendremos que utilizar las clases wrapper: `Integer`, `Double`,

Colecciones - Part I II

Interface Collection

Operaciones básicas

<code>int size()</code>	Nº de elementos de la colección
<code>boolean isEmpty()</code>	Si está vacía o no.
<code>boolean Contains(Object element)</code>	Si contiene el elemento
<code>boolean add(Object element)</code>	Añadir el elemento a la colección
<code>remove(Object element)</code>	Borrar el elemento
<code>Iterator iterator()</code>	Devuelve una instancia de Iterator

Colecciones - Part I III

Interface Collection

Operaciones masivas

<code>boolean containsAll(Collection c)</code>	Si contiene todos los elementos de c
<code>boolean addAll(Collection c)</code>	Añadir todos los elementos de c
<code>boolean removeAll(Collection c)</code>	Borrar todos los elementos que estén en c
<code>boolean retainAll(Collection c)</code>	Borrar todos menos los que estén en c
<code>void clear()</code>	Borrar la colección

Colecciones - Part I IV

Interface Collection

Interface



Iterator



Permite recorrer todos los elementos que forman una colección.

```
public Iterator iterator();
```

```
Iterator<String> it = miColeccion  
.iterator();
```

```
public interface Iterator<E> {  
    boolean hasNext();  
    E next();  
    void remove();  
}
```

```
while (it.hasNext()) {  
    ...  
}
```

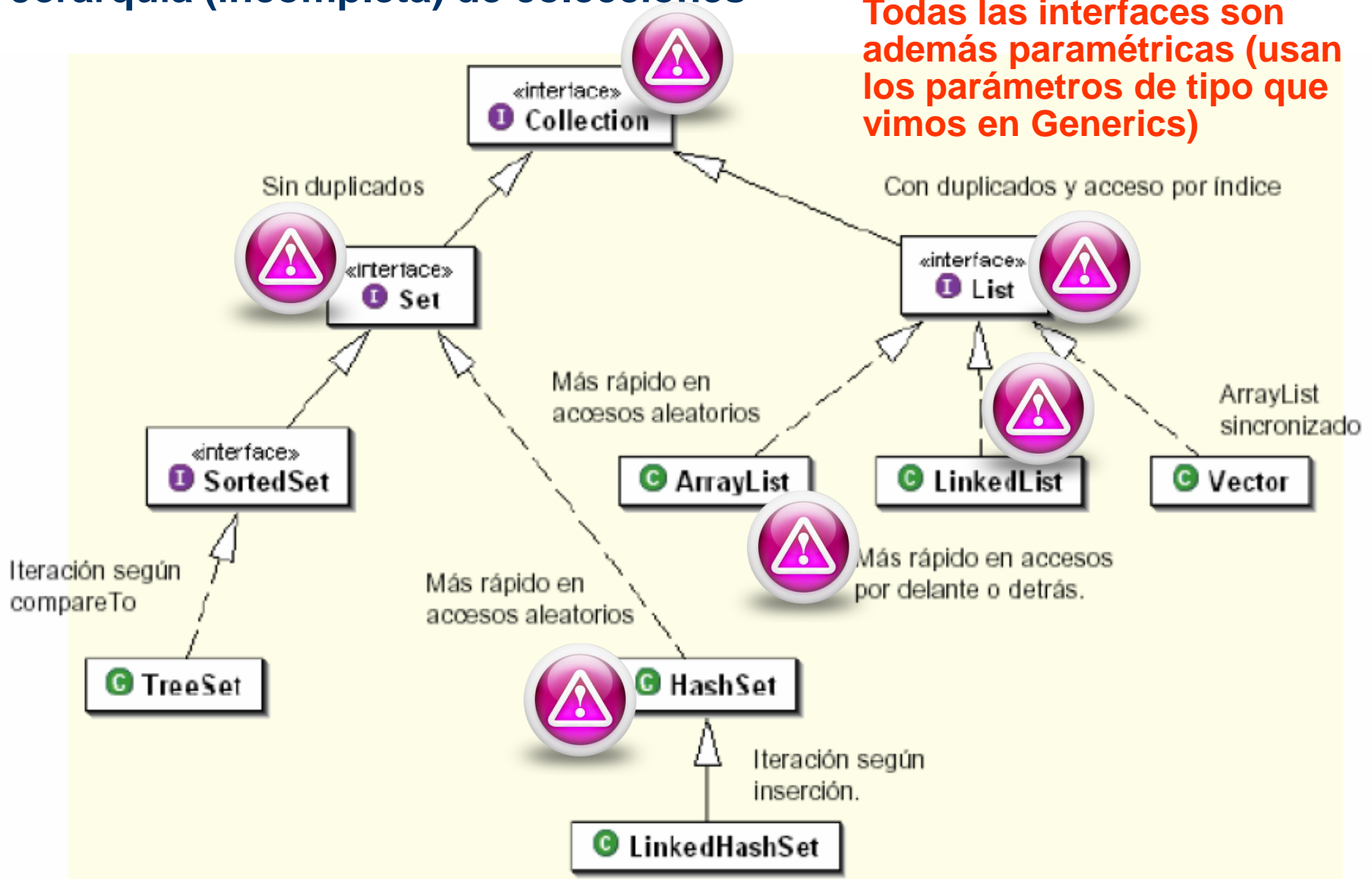
```
String s = it.next();
```

```
it.remove();
```


Colecciones - Part I V

Jerarquía (incompleta) de colecciones

Todas las interfaces son además paramétricas (usan los parámetros de tipo que vimos en Generics)



Colecciones - Part I VI

Collections SIN duplicados

Heredan de la interface Set, que a su vez hereda de Collection.

Algunas de las clases que implementan esta interface son:



HashSet. Ofrece el acceso más rápido si es aleatorio. El orden de iteración es impredecible.



LinkedHashSet. Igual que HashSet pero el orden de iteración es el de inserción.



TreeSet. Su orden de iteración depende de la implementación que sus elementos hagan del método public int compareTo(Object o) (de la interface Comparable).

Colecciones - Part I VII

Collections SIN duplicados

Ejemplo HashSet

```
public static void main(String[] args) {  
  
    Set<String> ciudades = new HashSet<String>();  
  
    ciudades.add("Buenos Aires");  
    ciudades.add("Tucumán");  
    ciudades.add("Rosario");  
    ciudades.add("Buenos Aires");  
    ciudades.add("Bariloche");  
  
    Iterator<String> it = ciudades.iterator();  
  
    while (it.hasNext()) {  
        System.out.println("Ciudad:" + it.next());  
    }  
}
```

Ciudad:Bariloche
Ciudad:Rosario
Ciudad:Tucumán
Ciudad:Buenos Aires

Colecciones - Part I VIII

Collections SIN duplicados

Ejemplo LinkedHashSet

```
public static void main(String[] args) {
```

```
    Set<String> ciudades = new LinkedHashSet<String>();
```

```
    ciudades.add("Buenos Aires");
```

```
    ciudades.add("Tucumán");
```

```
    ciudades.add("Rosario");
```

```
    ciudades.add("Buenos Aires");
```

```
    ciudades.add("Bariloche");
```

```
    Iterator<String> it = ciudades.iterator();
```

```
    while (it.hasNext()) {
```

```
        System.out.println("Ciudad:" + it.next());
```

```
    }
```

```
}
```

Ciudad:Buenos Aires
Ciudad:Tucumán
Ciudad:Rosario
Ciudad:Bariloche

Colecciones - Part I IX

Collections SIN duplicados

Ejemplo TreeSet

```
public static void main(String[] args) {
```

```
    Set<String> ciudades = new TreeSet<String>();
```

```
    ciudades.add("Buenos Aires");
```

```
    ciudades.add("Tucumán");
```

```
    ciudades.add("Rosario");
```

```
    ciudades.add("Buenos Aires");
```

```
    ciudades.add("Bariloche");
```

```
    Iterator<String> it = ciudades.iterator();
```

```
    while (it.hasNext()) {
```

```
        System.out.println("Ciudad:" + it.next());
```

```
    }
```

```
}
```

Ciudad:Bariloche
Ciudad:Buenos Aires
Ciudad:Rosario
Ciudad:Tucumán

Ejercicio



Ejercicio XII de la guía



Muchas gracias!