

DAM PROGRAMACIÓN



UD1. IDENTIFICACIÓN DE UN PROGRAMA INFORMÁTICO



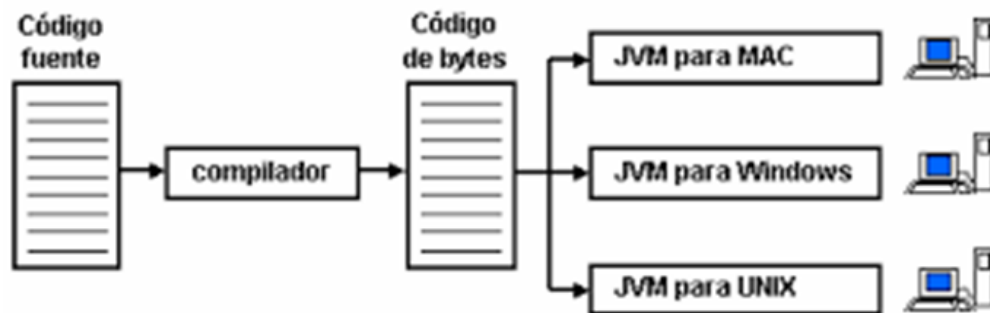
Java

- La tecnología Java (lenguaje y plataforma)
 - Fue desarrollada, en su primera versión, en 1995 por SUN MICROSYSTEMS.
 - Actualmente es propiedad de ORACLE.
- Es un lenguaje de programación y una plataforma de desarrollo y ejecución.
- Es multiplataforma ya que los programas desarrollados en Java se pueden ejecutar en sistemas operativos/hardware muy diverso.
- Es un lenguaje orientado a objetos.
- Tiene utilidad para realizar aplicaciones:
 - En entorno de escritorio.
 - En aplicaciones cliente-servidor en la Web.
 - En dispositivos móviles Android.
- Facilita el acceso a distintas fuentes de datos.
- Es uno de los lenguajes más utilizados por los programadores/as.

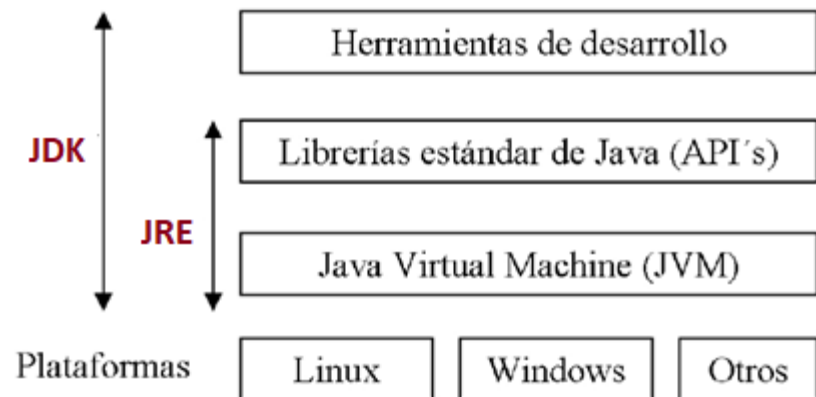
Plataforma Java

- La plataforma Java se ejecuta sobre distintas plataformas hardware.
- En la plataforma Java podemos diferenciar:
 - **JRE (Java Runtime Environment):**
 - Actúa como intermediario entre el sistema operativo donde está instalado y las aplicaciones Java.
 - Su función es ofrecer el *entorno necesario para ejecutar una aplicación* Java.
 - Está formada por:
 - La **máquina virtual Java o JVM** (Java Virtual Machine), encargada de ejecutar el *bytecode* Java,
 - Las **bibliotecas** que ofrecen los servicios definidos en la plataforma que incluye un API como *java.net* o *java.io*.
 - **JDK (Java Development Kit):**
 - Es el *paquete de herramientas necesarias para desarrollar aplicaciones*.
 - **JDK** está compuesto por el
 - **JRE**. Permite ejecutar aplicaciones Java (código *bytecode*)
 - **Compilador Java (*javac*)**. Este toma el código fuente Java y genera como resultado *bytecode*, un formato de código objeto independiente del sistema operativo y el hardware.
 - **Archivador (*jar*)**. Permite comprimir y descomprimir ficheros **jar** que pueden ser programas o bibliotecas con varios archivos.
 - **Visualizador de applets**. Facilita la prueba de applets.
 - **Depurador (*jdb*)**. Permite depurar las aplicaciones Java por comandos.
 - **Generador de documentación (*javadoc*)**. Genera páginas HTML basadas en las declaraciones y comentarios.

Para **ejecutar** un programa Java solo necesitamos el JRE, mientras que para **desarrollar** nuevas aplicaciones es necesario un entorno de desarrollo, denominado JDK, que además del JRE (mínimo imprescindible) incluye, entre otros, un **compilador** para Java.

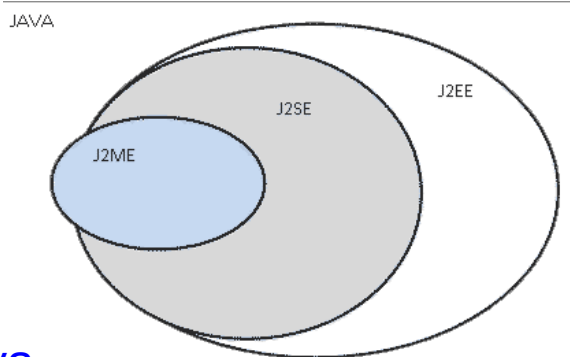


El Java Development Kit (JDK) proporciona el conjunto de herramientas básico para el desarrollo de aplicaciones con Java estándar. Se puede obtener de manera gratuita en internet, descargándola desde el [sitio de Oracle](#).



Plataformas Java

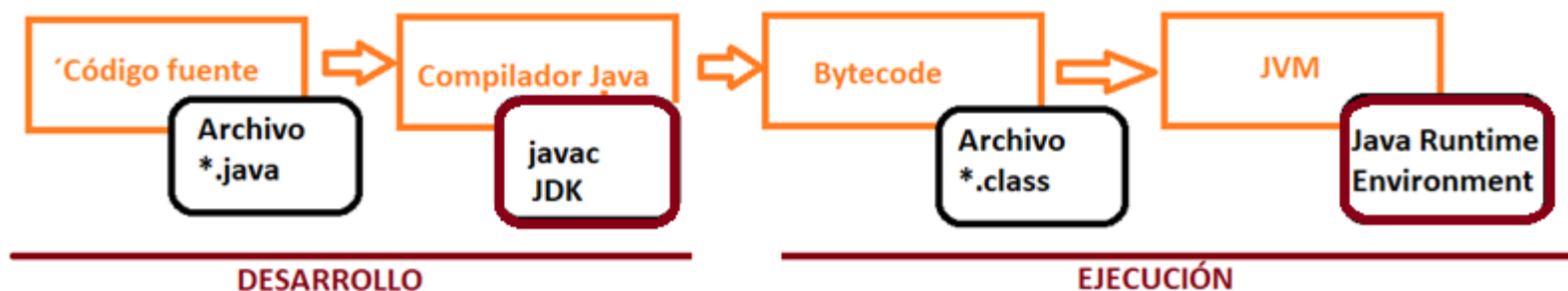
- Hay tres distribuciones del lenguaje de programación Java:
 - **Plataforma Java, Edición Estándar (Java SE).**
 - Orientado al desarrollo de aplicaciones de escritorio cliente/servidor.
 - **Plataforma Java, Edición Enterprise (Java EE).**
 - Está construida sobre la plataforma Java SE.
 - Proporciona una API y un entorno de tiempo de ejecución para desarrollar y ejecutar aplicaciones de red a gran escala, de múltiples niveles, escalables, confiables y seguras
 - **Plataforma Java, Micro Edition (Java ME).**
 - La plataforma Java ME proporciona una API y una máquina virtual de tamaño reducido para ejecutar aplicaciones de lenguaje de programación Java en dispositivos pequeños, como teléfonos móviles



[Instalar la plataforma Java](#)

Programas Java

- Los fuente de Java están organizados en clases que se almacenan en ficheros ***.java**.
- Los fichero ***.java** deben ser compilados para crear los ficheros objeto de la JVM.
- Cuando se compila un fichero fuente *.java se generan uno o varios archivos ***.class**, uno por cada clase que tenga el fichero.
- **Los ficheros *.class:**
 - Se llaman igual que la clase a la que hace referencia cada uno.
 - Están codificados en *bytecode* para poder ser interpretados por cualquier JVM.
- La **JVM** funciona como un intérprete que traduce las instrucciones codificadas en *bytecode* a *código ejecutable* por el sistema operativo en el que se realiza la operación.



VARIABLES DEL SISTEMA

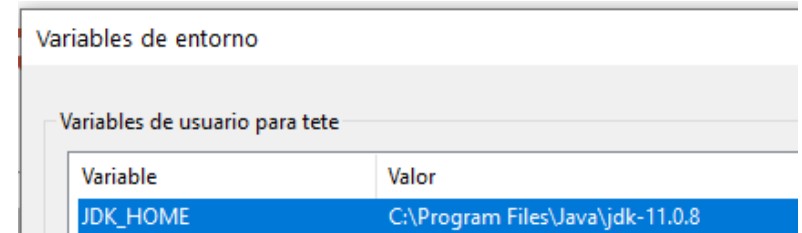
- Para que el sistema pueda acceder a los archivos de Java para compilar y ejecutar los programas desarrollados en Java se debe crear la siguiente variables del sistema:

- **JAVA_HOME**

- Ruta donde está la carpeta **bin** del JDK

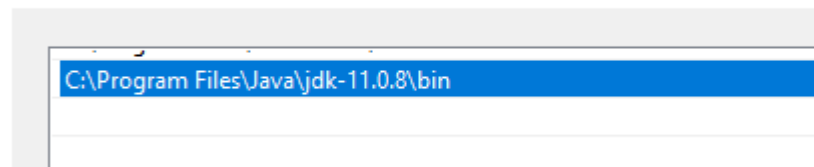
- **JDK_HOME** (usada por algunas aplicaciones)

- Ruta donde está la carpeta **bin** del JDK.



- También es necesario añadir la ruta **%JAVA_HOME%\bin** para poder acceder al fichero **javac.exe** y **java.exe** desde la línea de comandos

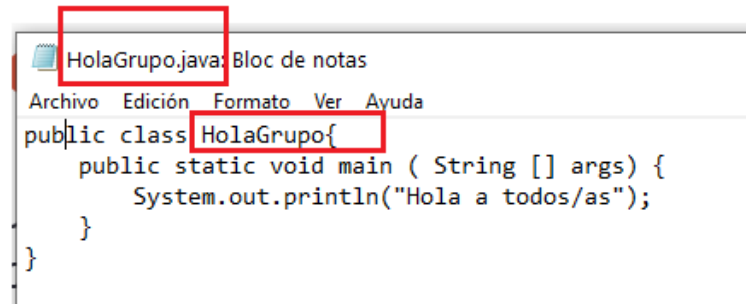
Editar variable de entorno



Primer programa en Java

1. Codificación

- Se puede utilizar cualquier editor de texto plano (*Bloc de notas, Notepad++, gedit, vi, etc.*) escribimos el siguiente código fuente:



```
HolaGrupo.java: Bloc de notas
Archivo Edición Formato Ver Ayuda
public class HolaGrupo{
    public static void main ( String [] args) {
        System.out.println("Hola a todos/as");
    }
}
```

2. Compilación:

- La compilación de un archivo de código fuente ***.java** se realiza utilizando el comando **javac** del **JDK**.
- Como resultado de la compilación obtendremos un fichero **NombreClase.class** por cada clase del fichero compilado (habitualmente, será una clase por fichero).
- Si al realizar la **compilación**, el **fichero fuente** tuviera **algún error** sintáctico, el compilador nos informaría de ello y no generaría el fichero objeto **bytecode** (*.class)

Compilación del código fuente **HOLAGRUPO.JAVA**

```
D:\Dropbox\Dropbox\6_Teis\Programación\Ejercicios>dir
El volumen de la unidad D es DATA
El número de serie del volumen es: 1EF5-DBF6

Directorio de D:\Dropbox\Dropbox\6_Teis\Programación\Ejercicios

11/10/2020  18:28    <DIR>        .
11/10/2020  18:28    <DIR>        ..
11/10/2020  18:28                130 HolaGrupo.java
                1 archivos                130 bytes
                2 dirs  1.651.783.815.168 bytes libres

D:\Dropbox\Dropbox\6_Teis\Programación\Ejercicios>javac HolaGrupo.java

D:\Dropbox\Dropbox\6_Teis\Programación\Ejercicios>dir
El volumen de la unidad D es DATA
El número de serie del volumen es: 1EF5-DBF6

Directorio de D:\Dropbox\Dropbox\6_Teis\Programación\Ejercicios

11/10/2020  18:54    <DIR>        .
11/10/2020  18:54    <DIR>        ..
11/10/2020  18:54                427 HolaGrupo.class
11/10/2020  18:28                130 HolaGrupo.java
                2 archivos                557 bytes
                2 dirs  1.651.783.815.168 bytes libres
```

La operación de compilación se podrá realizar desde la carpeta donde está el fichero a compilar si las variables del sistema de Java están bien definidas.

Compilación del código fuente **HOLAGRUPO.JAVA** con errores

```
HolaGrupo.java: Bloc de notas
Archivo Edición Formato Ver Ayuda
public class HolaGrupo{
    public static void main ( String [] args) {
        Syste.out.println("Hola a todos/as");
    }
}
```

Al compilar se informa del error y no se genera el fichero *.class

```
D:\Dropbox\Dropbox\6 Teis\Programación\Ejercicios>javac HolaGrupo.java
HolaGrupo.java:3: error: package Syste does not exist
    Syste.out.println("Hola a todos/as");
      ^
1 error
```

Primer programa en Java

1. Ejecución

- Una vez compilado el fichero fuente de java, se utiliza el comando java seguido del nombre de la clase que contiene el método **main()**.

Ejecución del código objeto HOLAGRUPO.CLASS

```
diapositiva x | SECCION
C:\> Seleccionar Símbolo del sistema
D:\Dropbox\Dropbox\6_Teis\Programación\Ejercicios> java HolaGrupo
Hola a todos/as
```



Entorno de desarrollo integrado (IDE)

- Un **entorno de desarrollo integrado** o **IDE (Integrated Development Environment)** es una aplicación que facilita el desarrollo de aplicaciones en algún lenguaje de programación.
- Un IDE proporciona al programador/a una interfaz gráfica que facilita:
 - La escritura del código fuente en un lenguaje de programación.
 - La depuración del código fuente para la localización de errores sintácticos y de ejecución.
 - La organización de las clases en paquetes y proyectos.
 - La documentación.
 - La creación de paquetes Java *.jar,
 - Etc.
- Los IDE para Java utilizan internamente las herramientas básicas de JDK para realizar las operaciones descritas.



Entorno de desarrollo integrado (IDE)

- Existen en el mercado distintos IDE para Java
 - NetBeans.
 - Eclipse.
 - jEdit.
 - IntelliJ IDEA.
 - jBuilder.
 - Jdeveloper.
- En este curso desarrollaremos los programas Java en NetBeans v12.1.



Estructura general de un programa Java



- Todo programa debe estar escrito en una o varias **clases**.
- Dentro de cada clase se podrán utilizar distintas librerías Java incluidas en el JDK.
- Un programa Java consta de:
 - Una clase principal, que contiene el método **main**.
 - Algunas clases de usuario, específicas del programa que se está desarrollando, que son utilizadas por la clase principal.
- La clase principal:
 - Debe ser declarada con el modificación de acceso **public**.
 - La palabra reservada **class**, seguida del nombre de la clase con la primera letra en mayúscula.
- Un archivo fuente *.java puede contener más de una clase, pero sólo una puede ser **public**.
- El nombre del archivo fuente debe coincidir con el de la clase **public**.



El método main()

- En la clase principal debe existir una **función o método** estático llamado **main()** cuya cabecera debe ser

public static void main(String[] args)

```
public class HolaGrupo{  
    public static void main ( String [] args) {  
        System.out.println("Hola a todos/as");  
    }  
}
```

- El método **main()** debe tener las siguientes características:
 - Debe ser un método público (**public**).
 - Deber ser un método estático (**static**).
 - No puede devolver ningún resultado (**void**).
 - Debe declarar un array de cadenas de caracteres en la lista de parámetros o un número variable de argumentos (**String[] args**).
- El método **main** es el punto de arranque de un programa en Java.
- Cuando se ejecuta un fichero ***.class** con el comando **java** desde la línea de comandos, la **JVM** busca en la clase referida el método **main()**.
- Dentro del método **main()** puede:
 - crearse objetos de otras clases e invocar sus métodos,
 - Incluir cualquier instrucción java que desarrolle cualquier algoritmo.

```
public class HolaGrupo{  
    public static void main ( String [] args) {  
        System.out.println("Hola a todos/as");  
    }  
}
```



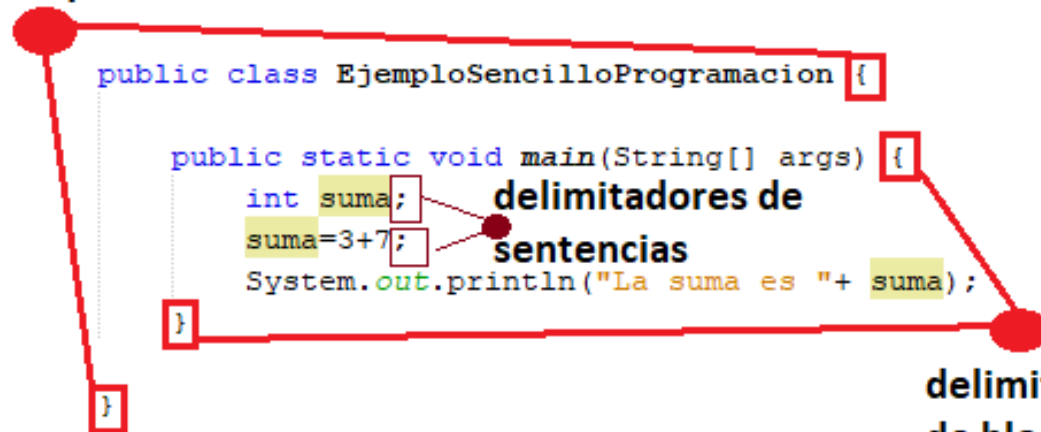

Sintaxis del lenguaje

- La sintaxis de un lenguaje de programación se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación.
- Solamente las instrucciones sintácticamente correctas pueden ser interpretadas por el compilador y los programas que contengan errores de sintaxis no pueden ser siquiera compilados, mucho menos ejecutados.
- Los elementos básicos de un programa son:
 - Palabras reservadas (propias de cada lenguaje)
 - Identificadores (nombres de variables, nombres de métodos, nombres de clases, etc.)
 - Caracteres especiales (operadores, delimitadores, comentarios, etc.)
 - Expresiones.
 - Instrucciones.

Sintaxis básica

- Java es **Case Sensitive**, es decir, diferencia entre mayúsculas y minúsculas.
- Las instrucciones finalizan siempre en ;
- Los bloques de instrucciones se delimitan con llaves { }

delimitadores
de bloque



delimitadores
de bloque

Comentarios

- Los comentarios son ignorados por el compilador.
- Los introduce el programador en su propio lenguaje a lo largo del código.
- Los comentarios se utilizan para facilitar la comprensión del código fuente, sirviendo de documentación.
- Existen tres tipos de comentarios.
 - Comentarios de una sola línea. *// Esta es una línea comentada.*
 - Comentarios de bloques. */* Aquí empieza el bloque comentado
y aquí acaba */*
 - Comentarios de documentación, que utiliza **javadoc**
*/** Los comentarios de documentación se realizan de este modo */*

comentario que será documentación de referencia del paquete

```
/**Este programa se utiliza con objetivo didáctico  
 * @author Esther Ferreiro  
 */
```

```
public class EjemploSencilloProgramacion {
```

```
    /* Método main que muestra la suma de dos  
    enteros*/
```

comentario de más de una línea

```
    public static void main(String[] args) {  
        int suma;  
        //Suma de enteros  
        suma=3+7;  
        System.out.println("La suma es "+ suma);  
    }
```

comentario de una línea

```
}
```

[Documentación relativa la herramienta de documentación javadoc](#)

Identificadores

Convenciones de
escritura de código
en Java

- Nombres de variables, métodos, clases, etc.
- Se pueden utilizar letras o dígitos y los caracteres \$ y _
- No hay límite en el número de caracteres.
- No pueden comenzar por un dígito.
- Se deben utilizar nombres que comiencen en mayúsculas en clases e interfaces. En cualquier otro caso, la primera palabra del identificador será en minúsculas.
- Cuando el identificador consta de varias palabras se debe utilizar la norma **CamelCase**, que en comenzar por mayúsculas todas las palabras, después de la primera. Por ejemplo, *ventanaBatiente*, *cocheCarreras*, etc.
- Los nombres de las constantes se definen siempre en mayúsculas. Por ejemplo, *PI*.

Identificadores

```
public class EjemploSencilloProgramacion {  
    public static void main(String[] args) {  
        int suma;  
        suma=3+7;  
        System.out.println("La suma es "+ suma);  
    }  
}
```

Palabras reservadas

- Las siguientes son palabras reservadas utilizadas por Java que no pueden ser usadas como identificadores.

abstract	continue	finally	int	public	throw
assert	default	float	interface	return	throws
boolean	do	for	long	short	transient
break	double	goto	native	static	true
byte	else	if	new	strictfp	try
case	enum	implements	null	super	void
catch	extends	import	package	switch	volatile
class	false	inner	private	synchronized	while
const	final	instanceof	protected	this	

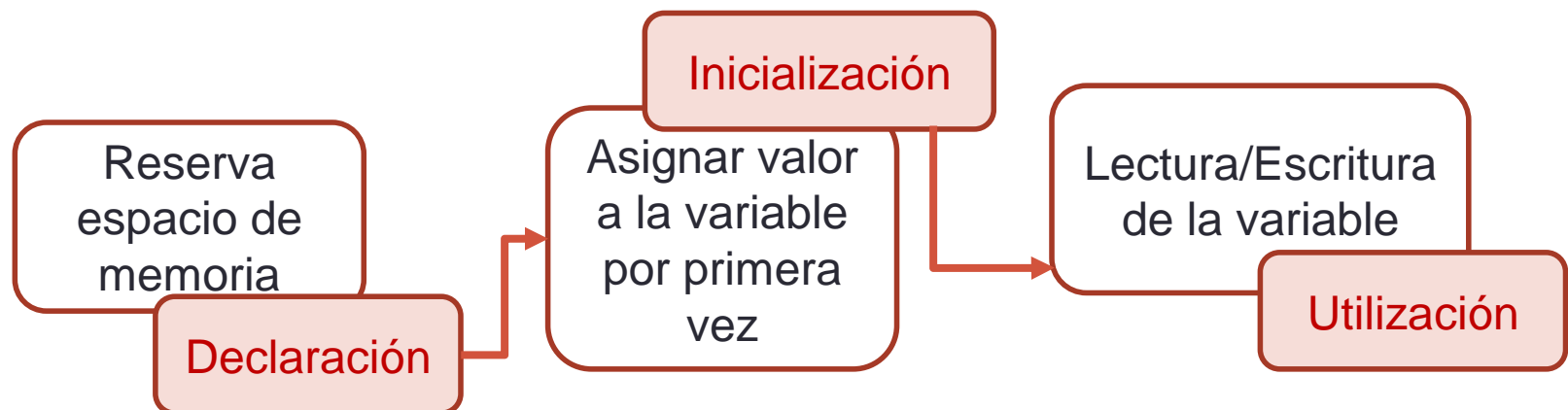
Palabras reservadas

```
public class EjemploSencilloProgramacion {  
    public static void main(String[] args) {  
        int suma;  
        suma=3+7;  
        System.out.println("La suma es "+ suma);  
    }  
}
```

Tipos de datos

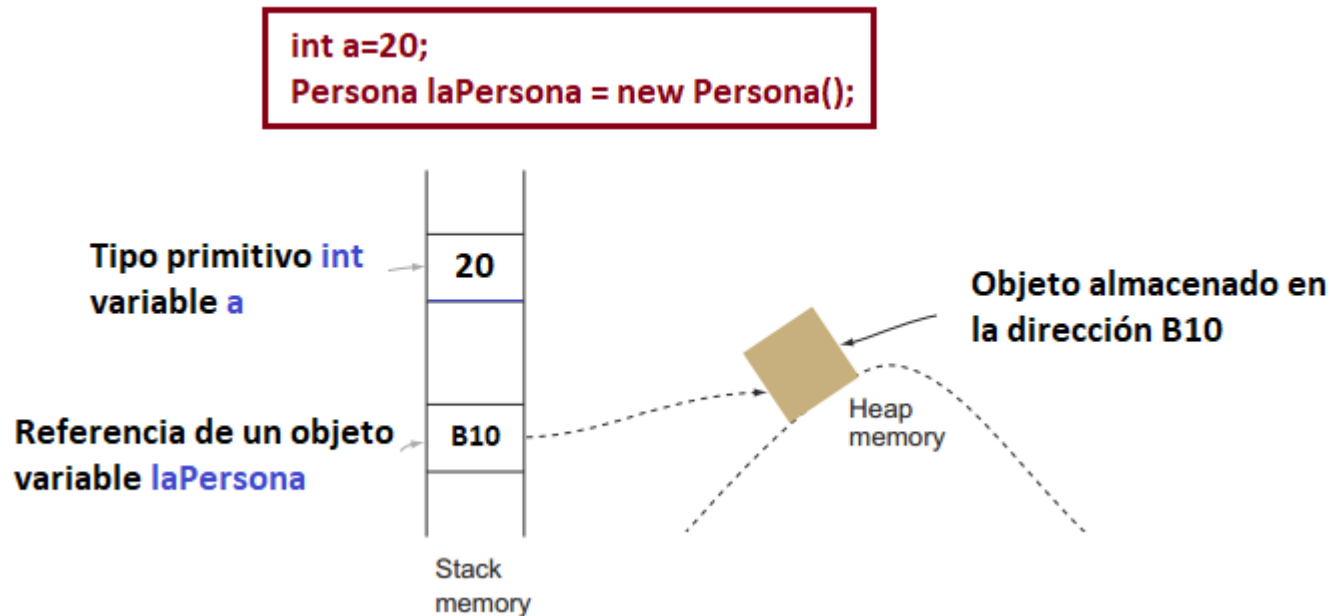
- Java es un lenguaje tipado estáticamente y fuertemente tipado.
- Un tipo de datos define un conjunto de valores y las operaciones sobre estos valores.
 - Por ejemplo, el tipo de datos **int** permite
 - Conjunto de valores: números enteros incluidos en el rango: -2^{63} a $2^{63}-1$.
 - Almacenar números enteros dentro del rango establecido y ser un operando en operaciones aritméticas, así como almacenar resultado de operaciones entre enteros.
- Toda variable debe ser inicializada antes de utilizar o leer su valor.
 - Declarar una variable reserva memoria pero no asigna valor.
- Después de declarada e inicializada la variable puede usarse en cualquier instrucción según permita su tipo de datos.
 - Un entero puede utilizarse en operaciones aritméticas, pero no para almacenar una cadena de caracteres.

tipo nombre
int count;



Tipo de datos

- Java tiene dos categorías de datos:
 - **Datos primitivos** definidos por el lenguaje: *int*, *char*, etc.
 - **Datos objeto**, que son tipos de datos creados en el programa a partir de una clase.



Tipos de datos primitivos

tipos enteros

Tipo de dato	Representación	Tamaño (Bytes)	Rango de Valores	Valor por defecto	Clase Asociada
byte	Numérico Entero con signo	1	-128 a 127	0	Byte
short	Numérico Entero con signo	2	-32768 a 32767	0	Short
int	Numérico Entero con signo	4	-2147483648 a 2147483647	0	Integer
long	Numérico Entero con signo	8	-9223372036854775808 a 9223372036854775807	0	Long
float	Numérico en Coma flotante de precisión simple Norma IEEE 754	4	$\pm 3.4 \times 10^{-38}$ a $\pm 3.4 \times 10^{38}$	0.0	Float
double	Numérico en Coma flotante de precisión doble Norma IEEE 754	8	$\pm 1.8 \times 10^{-308}$ a $\pm 1.8 \times 10^{308}$	0.0	Double
char	Carácter Unicode	2	\u0000 a \uFFFF	\u0000	Character
boolean	Dato lógico	-	true ó false	false	Boolean
void	-	-	-	-	Void

tipos reales

Valores literales

Tipo	Norma	Ejemplos
int	Valores en base decimal (la cifra tal cual), binario (la cifra precedida de 0b/0B), en octal (la cifra precedida de 0), o en hexadecimal (la cifra precedida de 0x/0X).	int d = 75; int b = 0b101001 int o = 024; int h = 0x36;
long	Se añade a la cifra la letra L (mayúsculas o minúsculas)	long d = 22L; long b = 0b100101L; long o = 033L; long h = 0x45aL
float	Valores en coma fija o coma flotante. Se añade la letra F (mayúsculas o minúsculas)	float x = 82.5f; float y = 0.5e15f; //1.2E9f=1,2*10^9
double	Valores en coma fija o coma flotante. Se añade la letra D (mayúsculas o minúsculas). <i>Este es el valor por defecto si no se pone ninguna letra.</i>	double y = 3.24e3d; double z = 2.75e-2;
boolean	Sólo hay dos posibles valores: <i>true</i> o <i>false</i> .	boolean acierto=true;
char	El tipo <i>char</i> puede contener un carácter Unicode. Caracteres visibles: Escribir el carácter entre comillas simples. Caracteres especiales, no visibles: Usar las comillas simples con secuencia de escape ("\", \"n'). También se puede usar la secuencia de escape seguida del código octal o hexadecimal que corresponda al carácter.	char letra = 'A'; // salto de línea: char salto = \"n'; // á (código en octal): char a1 = \"141'; // á (en hexadecimal): char a2 = \"u0061';
String	Cerrar la cadena entre comillas dobles	String nombre="Manuel"

A partir de la versión 1.7 de Java se puede utilizar el subrayado para realizar separaciones entre números para una mejor visualización.

A todos los efectos el valor del número es como si no existiese el carácter de subrayado.

long saldo = **123_345_342L**;

No se puede utilizar el literal de subrayado al principio o final del número, alrededor de un punto decimal, ni entre el número y un literal de entero o decimal (D, F o L).

Otros caracteres no visibles que se pueden usar con \

Secuencia	Significado
b	retroceso
t	tabular la cadena
n	salto de línea
f	form feed
r	retorno de carro
'	comilla simple
"	comilla doble
\	barra invertida

Ejemplo de literales de datos primitivos

```
//Asignación de un tipo carácter
char a = 'L';
```

```
// int es el tipo más utilizado
// para valores numéricos
int i=89;
```

```
//use byte y short para enteros
//más pequeños
byte b = 4;
```

```
//esto dará error ya que el n°
//es mayor que el rango de bytes
//byte b1 = 7888888955;
```

```
short s = 56;
```

```
// esto dará error ya que el n° es
// más grande que el rango de short
// short s1 = 87878787878;
```

```
// por defecto, el valor de la
//fracción es double en Java
double d = 4.355453532;
```

```
// para float use 'f' como sufijo
//Si no pongo el sufijo dará error
float f = 4.7333434f;
```

```
System.out.println("char: " + a);
System.out.println("int: " + i);
System.out.println("byte: " + b);
System.out.println("short: " + s);
System.out.println("float: " + f);
System.out.println("double: " + d);
```

```
char: G
integer: 89
byte: 4
short: 56
float: 4.7333436
double: 4.355453532
```



Ejercicio



- Realiza la declaración y asignación que te parezca más adecuada a las variables:
 - Censo de un país.
 - El nombre de un amigo/a.
 - El código postal.
 - Si tiene carnet de conducir o no.
 - El diámetro de una superficie de una instalación en una ciudad.
 - El peso de un contenedor de un puerto.
 - La edad de una persona.
 - La inicial de una letra.
 - Los plantas de un edificio en Galicia.
 - El número de alumno/as de un centro escolar.
 - El número de módulos de un ciclo de FP



Overflow de tipos primitivos

- Si se supera valor máximo posible de un tipo de datos se produce **overflow**.

```
byte a = 126;
```

```
//byte tiene un valor de 8 bits  
System.out.println(a);
```

```
a++;  
System.out.println(a);
```

```
//Se desborda aquí porque  
//el byte puede contener valores de -128 a 127  
a++;  
System.out.println(a);  
//rotación dentro del rango  
a++;  
System.out.println(a);
```

```
//Se produce una excepción  
a=300;
```

Cuando se supera el valor permitido en el tipo de datos de una variable **en tiempo de ejecución**, los valores rotan entre el rango de valores permitido. Esto sucede con todos los tipos de datos primitivos.

Si a una variable se le asigna un **valor literal** fuera de su rango permitido, se produce una excepción.

Precisión en float y double

- Los tipos **float** y **double** fueron diseñados para cálculos científicos, con un margen de error de precisión aceptables para esta función.
- Si la precisión es un requisito importante, es recomendable utilizar la clase **BigDecimal** que veremos más adelante.

```
float a;  
a=1/3;  
System.out.println("El resultado es "+a);  
a=1f/3f;  
System.out.println("El resultado es "+a);
```

```
El resultado es 0.0  
El resultado es 0.33333334
```

Si dividimos 1/3 el resultado es 0,333333333...(no acaba)

Este número no puede ser representado en un sistema computacional binario, por ello internamente se trunca el número y nos devuelve 0.0, que obviamente no es un resultado correcto.

Los números 1 y 3 son enteros pero se produce una conversión automática a float.

En el caso de utilizar 1f y 3f, la operación es más precisa, pero produce un redondeo automático.

Conversión automática de tipos primitivos numéricos

- A una variable de un tipo determinado le puedo asignar un literal de otro tipo siempre que:
 - Los dos tipos de datos sean compatibles.
 - El valor del literal sea más pequeño que el valor permitido en el tipo de la variable.

Conversión automática de tipos

byte -> short -> int -> long -> float -> double

```
int i = 100;

//conversion automatica de tipo
long l = i;

//conversion automatica de tipo
float f = l;

System.out.println("Valor Int "+i);
System.out.println("Valor Long "+l);
System.out.println("Valor Float "+f);
```

```
Valor Int 100
Valor Long 100
Valor Float 100.0
```


Conversión explícita de tipo (*casting*)

- Si queremos asignar un valor de tipo de dato más grande a un tipo de dato más pequeño, realizamos un **casteo/casting** o lo que se conoce como **conversión de tipo explícito**.
 - Esto es útil para tipos de datos incompatibles donde la conversión automática no se puede realizar.
 - **char** y **int** no son compatibles entre sí. Veamos luego cuando tratamos de convertir uno en otro.

Conversión explícita o casting

double -> float -> long -> int -> short -> byte

- Para hacer una conversión explícita debemos utilizar la siguiente sintaxis:

Tipo1 vable1=(Tipo1) variable

Resumen de las posibilidades de casting

Convertir desde	Convertir a...							
	boolean	byte	short	char	int	long	float	double
boolean		no	no	no	no	no	no	no
byte	no		si	cast	si	si	si	si
short	no	cast		cast	si	si	si	si
char	no	cast	cast		si	si	si	si
int	no	cast	cast	cast		si	si*	si*
long	no	cast	cast	cast	cast		si*	si*
float	no	cast	cast	cast	cast	cast		si
double	no	cast	cast	cast	cast	cast	cast	

Donde:

- no: indica que no hay posibilidad de conversión.
- si: indica que el casting es implícito.
- si*: indica que el casting es implícito pero se puede producir pérdida de precisión.
- cast: indica que hay que hacer casting explícito.

Conversión explícita. Ejemplo1

```
double d = 100.04;
```

```
//casting de tipo  
long l = (long)d;
```

```
//casting de tipo  
int i = (int)l;  
System.out.println("Valor Double "+d);
```

```
//parte fraccionaria perdida  
System.out.println("Valor Long "+l);
```

```
//parte fraccionaria perdida  
System.out.println("Valor Int "+i);
```

```
Valor Double 100.04  
Valor Long 100  
Valor Int 100
```

Conversión explícita. Ejemplo2

```
byte b;  
int i = 257;  
double d = 323.142;  
System.out.println("Conversion de int a byte.");
```

```
b = (byte) i;  
System.out.println("i = " + i + " b = " + b);  
System.out.println("\nConversion de double a byte.");
```

```
b = (byte) d;  
  
System.out.println("d = " + d + " b= " + b);
```

```
Valor Double 100.04  
Valor Long 100  
Valor Int 100
```

Variables y constantes

- En un programa puede ser necesario definir:
 - **Variables:** representan un espacio de memoria que puede variar su contenido a lo largo del programa.
 - **Constantes:** representa un espacio de memoria que no puede cambiar su contenido durante la ejecución del programa.
- En los dos casos se debe especificar su tipo, pero al declarar una constante se tiene que añadir la palabra reservada **final** y el identificador que la representa se suele escribir en mayúsculas.

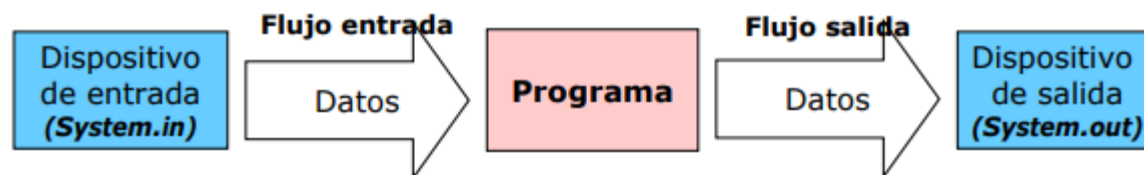
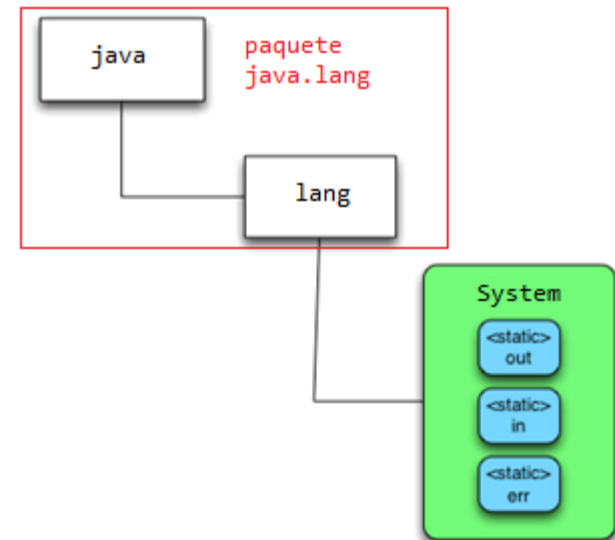
```
int contador = 0;    // variable entera con valor inicial
int i, j, k;         // 3 variables enteras, sin inicializar
```

```
final int N = 100;    // constante entera, de valor 100
final char ULTIMA = 'Z'; // constante carácter, de valor Z
final double PI = 3.141592; // constante real, de valor 3,1416
```

Flujo de datos

- Todos los datos fluyen a través del ordenador desde una entrada hacia una salida.
- Este flujo de datos se denomina también **stream**.
- Hay un **flujo de entrada** (*input stream*) que manda los datos desde el exterior (normalmente el teclado) del ordenador, y un **flujo de salida** (*output stream*) que dirige los datos hacia los dispositivos de salida (la pantalla o un archivo).
- En Java se accede a la e/s estándar a través de campos estáticos de la clase **System** del paquete **java.lang**
 - **System.in** implementa la entrada estándar.
 - **System.out** implementa la salida estándar.
 - **System.err** implementa la salida de error.

El paquete `java.lang` es importado por defecto en los ficheros de código de Java



Entrada y salida de datos por consola

- El **API de Java** (*JLC Java Class Library*) incluye clases que permiten **leer información desde el teclado** y **mostrarla en la consola**.
- Para el envío de datos al exterior se utiliza un flujo de datos de impresión **System.out** (objeto de la clase **PrintStream**) con la siguiente sintaxis:

```
System.out.println("Este es mi mensaje");
```

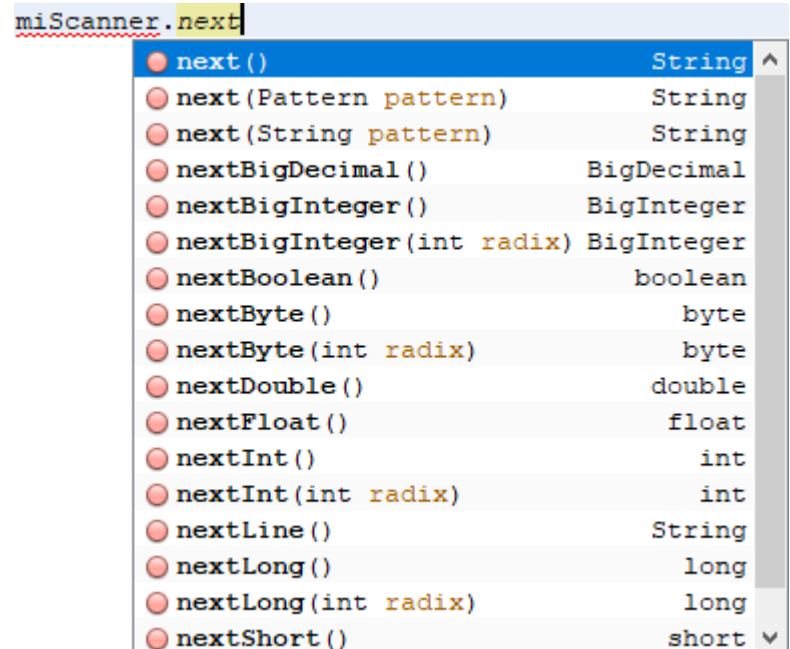
- Para la recepción de datos desde el exterior se utiliza un *flujo de datos de entrada* que será **System.in** (objeto de la clase **InputStream**). La sintaxis es la siguiente:

```
Scanner miScanner = new Scanner(System.in);
```

La clase **Scanner** debe ser importada. Para importar clases **Ctrl+↑+I**

Clase Scanner

- La clase **Scanner**:
 - Es una clase del paquete **java.util**
 - Se utiliza para **leer datos** de **tipos primitivos y String** por teclado.
- Para crear un objeto Scanner que lea información del teclado:
 - Pasamos por parámetro en el constructor de **Scanner** un objeto predefinido **System.in**, que representa el *flujo de datos estándar*.
 - A partir del objeto podemos hacer lecturas por tipo de datos.



```
Scanner miScanner = new Scanner(System.in);  
String miCadena = miScanner.nextLine();  
int miEntero = miScanner.nextInt();  
float miFloat = miScanner.nextFloat();
```


Lectura de caracteres y String

- Los métodos de la clase Scanner **next()** y **nextLine()** permiten la lectura de datos de tipo **String**.
 - **next()**:
 - Lee hasta el primer espacio en blanco, es decir, solo lee una palabra.
 - Coloca el cursor en la *misma línea* después de leer la entrada.
 - **nextLine()**: **Este es el método recomendado para leer un String.**
 - Lee hasta que se introduce un salto de línea (`\n` → tecla *intro*).
 - Coloca el cursor en una *nueva línea* después de leer la entrada.
- Para leer un carácter se debe utilizar el método **next().charAt(0)**.
 - La función **next()** devuelve la palabra de la entrada como una cadena y la función **charAt(0)** extrae el primer carácter.

Ejemplo

//Programa que pida un número entero y nos diga si es par o no

```
public static void main(String[] args) {  
    int numero;  
    Scanner input = new Scanner(System.in);  
  
    System.out.println("introduzca un numero");  
    numero = input.nextInt();  
  
    if (numero % 2 == 0) {  
        System.out.println("el numero " + numero + " el numero es par.");  
    } else {  
        System.out.println("el numero " + numero + " el numero es impar.");  
    }  
}
```



Errores de compilación y de ejecución

- **Errores en tiempo de compilación:**

- Este tipo de errores **no permiten** que la aplicación se ejecute, por ejemplo:
 - No poner un ; al final de una sentencia.
 - No tener bien cerrados los bloques de un método, una clase o una estructura de control.
 - Repetir variables con el mismo nombre o hacer referencia a variables no definidas, etc.
- Estos errores son asistidos por los **IDEs** para hacer más fácil su detección.

- **Errores en tiempo de ejecución:**

- Se producen mientras el programa se está ejecutando.
- Lanzas **excepciones** que se pueden controlar por el programador.

```
public class ErroresCompilacion {

    public static void main(String[] args) {
        int suma=0;
        String nombre;
        suma=nombre; //error 1

        int valor=0;
        short numero=0;
        numero=valor; //error 2

        String valor=""; //error 3

        int mayor;
        int menor;
        if (true){
            mayor>menor; //error 4
        }
    }
}
```

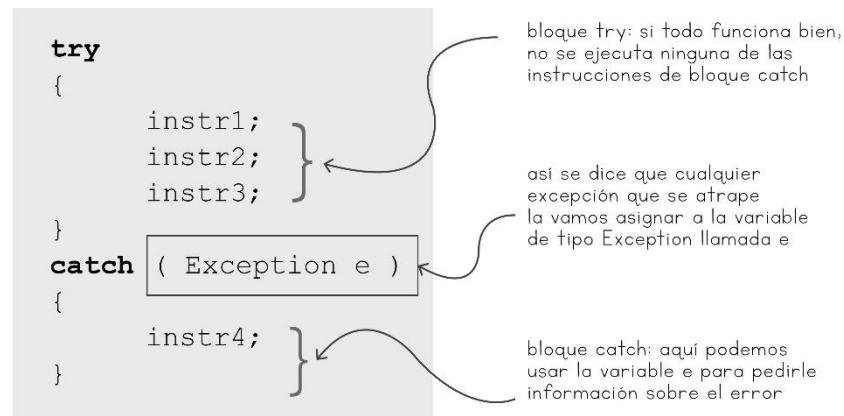
Errores en tiempo
de compilación

```
public class ErroresEjecucion {  
  
    public static void main(String[] args) {  
        int sumaNotasExámenes=60;  
        int numeroExámenes=0;  
        int promedio=sumaNotasExámenes/numeroExámenes;  
  
    }  
}
```

Errores en tiempo de
ejecución (intentar dividir
por 0)

Excepciones

- Una excepción es un evento que ocurre durante la ejecución de un programa al producirse un error en tiempo de ejecución
- Cuando se trata una excepción el programa no se interrumpe sino que ejecuta las instrucciones que el programador ha indicado para esos casos.
- Las excepciones deben ser capturadas para ser tratadas.
- Todas las excepciones derivan de la clase **Exception**.
 - Cada clase de excepción es descriptiva y representa un error concreto de forma clara y evidente.
 - Cada clase de excepción contiene también información específica.
 - Por ejemplo, una clase **FileNotFoundException** podría contener el nombre del archivo no encontrado
- **Para capturar una excepción** se utiliza el bloque **try/catch**.
 - Si se produce una excepción en el código que está dentro de **try** el flujo de ejecución salta al bloque **catch**.
 - En cuanto se produce una excepción, la ejecución del bloque **try** finaliza.
 - La clausula **catch** recibe como argumento un objeto **Throwable**



```
// Bloque 1
try {
    // Bloque 2
} catch (Exception error) {
    // Bloque 3
}
// Bloque 4
```

Captura de una
excepción genérica

Sin excepciones: $1 \rightarrow 2 \rightarrow 4$

Con una excepción en el bloque 2: $1 \rightarrow 2^* \rightarrow 3 \rightarrow 4$

Con una excepción en el bloque 1: 1^*

```
// Bloque 1
try {
    // Bloque 2
} catch (ArithmeticException ae) {
    // Bloque 3
} catch (NullPointerException ne) {
    // Bloque 4
}
// Bloque 5
```

Captura de
excepciones
concretas

Sin excepciones: $1 \rightarrow 2 \rightarrow 5$

Excepción de tipo aritmético: $1 \rightarrow 2^* \rightarrow 3 \rightarrow 5$

Acceso a un objeto nulo (`null`): $1 \rightarrow 2^* \rightarrow 4 \rightarrow 5$

Excepción de otro tipo diferente: $1 \rightarrow 2^*$

¡Ojo! Las cláusulas check se comprueban en orden

```
// Bloque 1
try {
    // Bloque 2
} catch (Exception error) {
    // Bloque 3
} catch (ArithmeticException ae) {
    // Bloque 4
}
// Bloque 5
```

La excepción
ArithmeticException
nunca se va a
capturar

Sin excepciones:	$1 \rightarrow 2 \rightarrow 5$
Excepción de tipo aritmético:	$1 \rightarrow 2^* \rightarrow 3 \rightarrow 5$
Excepción de otro tipo diferente:	$1 \rightarrow 2^* \rightarrow 3 \rightarrow 5$

Poner un bloque catch al final con el parámetro de tipo **Exception** garantiza la captura de todo tipo de excepciones en el Bloque 2

```
// Bloque 1
try {
    // Bloque 2
} catch (ArithmeticException ae) {
    // Bloque 3
} catch (NullPointerException ne) {
    // Bloque 4
} catch (Exception error) {
    // Bloque 5
}
//Bloque 6
```


Excepciones. Clausula finally

- En ocasiones, interesa ejecutar un fragmento de código independientemente de si se produce o no una excepción, por ejemplo, cerrar un fichero que estamos manipulando.

```
// Bloque1
try {
    // Bloque 2
} catch (ArithmeticException ae) {
    // Bloque 3
} finally {
    // Bloque 4
}
// Bloque 5
```

El bloque 4 siempre se ejecutará, se produzca una excepción o no

Sin excepciones:

$1 \rightarrow 2 \rightarrow 4 \rightarrow 5$

Excepción de tipo aritmético:

$1 \rightarrow 2^* \rightarrow 3 \rightarrow 4 \rightarrow 5$

Excepción de otro tipo diferente:

$1 \rightarrow 2^* \rightarrow 4$

Excepciones. Resumen

```
try {  
    // Código del programa  
    // Puede producir excepciones  
} catch(TipoDeExcepcion1 e1) {  
    // Código que trata las excepciones de tipo  
    // TipoDeExcepcion1 o subclases de ella.  
    // Los datos sobre la excepción los encontraremos  
    // en el objeto e1.  
} catch(TipoDeExcepcion2 e2) {  
    // Código que trata las excepciones de tipo  
    // TipoDeExcepcion2 o subclases de ella.  
    // Los datos sobre la excepción los encontraremos  
    // en el objeto e2.  
  
...  
} catch(TipoDeExcepcionN eN) {  
    // Código que trata las excepciones de tipo  
    // TipoDeExcepcionN o subclases de ella.  
    // Los datos sobre la excepción los encontraremos  
    // en el objeto eN.  
} finally {  
    // Código de finalización (opcional)  
}
```

Excepciones. Ejemplo

//Programa que pida un número entero y nos diga si es par o no
//Con control básico de excepciones

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

```
    int numero;
```

```
    Scanner input = new Scanner(System.in);
```

```
    System.out.println("introduzca un numero");
```

```
    try {
```

```
        numero = input.nextInt();
```

```
        if (numero % 2 == 0) {
```

```
            System.out.println("el numero " + numero + " el numero es par.");
```

```
        } else {
```

```
            System.out.println("el numero " + numero + "e l numero es impar.");
```

```
        }
```

```
    } catch (Exception e) {
```

```
        System.out.println("Ha habido un error en la introducción de datos");
```

```
    }
```

```
}
```

Al ejecutar, introducir un carácter para provocar una excepción.

Excepciones. Ejemplo

```
try {  
    int[] myNumbers = {1, 2, 3};  
    System.out.println(myNumbers[10]);  
} catch (Exception e) {  
    System.out.println("Algo ha ido mal.");  
} finally {  
    System.out.println("Este bloque se ejecuta en cualquier caso.");  
}
```

El bloque **finally** se ejecuta se produzca la excepción o no

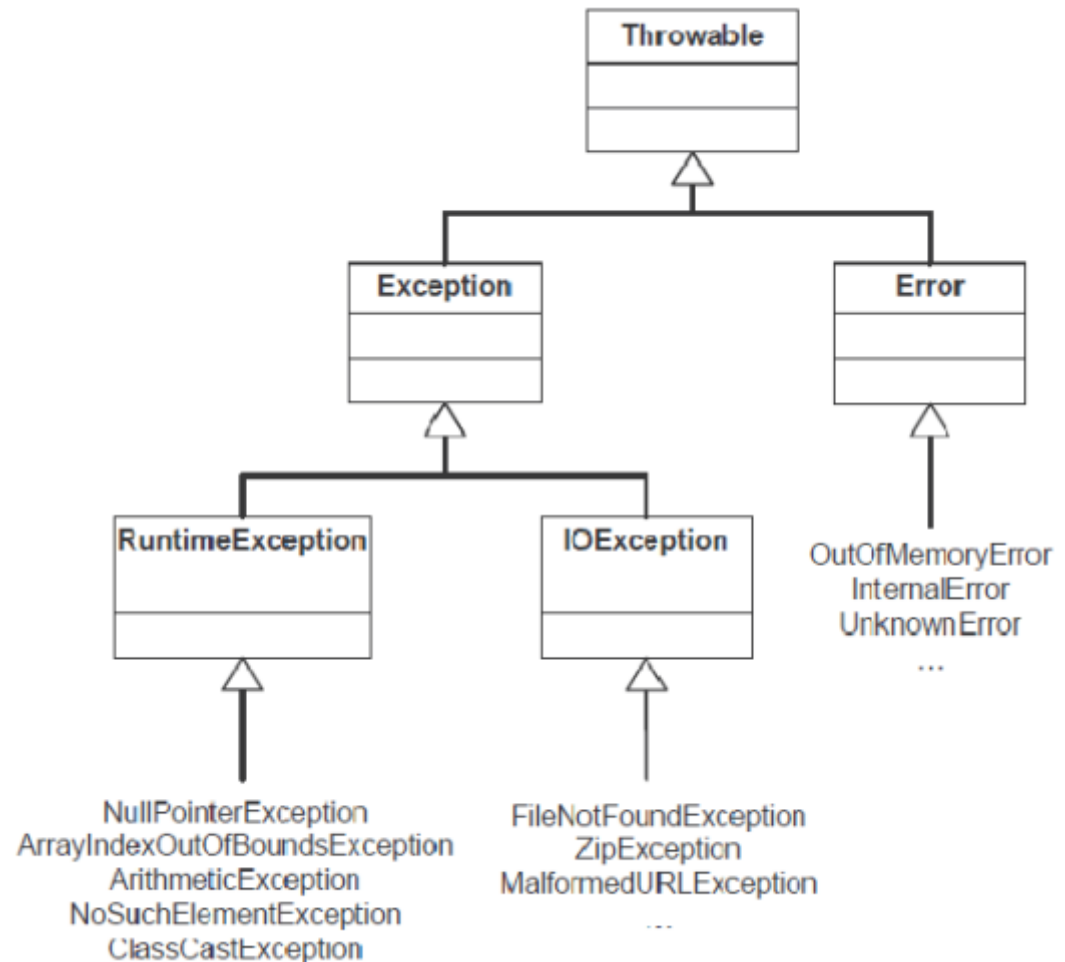
Excepciones. Ejemplo

```
public static void main(String[] args) throws IOException {  
    int[] array = new int[20];  
    try {  
        array[20] = 24;  
        int b = 0;  
        int a = 23 / b;  
    } catch (ArrayIndexOutOfBoundsException excepcion) {  
        System.out.println(" Error de índice en un array");  
    } catch (ArithmeticException excepcion) {  
        System.out.println(" Error aritmético");  
    }  
}
```

Se puede comentar/descomentar la línea
array[20]=24
para comprobar la captura de las distintas excepciones

Excepciones. Jerarquía de clases

- Todas las excepciones derivan de la superclase **Throwable**.



Operadores aritméticos

Operación	Operador normal	Operador reducido	Operador automático
Suma	+	+=	++
Resta	-	--	--
Multiplicación	*	*=	
División (cociente)	/	/=	
Módulos (resto)	%	%=	

```
int i=2, j, k;  
j = i * 3;    // resultado: j = 2*3 = 6  
i++;         // resultado: i = 2+1 = 3  
j *= i;      // resultado: j = 6*3 = 18  
k = j % i;   // resultado: k = 18%3 = 0  
i++;         // resultado: i = 3+1 = 4  
j /= i;      // resultado: j = 18/4 = 4  
double x=3.5, y, z;  
y = x + 4.2; // resultado: y = 3.5+4.2 = 7.7  
x--;        // resultado: x = 3.5-1.0 = 2.5
```

Los operadores aritméticos permiten realizar distintas operaciones con números enteros y reales.

No es conveniente que el resultado de las operaciones aritméticas se almacenen en una variable de tipo short o byte.

Probar a cambiar en el ejemplo y analizar posibles conflictos.

Operadores pre-incremento y post-incremento

- Los operadores aritméticos automáticos se utilizan para incrementar y disminuir el valor de una variable en 1 unidad.
- Los operadores **++** y **--** se pueden utilizar antes o después de la variable y, en función de su posición tendrá un funcionamiento diferente en el caso de que se produzca una asignación.

x = z++ //se asigna el valor de **z** a **x** y después se incrementa **z** en 1 unidad

x = ++z //se incrementa **z** en 1 unidad y después se asigna el valor resultante a **x**

x = z-- //se asigna el valor de **z** a **x** y después se decrementa **z** en 1 unidad

x = --z //se decrementa **z** en 1 unidad y después se asigna el valor resultante a **x**

Ejemplos

```
static void postIncremento()
```

```
    int a = 5;
```

```
    System.out.println("El valor inicial de a es: " + a);
```

```
    //operador post incremental
```

```
    //el valor actual de "a" es usado y
```

```
    // mostrado por pantalla, luego
```

```
    //el valor incrementa en una unidad
```

```
    System.out.println("El valor de a es: " + a++);
```

```
    //mostramos nuevamente el valor de "a", donde
```

```
    //podemos observar el incremento realizado
```

```
    System.out.println("El valor de a es: " + a);
```

```
}
```

```
El valor inicial de a es: 5
El valor de a es: 5
El valor de a es: 6
```

```
static void preIncremento() {  
    int a = 5;  
    System.out.println("El valor inicial de a es: " + a);  
  
    //operador pre incremental  
    //el valor actual de "a" es incrementado en una unidad  
    luego  
    //es usado para ser mostrado por pantalla  
    System.out.println("El valor de a es: " + ++a);  
  
    //mostramos nuevamente el valor de "a", que no ha  
    variado  
    System.out.println("El valor de a es: " + a);  
}
```

```
El valor inicial de a es: 5  
El valor de a es: 6  
El valor de a es: 6
```

Operadores aritméticos

• Caso 1: Operador %

- El operador % devuelve el resto si los operandos son enteros.
- En el caso de que los operandos sean reales, el resultado de corresponde con la siguiente operación:
 - Dividendo – (divisor*parte entera del cociente)
 - Ejemplo: $5.7 \% 3.0$
 - $5.7 - (3.0 * 1) = 2.7$ //El cociente de la operación es 1,9

• Caso 2: Dividir por 0

- Entre enteros provoca una excepción.
- Entre reales, devuelve **Infinity**

```
int x = 5;  
int y = 0;  
int z = x/y;  
System.out.println(z);
```

Infinity

```
float x = 5.0f;  
float y = 0.0f;  
float z = x/y;  
System.out.println(z);
```

Exception in thread "main" java.lang.ArithmeticException: / by zero

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

```
{
```

```
    int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;
```

```
    // operador de asignación simple
```

```
    c = b;
```

```
    System.out.println("Valor de c = " + c);
```

```
    // operadores de asignación simples
```

```
    a = a + 1;
```

```
    b = b - 1;
```

```
    e = e * 2;
```

```
    f = f / 2;
```

```
    System.out.println("a,b,e,f = " + a + "," + b + "," + e + "," + f);
```

```
    a = a - 1;
```

```
    b = b + 1;
```

```
    e = e / 2;
```

```
    f = f * 2;
```

```
    System.out.println("a,b,e,f = " + a + "," + b + "," + e + "," + f);
```

```
    // operados de asignación reducidos
```

```
    a += 1;
```

```
    b -= 1;
```

```
    e *= 2;
```

```
    f /= 2;
```

```
    System.out.println("a,b,e,f (usando operadores reducidos)= " + a + "," + b + "," + e + "," + f);
```

```
}
```

Si en lugar de asignar c=b
hiciéramos c=d....¿Qué
ocurriría? ¿Por qué?

Operadores aritméticos relacionales

- Los operadores relacionales se utilizan para comparar dos valores de variables o literales del mismo tipo, el resultado es siempre *booleano*.

Operador	Operación
<code>==</code>	Devuelve true si los valores que compara son iguales
<code>!=</code>	Devuelve true si los valores que compara son diferentes
<code><</code>	Devuelve true si el valor de la izquierda es menor que el de la derecha
<code>></code>	Devuelve true si el valor de la izquierda es mayor que el de la derecha
<code><=</code>	Devuelve true si el valor de la izquierda es menor o igual que el de la derecha
<code>>=</code>	Devuelve true si el valor de la izquierda es mayor o igual que el de la derecha



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

```
    int a = 20, b = 10;
```

```
    boolean condicion = true;
```

```
    //varios operadores condicionales
```

```
    System.out.println("a == b :" + (a == b));
```

```
    System.out.println("a < b :" + (a < b));
```

```
    System.out.println("a <= b :" + (a <= b));
```

```
    System.out.println("a > b :" + (a > b));
```

```
    System.out.println("a >= b :" + (a >= b));
```

```
    System.out.println("a != b :" + (a != b));
```

```
    System.out.println("condicion==true :" + (condicion == true));
```

```
}
```

Cambia los valores para
comprobar los operadores

Operadores lógicos

Operador	Operación
<code>&&</code>	AND lógico. Devuelve true si las condiciones son verdaderas
<code> </code>	OR lógico. Devuelve true si alguna de las condiciones es verdaderas o si lo son las dos
<code>!</code>	NOT lógico. Devuelve el valor boolean inverso del valor que acompaña

Estos operadores lógicos se denominan **operadores en cortocircuito** porque solo evalúan la expresión hasta que pueden obtener un resultado, el cual, en muchos casos, no necesita la evaluación de toda la expresión.

Por ejemplo:

`(a == b && c != d && e >= f)`

Esta expresión realiza tres evaluaciones. En primer lugar evalúa si `a = b`, en el caso de que el resultado sea *false* ya no se sigue evaluando la expresión pues sería necesario que todas las condiciones fuesen *true* para que el resultado de la expresión también lo fuese.

Ejemplos

Operador	Significado	True	False
&&	Operador AND conjunción	$(7 < 9) \&\& (7 < 15)$	$(-100 > 0) \&\& (-100 < 500)$
	Operador OR disyunción	$(5 > 2) (5 < 3)$	$(5 < 2) (5 < 0)$
!	Operador NOT Negación	$!(5 < 3)$	$!(7 < 11)$

Operador ternario

- El operador ternario (?) es una versión abreviada de la declaración **if-else**.
- Tiene tres operandos y de ahí el nombre ternario. El formato general es:

valor = expresion_boolean? expre_si_true: expre_si_false

- La declaración anterior significa que si la condición se evalúa como verdadera, entonces se asigna a **valor** la **expr_si_true** y si es falso se le asigna **expr_si_false**.
- La expresión ternaria anterior es equivalente a:

```
if (expresion_boolean)  
    valor = expre_si_true;  
else  
    valor = expre_si_false
```

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

```
    int a = 18, b = 14, c = 22, maximo;
```

```
    maximo = a > b ? a : b;  
    System.out.println (maximo);
```

```
    maximo = maximo > c ? maximo : c;  
    System.out.println (maximo);
```

```
}
```

18

22

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

```
    int a = 18, b = 14, c = 22;  
    String s = "";
```

```
    s = (a > b && a > c) ? "máximo = " + a : s + "";
```

```
    s = (b > a && b > c) ? "máximo = " + b : s + "";
```

```
    s = (c > a && c > b) ? "máximo = " + c : s + "";
```

```
    System.out.println(s);
```

```
}
```

máximo = 22

Expresiones: uso de paréntesis

- Usamos paréntesis cuando:
 - Exista ambigüedad sobre qué operador aplicar antes que otro.
 - Se quiera dar más precedencia a unos operadores que a otros.
 - Se quiera hacer el código más legible/ entendible.

```
int a1 = 5 * 3 + 2;  
int a2 = (5 * 3) + 2;  
int a3 = 5 * (3 + 2);  
System.out.println("a1: "+a1+" a2: "+a2+" a3: "+a3);
```

```
int b1 = 36 / 2 * 5;  
int b2 = 36 / (2 * 5);  
System.out.println("b1: "+b1+" b2: "+b2);
```

```
int c1 = 12 / 2 + 4;  
int c2 = 12 / (2 + 4);  
System.out.println("c1: "+c1+" c2: "+c2);
```

```
int d1 = 2 + (2 * (5 + 5));  
int d2 = (2 + 2) * (5 + 5);  
int d3 = 2 + (2 * 5) + 5;  
System.out.println("d1: "+d1+" d2: "+d2+" d3: "+d3);
```

```
a1: 17 a2: 17 a3: 25  
b1: 90 b2: 3  
c1: 10 c2: 2  
d1: 22 d2: 40 d3: 17
```

Los paréntesis determinan el orden en el que se ejecutan los operadores

Expresiones: preferencia de operadores

<div>Mayor</div> <div>Menor</div>	<div></div>	()
	<div></div>	! ~ ++ --
	<div></div>	(cast)
	<div></div>	* / %
	<div></div>	+ -
	<div></div>	>> << >>>
	<div></div>	> >= <= > instanceof
	<div></div>	== !=
	<div></div>	&
	<div></div>	^
	<div></div>	
	<div></div>	&&
	<div></div>	
	<div></div>	Ternario
	<div></div>	asignación

Expresiones: preferencia de operadores

- Dada la siguiente expresión:

$$A+B == 8 \ \&\& \ A-B == 1$$

siendo $A = 3$ y $B = 5$

- El orden de evaluación será el siguiente:
 1. Se calcula primero $A+B$ que vale 8,
 2. Luego se calcula $A-B$ que vale -2.
 3. A continuación, se evalúa si se cumple que $8==8$, que tiene como resultado *true*.
 4. Después evalúa si $-2==1$, que tiene como resultado *false*.
 5. En último lugar, se evalúa el operador $true \ \&\& \ false$, resultando *false*.

Expresiones: preferencia de operadores

- **Ejercicio:** Dadas las variables de tipo int con valores $A = 5$, $B = 3$, $C = -12$ indicar si la evaluación de estas expresiones daría como resultado verdadero o falso:

- a) $A > B$
- b) $B \neq C$
- c) $A == 3$
- d) $A * B == 5$
- e) $C / B \neq -10$
- f) $A * B == -30$
- g) $C / B < A$
- h) $A + B + C == 5$
- i) $(A + B == 8) \&\& (A - B == 2)$
- j) $(A + B == 8) \|\ (A - B == 6)$
- k) $A > 3 \&\& B > 3 \&\& C < 3$
- l) $A > 3 \&\& B \geq 3 \&\& C < -3$

Cadenas de texto: String / StringBuilder

- Una cadena de texto es un conjunto de caracteres alfanuméricos que se tratan como una unidad.
- Existen dos tipos de cadenas con las que trabaja Java:
 - **String**: cadena de textos constante, no se puede modificar.
 - Su literal se escribe siempre con comillas dobles. `String saludo = "Hola";`
 - Podemos leer su valor, extraer parte de él, etc. Pero para cualquier modificación es preciso volcar los datos en una nueva cadena.
 - No es un tipo primitivo, a pesar de que se declara sin `new String()`.
 - **StringBuilder**: cadena de textos que permite la modificación de su contenido directamente.
 - StringBuilder es una clase, por lo tanto es necesario crear un objeto con `new` y el constructor de la clase.
`StringBuilder texto3 = new StringBuilder("Otra prueba");`
 - Podemos insertar letras, ponerlas en orden inverso, etc.

String: declaración

- El tipo **String** no es un tipo primitivo de Java.
- Cuando declaramos una variable **String** se crea un objeto, a pesar de que la sintaxis de esta declaración se hace como si fuese un tipo primitivo

String cadena1="Hola "; // Forma simple de crear un String

String cadena2= new String ("Mundo"); //Crear un objeto String

La forma habitual de crear un String es utilizar la forma simple.

String: uso del operador +

- El operador **+** permite la concatenación de literales y variables de tipo String.

```
String nombre, apellidos;
```

```
Scanner miScanner=new Scanner (System.in);
```

```
System.out.println("Introduce tu nombre:");  
nombre=miScanner.nextLine();
```

```
System.out.println("Introduce tus apellidos");  
apellidos=miScanner.nextLine();
```

```
System.out.println("Tu nombre completo es "+  
    nombre+  
    " "+  
    apellidos);
```

Los variables y los literales **String** permiten el uso del operador **+** cuyo resultado es la concatenación de los mismos.

Las operaciones de concatenación se pueden escribir en **líneas diferentes**.

String: comparación.

```
public static void main(String[] args) {  
    String x = "Pepe";  
    String y = "_23";
```

```
    Scanner s = new Scanner(System.in);  
    System.out.println("Usuario:");  
    String uuid = s.nextLine();  
    System.out.println("Contraseña:");  
    String upwd = s.nextLine();
```

```
    // Comprobar si el nombre de usuario y la contraseña coinciden o no  
    if ((uuid.equals(x) && upwd.equals(y)) ||  
        (uuid.equals(y) && upwd.equals(x))) {  
        System.out.println("Bienvenido al sistema.");  
    } else {  
        System.out.println("Identificación errónea");  
    }  
}
```

Las variables y literales de tipo **String** *no se comparan con el operador ==* sino que utilizan el método, **equals**.

String: métodos

- Para consultar todos los métodos de la clase **String** consultar el siguiente [enlace](#).
- Mediante el siguiente código destacaremos algunos de los más relevantes:

```
String cadena = "No por mucho madrugar amanece más temprano";
```

```
System.out.println(cadena.charAt(0));
```

```
System.out.println(cadena.charAt(9));
```

```
System.out.println(cadena.endsWith("o"));
```

```
System.out.println(cadena.startsWith("n"));
```

```
System.out.println(cadena.equals("No por mucho madrugar amanece más temprano"));
```

```
System.out.println(cadena.indexOf("mucho"));
```

```
System.out.println(cadena.length());
```

```
System.out.println(cadena.replace('o', 'i'));
```

```
System.out.println(cadena.toLowerCase());
```

```
System.out.println(cadena.toUpperCase());
```

```
N
c
true
false
true
7
42
Ni pir muchi madrugar amanece más temprani
no por mucho madrugar amanece más temprano
NO POR MUCHO MADRUGAR AMANECE MÁS TEMPRANO
```

String: modificaciones

- Los **String** **no pueden ser modificados**, en el caso de querer cambiar su contenido, debemos de realizar una nueva asignación:

```
String elString = "más ";
```

```
elString = elString + "vale ";
```

```
elString = elString + "pájaro ";
```

```
elString = elString + "en mano...";
```

```
System.out.println(elString);
```

```
más vale pájaro en mano...
```

StringBuilder

- La clase **StringBuilder** nos permite crear cadenas dinámicas cuyo **contenido y longitud pueden ser modificadas directamente**.
- **Constructores de StringBuilder:**

```
StringBuilder bufer1 = new StringBuilder();
```

Crea un objeto que no tiene valor y que permite 16 caracteres iniciales, es la capacidad por defecto

```
StringBuilder bufer2 = new StringBuilder(10);
```

Crea un objeto que permite 10 caracteres iniciales y no tiene valor

```
StringBuilder bufer3 = new StringBuilder("hola");
```

```
System.out.println("bufer1 = " + bufer1.toString());
```

```
System.out.println("bufer2 = " + bufer2.toString());
```

```
System.out.println("bufer3 = " + bufer3.toString());
```

Crea un objeto que contiene el valor "hola" y que permite 16 caracteres.

StringBuilder: métodos

Retorno	Método	Explicación
StringBuilder	append(...)	Añade al final del StringBuilder a la que se aplica, un String o la representación en forma de String de un dato asociado a una variable primitiva
int	length()	Devuelve el número de caracteres del StringBuilder
StringBuilder	reverse()	Invierte el orden de los caracteres del StringBuilder
void	setCharAt(int indice,char ch)	Cambia el carácter indicado en el primer argumento por el carácter que se le pasa en el segundo
char	charAt(int indice)	Devuelve el carácter asociado a la posición que se le indica en el argumento
String	toString()	Convierte un StringBuilder en un String
StringBuilder	insert(int indiceIni,String cadena)	Añade la cadena del segundo argumento a partir de la posición indicada en el primero
StringBuilder	delete(int indiceIni,int indiceFin)	Borra la cadena de caracteres incluidos entre los dos índices indicados en los argumentos
StringBuilder	deleteChar(int indice)	Borra el carácter indicado en el índice
StringBuilder	replace(int indiceIni, int indiceFin,String str)	Reemplaza los caracteres comprendidos entre los dos índices por la cadena que se le pasa en el argumento
int	indexOf (String str)	Analiza los caracteres de la cadena y encuentra el primer índice que coincide con el valor deseado
String	substring(int indiceIni,int indiceFin)	Devuelve una cadena comprendida entre los dos índices

StringBuilder: modificaciones

- Como habíamos indicado, la clase **StringBuilder** permite modificar cadenas de sus instancias.
- En el siguiente código utilizamos varios métodos que permiten realizar modificaciones.

```
StringBuilder texto3 = new StringBuilder("Otra prueba");
```

```
texto3.append(" mas");
```

```
System.out.println("Texto 3 es: " + texto3);
```

```
texto3.insert(2, "W");
```

```
System.out.println("Y ahora es: " + texto3);
```

```
texto3.reverse();
```

```
System.out.println("Y ahora: " + texto3);
```

```
System.out.println("En mayúsculas: "  
+ texto3.toString().toUpperCase());
```

```
Texto 3 es: Otra prueba mas  
Y ahora es: OtWra prueba mas  
Y ahora: sam abeurp arWtO  
En mayúsculas: SAM ABEURP ARWTO
```

StringBuilder: ejercicio

1.- Crea un proyecto que realice la siguiente operación. Todas las operaciones se deben realizar con **String** y con **StringBuilder**

- Prueba el funcionamiento de todos los métodos de **StringBuilder** e intenta hacer la misma operación con un objeto **String**.

2.- Analiza en cada caso las diferencias y las equivalencias en la realización de las distintas operaciones según utilices **String** o **StringBuilder**.

Tipos envoltorio o Wrapper

- Java posee tipos estructurados de datos que tienen correspondencia con cada uno de los tipos primitivos.
- Por ejemplo, existen una clase llamada **Integer** que envuelve el tipo **int** y le añade métodos y propiedades que son de gran utilidad.
- Estas clases **wrapper** están contenidas en el paquete **java.lang** por lo que no es necesario que sean importadas.
- De la misma forma que la clase *String*, las clases envoltorios tienen valores inmutables. Esto quiere decir que cada vez que cambiamos el valor de una variable de tipo **wrapper**, estamos construyendo un nuevo objeto.

Tipo primitivo	Wrapper
byte	Byte
short	Short
integer	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Wrapper: creación de objetos

- Los *wrapper* como objetos que necesitan el uso de constructores para su creación, así un **Integer** se creará de la siguiente forma:

Integer edad=new Integer(23);

- No es necesario el uso del constructor, se puede crear el objeto y darle valor de forma directa con una variable o literal de su tipo primitivo correspondiente:

Integer edad=23;

- Pero usemos el constructor o no, si utilizamos la clase **Integer** estamos creando objetos, por lo tanto su comportamiento será diferente de su tipo primitivo correspondiente.

Los otros *wrapper* funcionan de forma similar a **Integer**

Wrapper: asignación

- Un tipo primitivo y su *wrapper* correspondiente permiten asignaciones entre ellos de forma directa:

```
int edad_i=30;  
Integer edad_I=40;
```

```
double altura_d=1.70;  
Double altura_D=1.78;
```

```
//Se puede usar métodos para extraer el tipo primitivo de su wrapper o realizar conversiones  
altura_d=altura_D.doubleValue();  
edad_i=altura_D.intValue();
```

```
//Se poder realizar asignaciones directas entre un tipo primitivo y su wrapper  
edad_i=edad2_I;  
edad_I=edad1_i;
```

```
altura_d=altura_D;  
altura_D=altura_d;
```

```
//Los objetos wrapper no permiten conversiones implícitas  
altura1=edad1; //correcto, son tipos primitivos  
altura2=edad1; //error, altura2 es un wrapper
```

Los objetos **wrapper** no permiten las conversiones implícitas

Wrapper: métodos

• Métodos de instancia:

1. Para extraer del wrapper el dato numérico de un tipo primitivo dado:
 - `objInteger.intValue()` //devuelve un elemento de tipo int
 - `objInteger.doubleValue()` //devuelve un elemento de tipo double
 - Etc.
2. Para convertir un wrapper en String:
 1. `objInteger.toString()` //devuelve un elemento de tipo String

• Método de clase:

1. Para conocer el valor máximo y mínimo de un tipo dado:
 - `Integer.MAX_VALUE` //devuelve el valor máximo de int
 - `Integer.MIN_VALUE` //devuelve el valor mínimo de int
1. Para convertir un String al tipo primitivo equivalente:
 - `Integer.parseInt(valorString)` //devuelve un elemento de tipo int

Wrapper: métodos

//Muestra el valor máximo del tipo int

```
System.out.println("El valor máximo de Integer es "+Integer.MAX_VALUE);
```

//Muestra el valor mínimo del tipo int

```
System.out.println("El valor mínimo de Integer es "+Integer.MIN_VALUE);
```

```
Double valor_double=1.67;
```

//Conversión de Double a String

```
String edad_string=valor_double.toString();
```

//Conversión de String a tipo wrapper

```
valor_double=Double.parseDouble("34.5");
```

```
Float valor_float=Float.parseFloat("5.6");
```

```
String valor_string="3";
```

```
Integer edad=Integer.parseInt(valor_string);
```

//Muestra el valor máximo del tipo int

```
System.out.println("El valor máximo de Integer es "+Integer.MAX_VALUE);
```

//Muestra el valor mínimo del tipo int

```
System.out.println("El valor mínimo de Integer es "+Integer.MIN_VALUE);
```

Ámbito de variables y constantes

- Cuando definimos una variable o una constante, está tiene un ámbito de existencia.
- Un bloque de instrucciones entre llaves define un ámbito.
- Las variables definidas en un bloque solo son visibles en ese bloque y en los posibles bloques que existan dentro del primero.

