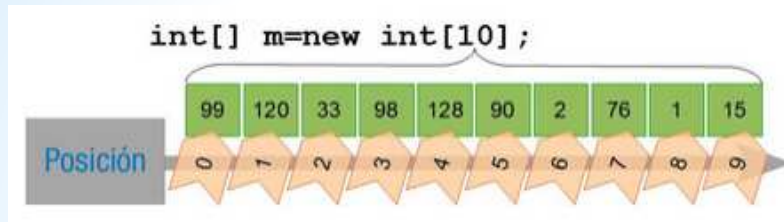




PROGRAMACIÓN

SEGUNDO TRIMESTRE
Tutoría colectiva (III)

ARRAYS



- * Permiten albergar datos del **mismo tipo**.
- * Su longitud se establece durante su creación.
- * Una vez establecida su longitud, ya no se puede modificar.
- * Un elemento de un array es el valor de una de sus posiciones y se puede acceder mediante un índice.
- * En Java es un tipo de clase especial que hereda implícitamente de la clase `java.lang.Object`

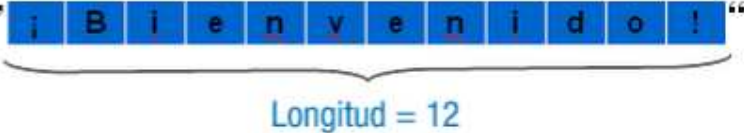
CADENAS DE CARACTERES. Clase String

- * Son estructuras de almacenamiento que permiten almacenar una secuencia de caracteres de casi cualquier longitud.
- * Los literales de cadena son instancias de la clase String.

```
String cad = "¡Bienvenido!";
```

```
String cad = new String ("Bienvenido");
```

```
String cad="¡ B i e n v e n i d o !"
```



Longitud = 12

- * Operaciones con cadenas:

- * Concatenación

```
String cad = "Bien".concat("venido");
```

```
String cad = "Bien" + "vendio" ;
```

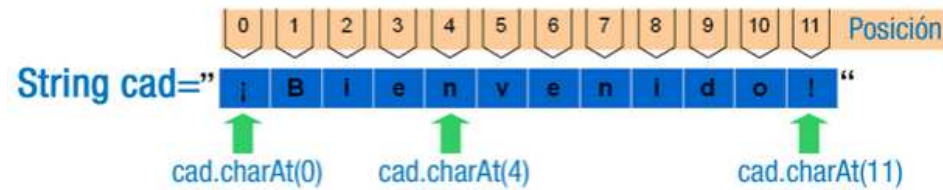
CADENAS DE CARACTERES. Clase String

* `length()`: retorna la longitud de la cadena

```
int longitud = cad.length();
```

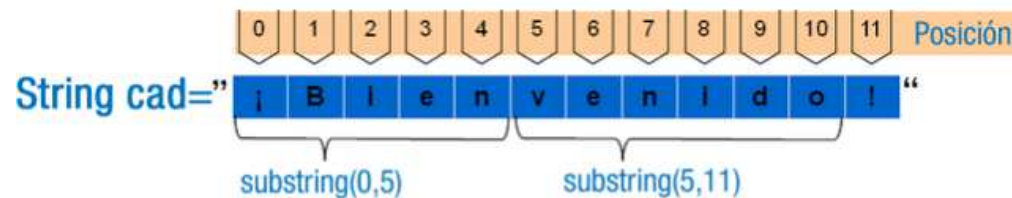
* `charAt(int pos)`:

```
char v = cad.charAt(5);
```



* `substring(inicio,fin)`: extrae un subcadena

```
String cad2 = cad.substring(0,5);
```



CADENAS DE CARACTERES. Clase String

- * Métodos más comunes de la clase String en documento pdf.
- * Transformar cadenas a números: Utilizaremos el método `valueOf` de la clase **envoltorio** correspondiente al tipo primitivo que queremos transformar. O el método `parseXX` correspondiente.

```
String c = "12.34";  
double n1 = Double.valueOf(c).doubleValue();  
double n2 = Double.valueOf(c);  
double n3 = Double.parseDouble(c);
```

CADENAS DE CARACTERES. Clase StringBuffer

- * La clase StringBuffer también nos permite representar cadenas de caracteres. De hecho la forma habitual de construir un objeto de tipo StrinBuffer es a partir de un objeto String.

```
String textolnicial = "Vamos a escribir un texto para procesarlo:";
StringBuffer textoDocumento = new StringBuffer (textolnicial);

//Las sentencias anteriores son equivalentes a la siguiente, que ponemos comentada:

//StringBuffer textoDocumento=new StringBuffer("Vamos a escribir un texto para procesarlo:");
```

- * StringBuffer define objetos que sí pueden ser directamente modificados, sin necesidad de crear un nuevo objeto, ni de actualizar las referencias.

CADENAS DE CARACTERES. Clase StringBuffer

- * Eso es posible porque la clase StringBuffer reserva para cadena de caracteres un espacio extra de almacenamiento, además del estrictamente necesario.
- * Es espacio extra que se puede usar para hacer modificaciones que alteren el tamaño de la cadena de caracteres, siempre y cuando no impliquen necesidades de espacio mayores del espacio extra adicional que reserva. Si fuera así, seguiría siendo necesario realojar el objeto en una nueva zona de memoria, en la que quepan todos los cambios, y actualizar la referencia.
- * Métodos más comunes de la clase StringBuffer en documento pdf.

EXCEPCIONES

- * Es la manera que tiene Java de manejar los errores en tiempo de ejecución.
- * Las excepciones nos permiten indicar el código que se ejecutará en el caso de producirse un error y continuar con la ejecución del programa, evitando de esta forma que se *rompa* la ejecución.
- * Ejemplo de *error de ejecución*: desbordamiento en el acceso a elementos de un array:

```
public class Desbordamiento {  
    public static void main(String[] args) {  
        String mensajes[] = {"Primero", "Segundo", "Tercero"};  
        for (int i=0; i<=3; i++)  
            System.out.println(mensajes[i]);  
    }  
}
```

```
Primero  
Segundo  
Tercero  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3  
    at Desbordamiento.main(Desbordamiento.java:7)
```


EXCEPCIONES.- Jerarquía

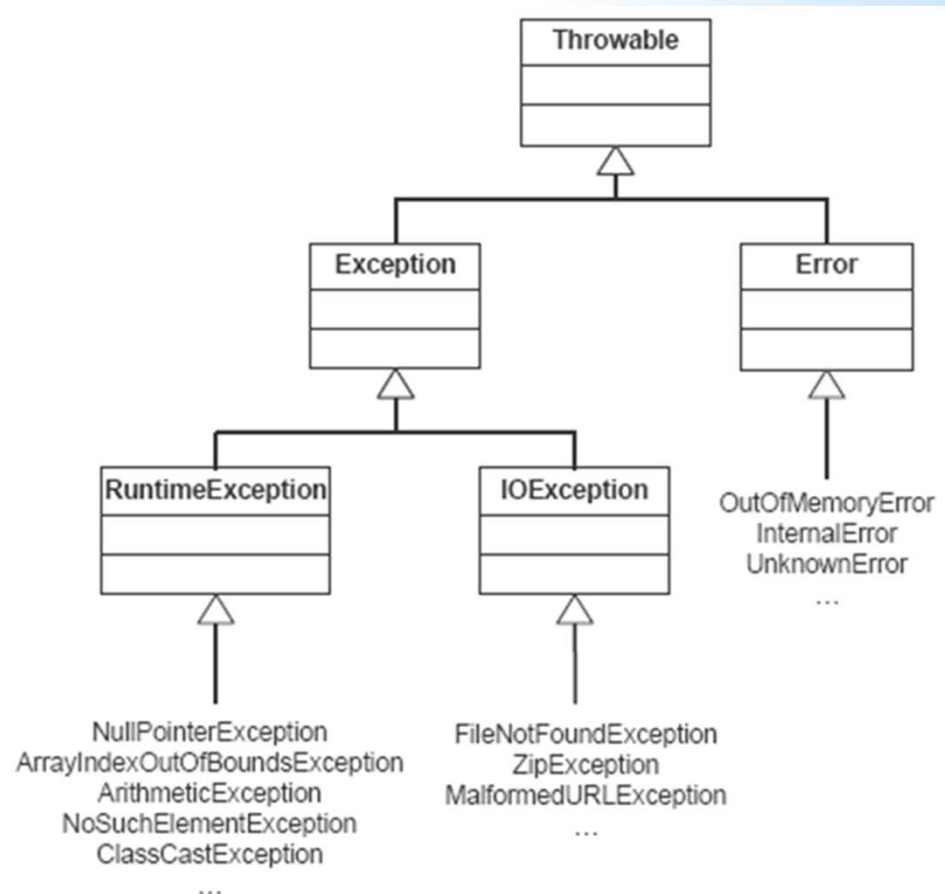
* **Throwable:** Superclase que engloba a todas las excepciones.

* **Error:** Representa a los errores graves provocados en la máquina virtual, no en nuestros programas.

No se controlan en el programa.

* **Exception:** Define las excepciones que los programas deberán controlar.

No es obligatorio hacerlo con las que heredan de *RunTimeException*.



EXCEPCIONES.- Manejo

Bloque try....catch.....finally

- * Se utiliza para detectar “cuándo” se produce una excepción e indicar las instrucciones que se deben ejecutar en ese caso:

```
try {  
    // código que puede generar una excepción  
}  
catch (TipoExcepcion 1 e1) {  
    //manejo de la excepción1  
}  
catch (TipoExcepcion2 e2) {  
    //manejo de la excepción2  
}...  
finally{  
    //instrucciones que se ejecutan siempre  
}
```

- * Si en la ejecución del código dentro del bloque **try** se produce una excepción del tipo indicado (o descendiente de éste), Java omite la ejecución del resto del código en el bloque try y ejecuta el código situado en el bloque **catch**.

EXCEPCIONES.- Manejo

try....catch.....finally

- * **try:** El bloque de código donde se prevé que se pueda generar una excepción. Siempre tiene que ir seguido, al menos de una cláusula **catch** o una cláusula **finally**.
- * **catch:** Es el código que se ejecuta cuando se lanza la excepción. Controla cualquier excepción que coincida con su argumento. Se pueden colocar varios catch sucesivos, cada uno controlando un tipo de excepción diferente.
El último catch debe ser el que capture excepciones genéricas y los primeros deben ser los más específicos.
- * **finally:** Bloque que se ejecuta siempre, haya o no excepción, por lo que se suele utilizar para liberar recursos, como cerrar ficheros, liberar conexiones, ...

EXCEPCIONES.- Ejemplo

- * Ejemplo de error de ejecución: desbordamiento en el acceso a elementos de un array:

```
public class Desbordamiento {  
    public static void main(String[] args) {  
        String mensajes[] = {"Primero", "Segundo", "Tercero"};  
        try {  
            for (int i = 0; i <= 3; i++)  
                System.out.println(mensajes[i]);  
        } catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println ("Estás accediendo a una posición fuera del array");  
        }  
        finally{  
            System.out.println ("Ha finalizado la ejecución del programa");  
        }  
    }  
}
```

Si se produce la excepción que deseamos capturar (o una derivada de la misma) se ejecutará el código que contiene el bloque catch.

```
Primero  
Segundo  
Tercero  
Estas accediendo a una posición fuera del array  
Ha finalizado la ejecución del programa
```

EXCEPCIONES.- Ejemplo

- * Ejemplo: detectar división por cero.

```
public class ExcepcionDividirPorCero {  
    public static void main(String[] args) {  
        try {  
            int a = args.length;  
            System.out.println("a = " + a);  
            int b = 3/a;  
        } catch (ArithmeticException e) {  
            System.out.println ("No dividas entre 0 (" + e.getMessage() +")");  
        }  
        System.out.println ("La ejecución del programa sigue .....");  
    }  
}
```

Toda excepción tiene un método `getMessage()` que devuelve un `String` con los detalles del mensaje.

```
a = 0  
No dividas entre 0 (/ by zero)  
La ejecución del programa sigue ...
```