

Introducción a JAVA

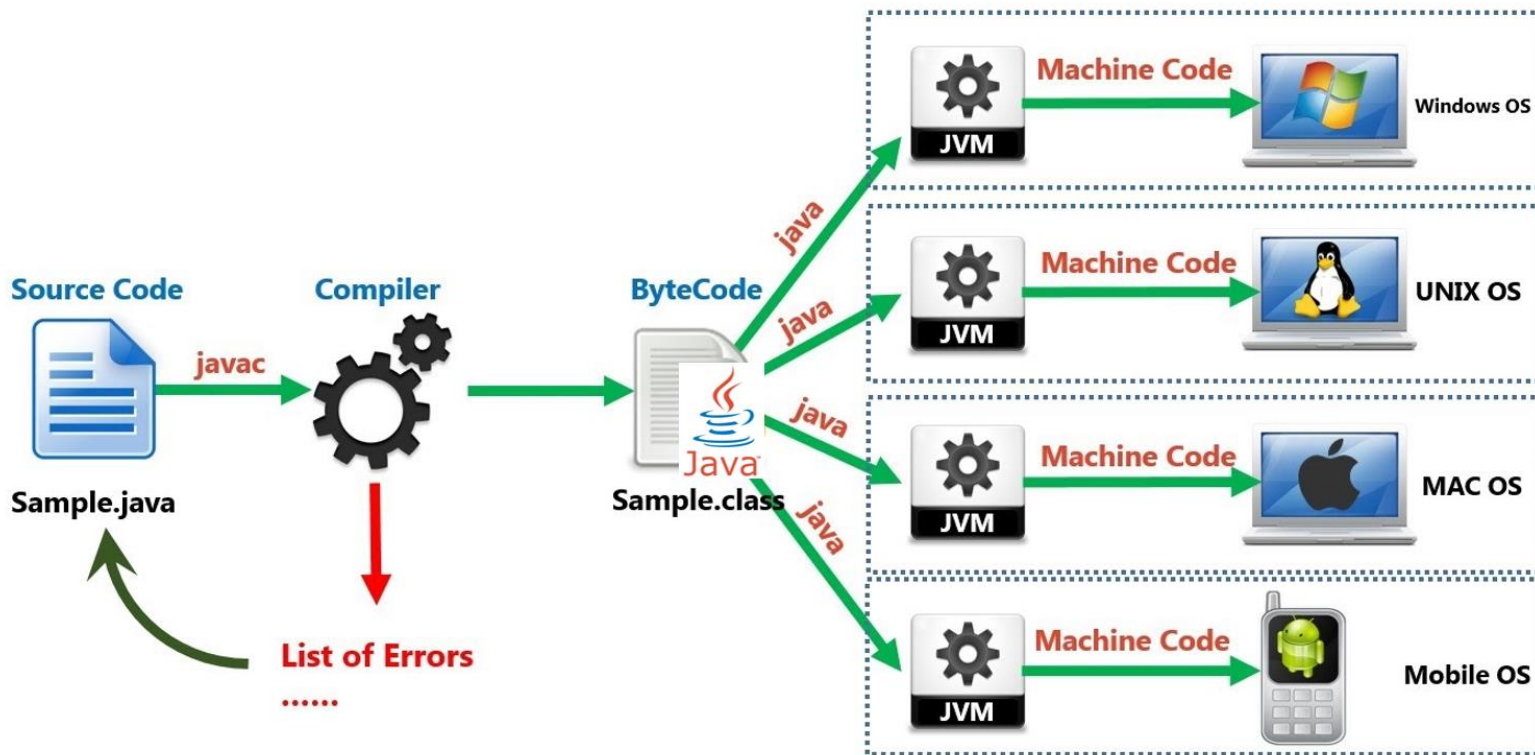
0.1 Java

Aspectos Básicos

Qué es Java?

- Java es un lenguaje de programación presentado en 1995, basado en clases y orientado a objetos.
- Uno de los objetivos de Java es ser multiplataforma, “*write once, run everywhere*”. El código puede ejecutar en cualquier plataforma con Java compatible sin recompilar.
- En 2022 sigue siendo uno de los lenguajes de programación más usados.
- Muy usado en aplicaciones cliente-servidor en aplicaciones web.
- No es un lenguaje totalmente orientado a objeto, tiene también **tipos elementales**.

Proceso de ejecución



Variables y Constantes

[<modificador>]* <tipo> <identificador> [,<identificador>];

```
int j;
```

```
int i, I, j101;
```

```
static char fin = '1';
```

```
final boolean eureka = true;
```

En la declaración se establece el nombre, tipo alcance y se determina si la variable es constante o no.

Tipos de Datos y Tipos de Elementales

Factorizar Propiedades. Todas las variables de un tipo comparten una misma representación, toman valores de un mismo conjunto y pueden participar de las mismas operaciones.

Efectuar Controles. El lenguaje establece restricciones que aseguran la consistencia entre los operadores provistos y los operandos. Estas restricciones van a ser controladas por el compilador o en ejecución.

Administrar la Memoria. El compilador decide cuánto espacio de almacenamiento va a requerir cada dato en ejecución, de acuerdo a su tipo.

Tipos de Datos y Tipos de Elementales

Category	Types	Size (bits)	Minimum Value	Maximum Value	Precision	Example
Integer	byte	8	-128	127	From +127 to -128	byte b = 65;
	char	16	0	$2^{16}-1$	All Unicode characters ^[1]	char c = 'A'; char c = 65;
	short	16	-2^{15}	$2^{15}-1$	From +32,767 to -32,768	short s = 65;
	int	32	-2^{31}	$2^{31}-1$	From +2,147,483,647 to -2,147,483,648	int i = 65;
	long	64	-2^{63}	$2^{63}-1$	From +9,223,372,036,854,775,807 to -9,223,372,036,854,775,808	long l = 65L;
Floating-point	float	32	2^{-149}	$(2 \cdot 2^{-23}) \cdot 2^{127}$	From 3.402,823,5 E+38 to 1.4 E-45	float f = 65f;
	double	64	2^{-1074}	$(2 \cdot 2^{-52}) \cdot 2^{1023}$	From 1.797,693,134,862,315,7 E+308 to 4.9 E-324	double d = 65.55;
Other	boolean	--	--	--	false, true	boolean b = true;
	void	--	--	--	--	--

fuelle: https://en.wikibooks.org/wiki/Java_Programming/Primitive_Types

Expresiones

Operadores relacionales

igual	==
distinto	!=
menor	<
menor o igual	<=
mayor	>
mayor o igual	>=

Operadores booleanos

Conjunción	&&
Disyunción	
Negación	!

Conversión

byte → short → int → long → float → double

Reglas de Precedencia y Asociatividad

++, --, !, - (unario), + (unario), type-cast

*, /, %

+, -

<, >, <=, >=

==, !=

&&

||

=, +=, -=, *=, /=, %=

Asignación

```
<ident> = <expresion>;
```

```
<tipo> <ident> = <exp> [, <ident> = <exp>];
```

```
boolean estado;
```

```
estado = true;
```

```
int a = 3, b = 4;
```

Cuando la expresión que aparece a la derecha de una asignación no coincide con el tipo de la variable que está a la izquierda puede producirse una **conversión automática** o un **error de compilación**.

El error puede salvarse mediante una operación de **casting**.

Asignación

`<ident> = <expresion>;`

`<tipo> <ident> = <exp> [, <ident> = <exp>];`

`boolean estado;`

`estado = true;`

`int a = 3, b = 4;`

Declaración

Asignación

Declaración y
Asignación

Cuando la expresión que aparece a la derecha de una asignación no coincide con el tipo de la variable que está a la izquierda puede producirse una **conversión automática** o un **error de compilación**.

El error puede salvarse mediante una operación de **casting**.

Conversión Automática de Tipos

Si al hacer la conversión de un tipo a otro se dan las 2 siguientes premisas:

- Los dos son tipos compatibles.
- El tipo de la variable destino es de un rango mayor al tipo de la variable que se va a convertir.

Entonces, la **conversión entre tipos es automática**.

Conversión Automática de Tipos

Si al hacer la conversión de un tipo a otro se dan las 2 siguientes premisas:

- Los dos son tipos compatibles.
- El tipo de la variable destino es de un rango mayor al tipo de la variable que se va a convertir.

Entonces, la **conversión entre tipos es automática**.

```
int a = 1;
```

```
long b = a;
```

Casting

Cuando el tipo a convertir está fuera del rango del tipo al que se quiere convertir entonces no es posible la conversión automática.

El programador se ve obligado a realizar una **conversión explícita**, que se denomina ***casting***.

La sintaxis es:

```
destino = (tipo_destino) valor;
```

Puede haber pérdida de precisión o incluso se pueden obtener valores erróneos.

Casting

Cuando el tipo a convertir está fuera del rango del tipo al que se quiere convertir entonces no es posible la conversión automática.

El programador se ve obligado a realizar una **conversión explícita**, que se denomina ***casting***.

La sintaxis es:

```
destino = (tipo_destino) valor;
```

Puede haber pérdida de precisión o incluso se pueden obtener valores erróneos.

```
int p = 37;  
short j = (short) p;
```

Casting

Cuando el tipo a convertir está fuera del rango del tipo al que se quiere convertir entonces no es posible la conversión automática.

El programador se ve obligado a realizar una **conversión explícita**, que se denomina ***casting***.

La sintaxis es:

```
destino = (tipo_destino) valor;
```

Puede haber pérdida de precisión o incluso se pueden obtener valores erróneos.

```
int p = 37;  
short j = (short) p;
```

```
int a = 40000;  
short j = (short) a;
```

```
j == -25536
```


Estructuras de Control en Java: Instrucciones

```
< instruccion > ::= < declaración de variable > |  
    < expresion > ; |  
    < bloque > |  
    < instruccion if > |  
    < instruccion while > |  
    < instruccion for > |  
    < instruccion switch > |  
    < instruccion try > |  
    < instruccion return > |  
    < break > | < continue > |
```

Estructuras de Control en Java: Bloques

```
< bloque> ::= { [< instruccion > ]* }  
{  
    promedio = total / n ;  
    System.out.print("El promedio es ");  
    System.out.println(promedio); }  
{  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
}  
}
```

Estructuras de Control en Java: Bloques

Un bloque define un nuevo **ambiente de referenciamiento**.

Las variables declaradas dentro de un bloque son **locales** y no son visibles fuera de él.

Un mismo nombre no puede ligarse a dos variables en el mismo bloque ni en bloques anidados.

Una variable puede ser referenciada a partir de la instrucción que sigue a su declaración.

Adoptaremos la convención de declarar las variables de un bloque al principio e inicializarlas explícitamente.

Estructuras de Control en Java: Condicional (if)

```
< instruccion if > ::=    if (< expresion booleana >)  
                           < instruccion >  
                           [else  
                             < instrucción >]
```

Estructuras de Control en Java: Condicional (if)

```
if (x > y)
    max = x;
else
    max = y;
```

```
if (x>y){
    max=x;
    min = y;}
else{
    max = y;
    min = x;
}
```

```
if (x>y){
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

Estructuras de Control en Java: Condicional (if)

```
if (x > y)
    if (x > z)
        max = x;
    else
        max = z;
else
    if (y > z)
        max = y;
    else
        max = z;
```

```
if (nota>9)
    estado = 'A';
else if (nota>7)
    estado = 'B';
else if (nota >4)
    estado = 'C';
else if (nota < 4)
    estado = 'D';
```

Estructuras de Control en Java: Condicional (if)

```
estado = 'B';  
if (promedio > 7)  
    if (inasistencias<3)  
        estado = 'A';  
    else  
        estado = 'C';
```

```
estado = 'B';  
if (promedio > 7){  
    if (inasistencias<3)  
        estado = 'A'; }  
else  
    estado = 'C';
```

Estructuras de Control en Java: Condicional (if)

```
estado = 'B';  
if (promedio > 7){  
    if (inasistencias<3)  
        estado = 'A';  
    else  
        estado = 'C';  
}
```

```
estado = 'B';  
if (promedio > 7){  
    if (inasistencias<3)  
        estado = 'A'; }  
else  
    estado = 'C';
```


Estructuras de Control en Java: Condicional (switch)

`<instruccion switch>::=`

```
switch (<expression>) {  
    [ case <constante> : <instruccion> ]*  
    default: <instruccion>  
}
```

Estructuras de Control en Java: Condicional (switch)

```
switch ( nota ) {  
    case 10:  
    case 9:  
        estado = 'A';  
        break;  
    case 8:  
    case 7:  
        estado = 'B';  
        break;  
    case 6:  
    case 5:  
        estado = 'C' ;  
        break;  
    default: estado = 'D'; }
```

Estructuras de Control en Java: Iteración (while)

```
< instruccion while > ::=  
    while (<expresion booleana>)  
        <instrucción>
```

```
< instruccion do while> ::=  
    do  
        <instrucción>  
    while (<expresion booleana>)
```

Estructuras de Control en Java: Iteración (while)

```
int numero;  
int digitos = 0;  
System.out.println ("Ingrese el numero");  
numero = ES.leerEntero ();  
while ( numero > 0 ) {  
    numero /=10;  
    digitos++;  
}  
System.out.println(digitos);
```

Estructuras de Control en Java: Iteración (while)

```
int numero;  
int digitos = 0;  
System.out.println ("Ingrese el numero");  
numero = ES.leerEntero ();  
do {  
    numero /=10;  
    digitos++;  
} while ( número > 0 );  
System.out.println(digitos);
```

Estructuras de Control en Java: Iteración (for)

```
< instruccion for > ::=  
    for ([<asignacion>]; [<expresion>] ; [<expresion>])  
        <instrucción>
```

Estructuras de Control en Java: Iteración (for)

```
< instruccion for > ::=  
    for ([<asignacion>]; [<expresion>] ; [<expresion>])  
        <instrucción>
```

```
int n;  
for ( n = 1 ; n <= 10 ; n++ )  
    System.out.println( n*n );
```

```
for ( int n = 1 ; n <= 10 ; n++ )  
    System.out.println( n*n );
```

Estructuras de Control en Java: Iteración (for)

```
sum = 0 ;  
for ( n = 1 ; n <= 10 ; n++ )  
    sum = sum + n ;  
for ( n = 1, sum = 0 ; n <= 10 ; n++ )  
    sum = sum + n ;  
for (n=1,sum=0; n<=10; sum=sum+n,n++);  
for ( n = 1, sum = 0 ; sum <= 100 ; n++ )  
    sum = sum + n ;  
for ( int n = 1, sum = 0 ; sum <= 100 ; n++ )  
    sum = sum + n ;
```


Estructura de un programa en Java

La unidad básica de programación en Java es la **clase**.

Un programa en Java está constituido por una colección de clases .

La implementación de una clase consiste en definir sus **miembros**:

- **Atributos**: variables de instancia y de clase
- **Servicios**: constructores y métodos

Estructura de un programa en Java: Métodos

Sintaxis

[< Modificador >]* < Tipo del Resultado >

< Identificador > ([< Parámetros Formales >]*)

{ < bloque > }

Estructura de un programa en Java: Métodos

Sintaxis

[< Modificador >]* < Tipo del Resultado >

< Identificador > ([< Parámetros Formales >]*)


{ < bloque > }

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

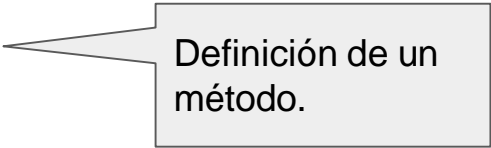
Estructura de un programa en Java: Métodos

Para que un programa en Java pueda ejecutarse es necesario definir una clase que incluya un método llamado `main()`.

```
class Hello {  
  
    public static void main(String [] args){  
        System.out.println("Hello World");  
  
    }  
  
}
```



definición de una
clase



Definición de un
método.

Estructura de un programa en Java: Métodos

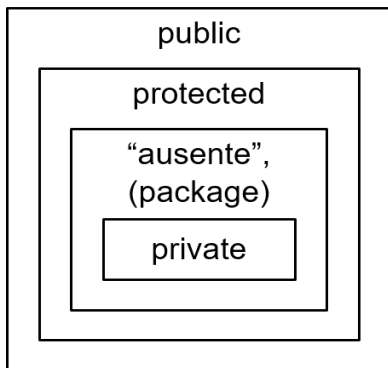
- La primera línea define una clase llamada **Hello**.
- La segunda clase define el método **main**, asociado a los modificadores **public** y **static**. Es importante que no omitan ni alteren el encabezamiento del **main**.
- La palabra **void** indica que el método **main** no retorna ningún valor.
- La forma (**String args[]**) es la definición de los argumentos que recibe el método **main**.
- La instrucción **System.out.println** muestra un literal en pantalla.

Identificadores

- Identificador **private**: a los métodos private solamente se puede acceder desde dentro de la clase.
- Identificador **public**: por su parte, a los métodos public puede acceder cualquiera, tanto desde dentro como desde fuera de la clase.
- **Tipo devuelto**: Un método puede devolver un valor al usuario. Puede tratarse de un tipo de datos simple, como int o una clase. Un tipo devuelto void indica que no se devolverá ningún valor.

Identificadores

Modificador de Acceso	Visibilidad
private	Sólo visible en la clase
Sin modificador (omitido)	En clase y el paquete
protected	En clase su-clases y paquete
public	Desde todas partes



Modificador	Clase	Package	Subclass	World
private	si	no	no	no
Sin modificador	si	si	no	no
protected	si	si	si	no
public	si	si	si	si

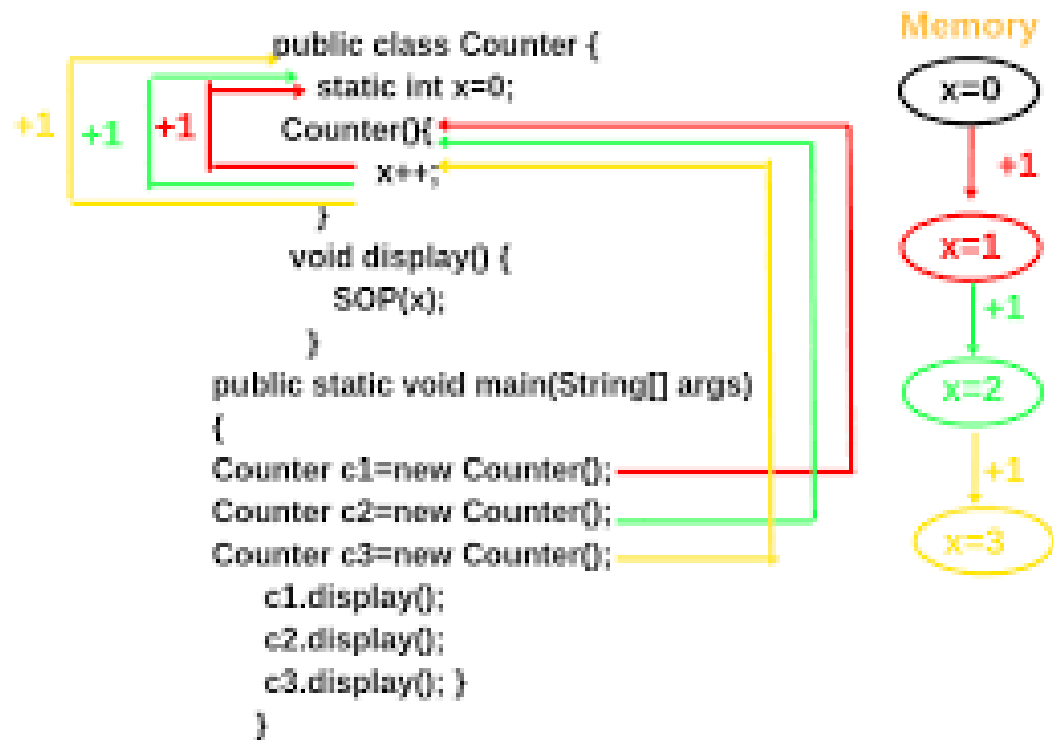
- **final** determina que un atributo no puede ser sobreescrito o redefinido. O sea: no funcionará como una variable “tradicional”, sino como una constante.
- Toda constante declarada con final ha de ser inicializada en el mismo momento de declararla.
- final también se usa como palabra clave en otro contexto: una clase final (final) es aquella que no puede tener clases que la hereden. Lo veremos más adelante cuando hablemos sobre herencia.

TERMINAL y NETBEANS

- ▶ Vamos a codificar el código anterior y en un archivo Hola.java
 - ▶ Ejecutar `javac Hola`
 - ▶ Ejecutar `javac Hello`
 - ▶ Ejecutar `java Hola.java`
-
- ▶ ... y que IDE podríamos utilizar

Identificadores

- Los atributos miembros de una clase pueden ser atributos de clase o atributos de instancia
- Se dice que son atributos de clase si se usa la palabra clave static: en ese caso la variable es única para todas las instancias (objetos) de la clase (ocupa un único lugar en memoria).
- A veces a las variables de clase se les llama variables estáticas. Si no se usa static, el sistema crea un lugar nuevo para esa variable con cada instancia (la variable es diferente para cada objeto). En el caso de una constante no tiene sentido crear un nuevo lugar de memoria por cada objeto de una clase que se cree. Por ello es adecuado el uso de la palabra clave static.



Procesador Número

ProcesadorNumero

sumaDigs (n:entero) : entero

estaDig (n:entero, d:entero) : booleano

Procesador Número

ProcesadorNumero

sumaDigs (n:entero) : entero

estaDig (n:entero, d:entero) : booleano

```
class ProcesadorNumero {  
    public static int sumaDig(int n ){  
        // Retorna la suma de los dígitos del número n  
        ...  
    }  
    public static boolean estaDig (int n, int d ){  
        //Retorna true si y solo si d es un dígito del número n  
        ...  
    }  
}
```

Procesador Número

ProcesadorNumero

sumaDigs (n:entero) : entero

estaDig (n:entero, d:entero) : booleano

class ProcesadorNumero {

public static **int** sumaDig(**int** n){

// Retorna la suma de los dígitos del número n

...

}

public static **boolean** estaDig (**int** n, **int** d){

//Retorna true si y solo si d es un dígito del número n

...

}

Clases como unidad
de programación

Modificadores
de visibilidad

Comentarios
de una sola
línea.

Procesador Número

```
class ProcesadorNumero{  
  
    public static int sumaDig(int n){  
        //Retorna la suma de los dígitos del número n  
        int s=0;  
        while (n>0) {  
            s = s + n % 10;  
            n = n/10;  
        }  
        return s;  
    }  
    ...  
}
```

Procesador Número

```
class ProcesadorNumero{
```

Clases como unidad
de programación

```
    public static int sumaDig(int n){
```

```
        //Retorna la suma de los dígitos del número n
```

```
        int s=0;
```

```
        while (n>0) {
```

```
            s = s + n % 10;
```

```
            n = n/10;
```

```
        }
```

```
        return s;
```

```
    }
```

```
    ...
```

```
}
```

Tipo Elemental
int

Parámetros y
Variables
Locales

Instrucción **while**

La instrucción
de **return**

Asignaciones y expresiones,
operadores y operandos

Procesador Número - Tester

Para que un programa en Java pueda ejecutarse es necesario que una clase incluya un método llamado **main()**.

```
class Tester{  
    public static void main (String args[]) {  
        int s = sumaDig(25036);  
        System.out.println("La suma es "+s);  
    }  
}
```

Procesador Número

```
public class ProcesadorNumero{  
    ...  
    public static boolean estaDig(int n,int d ){  
        /* Retorna true si y solo si d  
        es un dígito del número n */  
        boolean esta = false;  
        while (n>0) && !esta {  
            if (d == n % 10)  
                esta = true;  
            n = n/10;  
        }  
        return esta;  
    }  
}
```

Procesador Número

- ▶ public class ProcesadorNumero{
 - ▶ ...
 - ▶ public static **boolean** estaDig(int n,int d){
 - ▶ /* Retorna true si y solo si d

Tipo
elemental
boolean

es un dígito del número n */

boolean esta = false;

while (n>0) && !esta {

if (d == n % 10)

 esta = true;

 n = n/10;

}

return esta;

}

Condicional **if**

Operadores
relacionales y
lógicos.

Comentario
de múltiples
líneas.

System.out.println()

El `System.out.print()` es un método muy utilizado para imprimir en la consola o en la salida estándar.

Este método a veces se denomina método de línea de impresión.

Además de imprimir en la consola, el método `println()` mueve el cursor a una nueva línea.

```
public class PrintDemo {  
    public static void main(String args[]) {  
        int i = 10;  
        String s = "hello";  
        char c = '0';  
        System.out.println(i + s + c);  
    }  
}
```

¿Preguntas?

Paquetes en Java

- ▶ Un Paquete en Java es un contenedor de clases que permite agrupar las distintas partes de un programa y que por lo general tiene una funcionalidad y elementos comunes, definiendo la ubicación de dichas clases en un directorio de estructura jerárquica.
- ▶ Algunas ventajas son:
- ▶ Agrupamiento de clases con características comunes.
- ▶ Reutilización de código al promover principios de programación orientada a objetos como la encapsulación y modularidad.
- ▶ Mayor seguridad al existir niveles de acceso.

► Para importar paquetes de clases ya definidos usar import

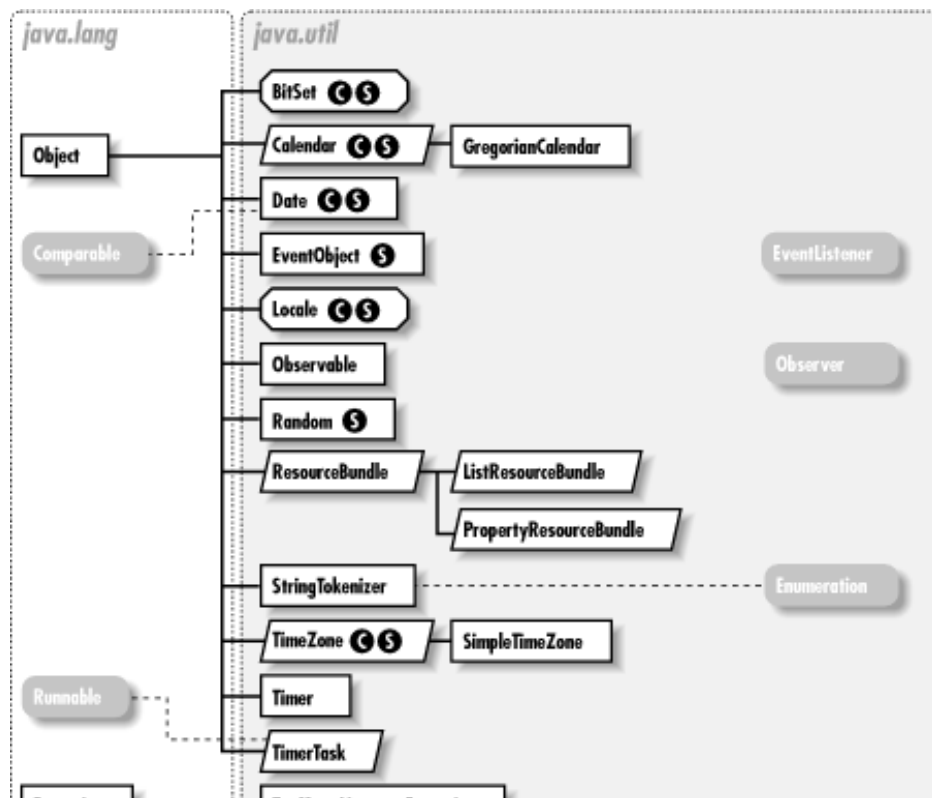
► Ejemplos:

- `import java.util.*` // incluye un conjunto de contenedores definidos en java como ArrayLists, Vectors, HashMaps, Lists, etc
- `import java.io.*` // incluye clases de clases para entrada/salida
- `import java.lang.*` // import no necesario

► Si nombre de clase en un paquete es igual a otra en otro paquete y ambos son importados, de debe usar el nombre de paquete con clase. Ejemplo `java.util.Date` y `java.sql.Date`

Paquetes

- ▶ Para garantizar nombres de paquetes únicos Sun recomienda dar nombres en sentido inverso a dominios en urls
- ▶ Los archivos de paquetes creados deben seguir jerarquía de directorios dados en secuencia dada en nombre paquete
 - uncuyo.pdp.main.app : Clases
 - uncuyo.pdp.main.utils
 - uncuyo.pdp.app.main



Creando paquetes

- Para poner una clase en un paquete se agrega la sentencia de package nombre paquete

```
package uncuyo.app.main;  
public class Cuenta{  
    ...  
}
```

- Cuando la sentencia package no se agrega la clase pertenece al paquete por defecto

0.3 Java

String y Random

La clase String

- La clase **String** provista por Java brinda facilidades para almacenar y procesar cadenas de caracteres.
- El **estado interno** de una instancia de tipo **String** es una secuencia de caracteres.
- Los objetos de tipo **String** son inmutables, quiere decir que cuando se opera con **String** el resultado es siempre un nuevo objeto de esta clase.
- Una variable de tipo **String** referencia a un objeto de este tipo.

La clase String

Declaración, creación e inicialización

```
String cad = "Buenas Buenas ...";
```

```
String var1;
```

```
var1 = new String("Buenas Buenas ..." );
```

Ahora la variable cad puede recibir mensajes que se ligarán a métodos provistos por la clase String. Ninguno de estos métodos modifica el estado interno de la variable.

La clase String

length(): entero	retorna la cantidad de caracteres de una cadena.
toLowerCase(): String	retorna la misma cadena pero con todos los caracteres en minúscula.
toUpperCase(): String	retorna la misma cadena pero con todos los caracteres en mayúscula.
trim(): String	retorna la misma cadena pero sin los espacios del principio y del final.
charAt(pos:entero): caracter	retorna el caracter que está en la posición que corresponde al argumento.

La clase String

<code>substring(ini: entero): String</code>	retorna la subcadena a partir de la posición ini .
<code>substring(ini:entero, fin:entero): String</code>	retorna la subcadena a partir de la posición ini hasta la anterior a la posición fin .
<code>indexOf(A: String): entero</code>	retorna la posición de la primera aparición de la subcadena A en la cadena.
<code>lastIndexOf(A: String): entero</code>	retorna la posición de la última aparición de la subcadena A en la cadena.
<code>compareTo(A: String): entero</code>	Retorna 0 si las cadenas son iguales , o un número positivo o un número negativo según el orden alfabético (minúsculas < mayúsculas).

La clase String

- En Java el mínimo valor para un índice es 0 y corresponde al primer carácter de la cadena.
- Los métodos **indexOf** y **lastIndexOf** retornan **-1** si la subcadena no aparece en la cadena.
- La comparación entre variables de tipo **String** no se realiza a través del operador relacional **==**, sino con los métodos **equals(A:String)** o **compareTo(A:String)**.

La clase String

- En Java el mínimo valor para un índice es 0 y corresponde al primer carácter de la cadena.
- Los métodos **indexOf** y **lastIndexOf** retornan **-1** si la subcadena no aparece en la cadena.
- La comparación entre variables de tipo **String** no se realiza a través del operador relacional **==**, sino con los métodos **equals(A:String)** o **compareTo(A:String)**.

```
String s1 = new String("Hello world");  
String s2 = new String("Hello world");  
System.out.println(s1==s2);    FALSE
```

La clase String

Ejemplos

```
String cad = "Buenas Buenas...";
```

```
cad.length()  
retorna 16
```

```
cad.toLowerCase()  
retorna "buenas buenas..."
```

```
cad.toUpperCase()  
retorna "BUENAS BUENAS..."
```

```
"  Buenas buenas...  ".trim()  
retorna "Buenas buenas..."
```

```
cad.charAt(1)  
retorna 'u'
```

```
cad.charAt(100)  
StringIndexOutOfBoundsException: String index out of range: 100 (in java.lang.String)
```

La clase String

Ejemplos

```
String cad = "Buenas Buenas...";
```

```
cad.substring(3)  
retorna "nas Buenas..."
```

```
cad.substring(3,5)  
retorna "na"
```

```
cad.substring(3,25)  
StringIndexOutOfBoundsException: String index out of range: 25 (in java.lang.String)
```

La clase String

Ejemplos

```
String cad = "Buenas Buenas...";
```

```
cad.indexOf("Bue")  
retorna 0
```

```
cad.indexOf("Nue")  
retorna -1
```

```
cad.lastIndexOf("Bue")  
retorna 7
```

```
cad.compareTo("Buenos Aires")  
retorna -14
```

La clase String

Mostrar

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

Concatenar

```
System.out.println ("El ganador es "+nombre);
```

Conversión implícita y Concatenación

```
System.out.println ("El puntaje es "+10);
```

```
int i =0;
```

```
System.out.println ("El puntaje es "+i);
```

La clase String

Conversión explícita

Para convertir un número en una cadena de caracteres se emplea el método `valueOf`.

```
int valor=10;  
String str= String.valueOf(valor);
```

Para convertir una cadena en un número entero, primero quitamos los espacios en blanco al principio y al final y usamos el método `parseInt` de la clase `Integer`

```
String str=" 12 ";  
int numero=Integer.parseInt(str.trim());
```

La clase String

Conversión explícita

Para convertir un número en una cadena de caracteres se emplea el método `valueOf`.

```
int valor=10;  
String str= String.valueOf(valor);
```

Notar que `valueOf` y `parseInt` son métodos estáticos (**static**) porque son métodos que le pedimos a la clase y no a un objeto de la clase.

Para convertir una cadena en un número entero, primero quitamos los espacios en blanco al principio y al final y usamos el método `parseInt` de la clase `Integer`

```
String str=" 12 ";  
int numero=Integer.parseInt(str.trim());
```


La clase String

Conversión explícita

Para convertir un string en número decimal se requieren dos pasos: convertir la cadena en un objeto de la clase **Double**, mediante el método **valueOf**, y a continuación convertir el objeto de la clase **Double** en un **tipo primitivo double** mediante el método **doubleValue**.

```
String str=" 12.35 ";  
double num=Double.valueOf(str).doubleValue();
```

La clase String

Conversión explícita

Para convertir un string en número decimal se requieren dos pasos: convertir la cadena en un objeto de la clase **Double**, mediante el método **valueOf**, y a continuación convertir el objeto de la clase **Double** en un **tipo primitivo double** mediante el método **doubleValue**.

```
String str=" 12.35 ";  
double num=Double.valueOf(str).doubleValue();
```

Notar que **valueOf** de la clase **Double** es un método estático (**static**).

La clase Random

La clase Random

Un **generador** de números aleatorios se utiliza cuando se desea simular **situaciones de azar**.

La clase **Random** de Java es un generador de números **pseudo-aleatorios**.

Los números no son realmente aleatorios porque se obtienen a través de un algoritmo que genera una secuencia distribuida uniformemente, a partir de una **semilla** inicial.

La clase Random

La clase brinda dos constructores para crear objetos **Random**, uno sin parámetros y otro con un parámetro que establece el valor de la semilla.

Si no se especifica parámetro, el constructor usa la hora actual del sistema como semilla, lo que disminuye la posibilidad de obtener secuencias de números repetidas.

Cuando se crea un objeto **Random** con una semilla como parámetro, se puede obtener a continuación lo que parece una secuencia aleatoria, pero si se vuelve a inicializar el objeto con la misma semilla se vuelve a obtener **la misma secuencia**.

La clase Random

1. Importar el paquete que incluye a la clase Random.

```
import java.util.Random;
```

2. Crear un objeto de la clase Random

```
Random rnd = new Random();
```

```
Random rnd = new Random(100);
```

3. Invocar uno de los métodos que generan un número aleatorio

```
rnd.nextInt();
```

```
rnd.nextInt(3);
```

```
rnd.nextFloat();
```

La clase Random

Ejemplo I

Para generar una secuencia de 10 números aleatorios entre 0.0 y 1.0 escribimos

```
for (int i = 0; i < 10; i++) {  
    System.out.println(rnd.nextDouble());  
}
```

La clase Random

Ejemplo II

Un jugador apuesta una cantidad de dinero y tira una moneda.

Si sale cara obtiene el doble de la cantidad apostada, pero si sale cruz pierde la mitad.

Implemente una simulación para el juego que parta de un valor inicial y lo actualice según en la moneda se obtenga cara o cruz, hasta que llegue a tener \$1 o se realicen 50 tiradas.

La clase Random

Ejemplo II

```
import java.util.Random;
import IP00.ES;
public class DOBLEoMITAD {
    public static void main (String arg[]) {
        Random gen;
        gen = new Random();
        System.out.print("Ingrese la apuesta ");
        int m = ES.leerEntero();
        int i = 0;
        int caracruz;
        ...
    }
}
```

La clase Random

Ejemplo II

```
while (i<50 && m > 1){  
    i++; caracruz = gen.nextInt(2);  
    if (caracruz==1){ //cara  
        m = m*2;  
        System.out.println (i+" cara " +m);  
    }  
    else{ //cruz  
        m = m / 2;  
        System.out.println (i+" cruz " +m);  
    }  
}
```

Arreglos

Ejemplo: Decidir si hubo al menos n días con temperaturas mínimas mayores a t

- `Tipo_de_variable[] Nombre_del_array = new Tipo_de_variable[dimensión];`
- ▶ Otra definición podría ser:
 - `Tipo_de_variable[] Nombre_del_array;`
 - `Nombre_del_array = new Tipo_de_variable[dimensión];`

Ejemplos:

- `byte[] edad = new byte[4];`
- `short[] edad = new short[4];`
- `int[] edad = new int[4];`
- `long[] edad = new long[4];`
- `float[] estatura = new float[3];`
- `double[] estatura = new double[3];`
- `boolean[] estado = new boolean[5];`
- `char[] sexo = new char[2];`
- `String[] nombre = new String[2];`

Administración de Memoria

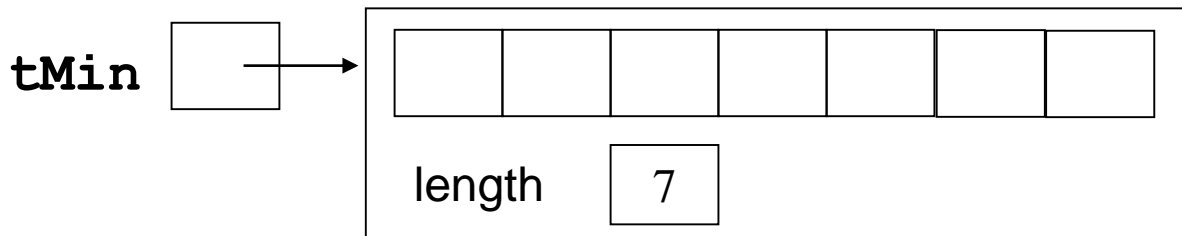
tMin 

```
float [] tMin;
```

En Java una variable declarada como un arreglo mantiene una **referencia**.

Se declara una variable **tMin** que referenciará a un arreglo con componentes de tipo float.

Administración de Memoria

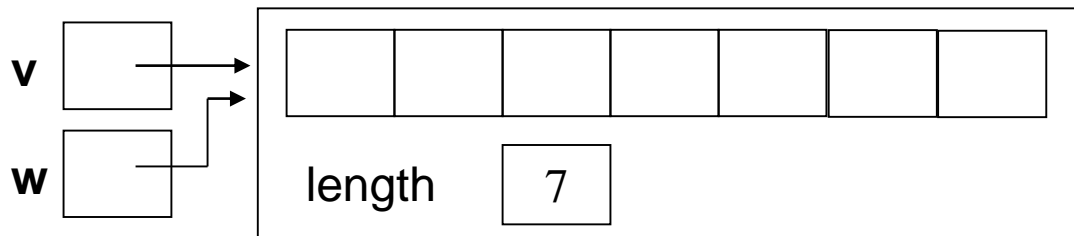


```
float [] tMin;  
tMin = new float[7];
```

Se crea un objeto y se asigna la referencia a la variable.

La variable mantiene una referencia ligada a un arreglo.

Administración de Memoria



```
float [] v,w;  
v = new float[5];  
w = v;
```

Una operación de asignación copia la dirección de memoria, no el contenido del arreglo

El operador relacional `==` compara referencias

Arreglos

Ejemplo: Decidir si hubo al menos n días con temperaturas mínimas mayores a t

```
int i = 0; int cont= 0;

while ((i<tMin.length) && cont < n)
    if (tMin[i]> t)
        cont++;
    i++;
}
boolean hubo = cont == n;
```

Arreglos

Ejemplo: Decidir si hubo exactamente n días con temperaturas mínimas mayores a t

```
int i = 0; int cont= 0;

while ((i<tMin.length)&& (cont<=n))
    if (tMin[i]> t)
        cont++;
    i++;
}
boolean hubo = cont == n;
```


Arreglos

Ejemplo: Decidir si hubo al menos n días consecutivos con temperaturas mínimas mayores a t

```
int i = 0; int cont= 0;

while ((i<tMin.length) && cont < n)
    if (tMin[i]> t) cont++;
    else cont = 0;
    i++;
}
boolean hubo = cont == n;
```

Matrices



```
int[][] arreglo = new int[5][4];
```

```
int[][] matriz = new int[4][]; // Sólo se indica el número de renglones, pero no el de columnas, porque será diferente en cada renglón.
```

```
matriz[0] = new int[2]; // En el renglón 0, se construyen 2 columnas
```

```
matriz[1] = new int[3]; // El renglón 1 contiene 3 columnas
```

```
matriz[2] = new int[2]; // El renglón 2 contiene 2 columnas
```

```
matriz[3] = new int[4]; // El renglón 3 contiene 4 columnas.
```

```
int[][] matriz = { {1, 2}, {3, 4, 5}, {6, 7}, { 8, 9, 10, 11} };
```

Excepciones

Objetivos

- Mejorar la confiabilidad del código incorporando
- Escribir métodos para propagar excepciones.
- Implementar los bloques try-catch para atrapar y manejar las excepciones.
- Escribir clases de excepción programadas.
- Diferenciar las excepciones de tiempo de compilación y en tiempo de corrida

Definición

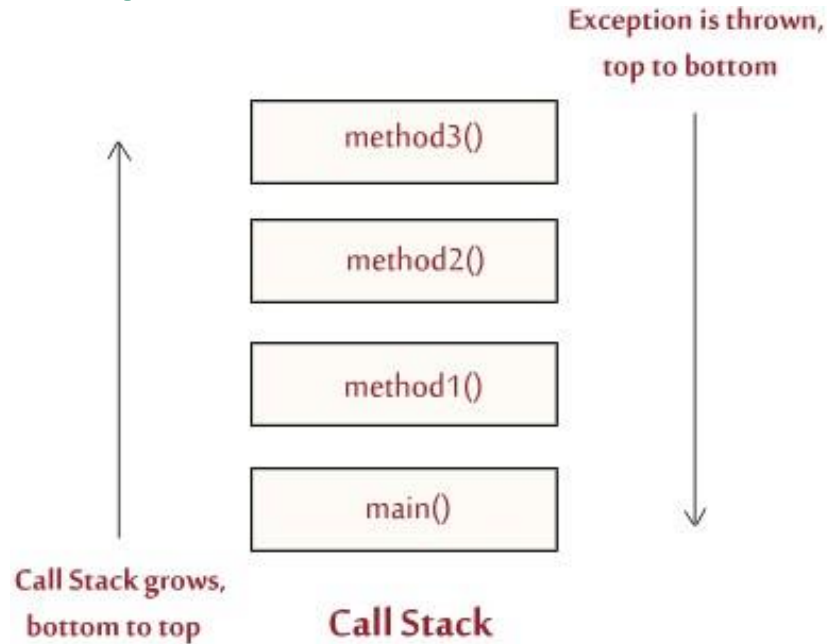
- Cuando un programa viola las restricciones semánticas del lenguaje, se produce un error y la maq. virtual comunica este hecho al programa mediante una excepción.
- Muchas clases de errores pueden provocar una excepción, desbordamiento de memoria, disco estropeado, intento de dividir por 0, acceder a un vector fuera de los límites, entre otros.
- Una **excepción** es un evento que se produce durante la ejecución de un programa, que interrumpe el flujo normal de las instrucciones del mismo.

- Se atrapa la excepción si una vez lanzada se interrumpe el flujo normal de ejecución y se ejecuta la rutina de manejo de excepciones.
- Cuando se crea una excepción la JVM recorre la pila de llamadas de métodos en busca de alguno que sea capaz de tratar la clase de excepción lanzada. Si ninguno de los métodos de la pila pueden la JVM muestra un mensaje de error y termina el programa.
- El uso de excepciones permite separar el código regular del manejo de las excepciones, propagar los errores al método invocante, agrupar los errores por tipo.

Manejo de las excepciones

- Cuando se produce un error dentro de un método, éste crea un objeto **exception**
- Este contiene la info sobre el error, tipo y el estado del programa cuando ocurrió.
- Lanzar una excepción es crear un objeto **exception**, también se puede lanzar explícitamente mediante la instrucción **throw**
- Después que el método crea la excepción, el sistema busca si hay una forma de manejarla.
- La búsqueda se realiza en la pila de llamadas que se realizaron hasta llegar al método que lanzó la excepción (call stack)

Propagación de la pila del llamado



Exception Propagation

Ejemplo de la generación del error

```
class Exp {  
    public static void main(String[] args){  
        Exp ob= new Exp(); ob.method1();  
    }  
  
    public void method1(){  
        method2();  
    }  
  
    public void method2(){  
        method3();  
    }  
  
    public void method3(){  
        System.out.println(100/0); //ArithmeticException se produce  
        System.out.println("Hello"); //Instruccion no ejecutada  
    }  
}
```

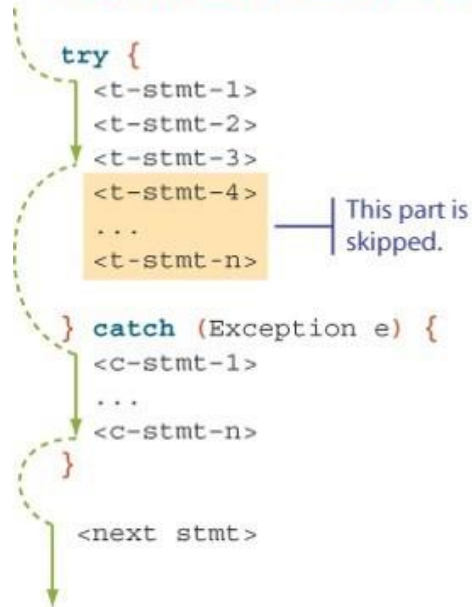
Ejemplo de generación de error

```
Exception in thread "main" java.lang.ArithmeticException: /by zero at  
  Exp.method3(Exp1.java:23)  
  at Exp.method2(Exp1.java:17)  
  at Exp.method1(Exp1.java:12) at  
  Exp.main(Exp1.java:6)
```

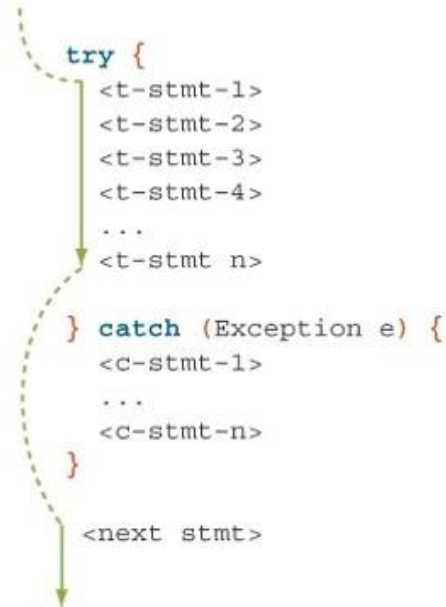
Try-catch

Exception

Assume `<t-stmt-3>` throws an exception.



No Exception



Ejemplo de try-catch

```
public class ExcepcionApp {  
    public static void main(String[] args) {  
        String str1="12";  
        String str2="0";  
        String respuesta;  
        int numerador, denominador, cociente;  
        try{  
            numerador=Integer.parseInt(str1);  
            denominador=Integer.parseInt(str2);  
            cociente=numerador/denominador;  
            respuesta=String.valueOf(cociente);  
        }catch(NumberFormatException ex){  
            ex.getMessage();  
        }  
    }  
}
```

Exception in thread "main" java.lang.ArithmeticException: / by
zero at ExcepcionApp.main(ExcepcionApp.java:10)

Try-catch

Hay dos métodos que podemos llamar para obtener información sobre una excepción:

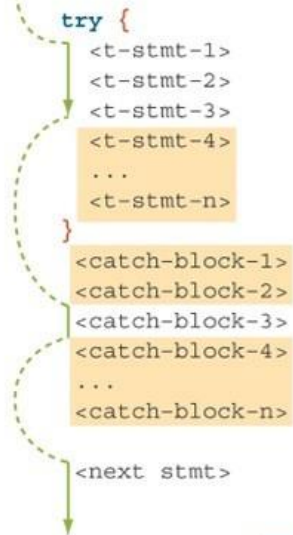
- getMessage()
- printStackTrace()

```
try {  
    ...  
} catch (NumberFormatException e){  
    System.out.println(e.getMessage());  
    System.out.println(e.printStackTrace());  
}
```

Multiples try-catch

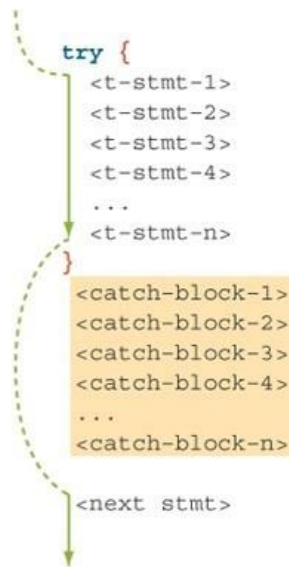
Exception

Assume `<t-stmt-3>` throws an exception and `<catch-block-3>` is the matching catch block.



Skipped portion

No Exception



Múltiples Try-catch

Una sentencia try-catch puede incluir varios bloques de captura, uno para c/ tipo de excepción.

```
try {  
    ...  
    age=Integer.parseInt(inputStr);  
    ...  
    val=cal.get(id);           //cal is a GregorianCalendar  
    ...  
} catch (NumberFormatException e){  
    ...  
} catch (ArrayIndexOutOfBoundsException e){  
    ...  
}
```

El bloque finally

- Hay situaciones en las cuales necesitamos tomar ciertas acciones sin importar si se produce una excepción o no.
- Las sentencias que deben ejecutarse siempre se ponen en el bloque final. Ej liberar recursos, arch, etc

El bloque finally

Exception

Assume `<t-stmt-i>` throws an exception and `<catch-block-i>` is the matching catch block.

```
try {  
  <t-stmt-1>  
  ...  
  <t-stmt-i>  
  ...  
  <t-stmt-n>  
}  
<catch-block-1>  
...  
<catch-block-i>  
...  
<catch-block-n>  
finally {  
  ...  
}  
<next statement>
```

Skipped portion

No Exception

```
try {  
  <t-stmt-1>  
  ...  
  <t-stmt-i>  
  ...  
  <t-stmt-n>  
}  
<catch-block-1>  
...  
<catch-block-i>  
...  
<catch-block-n>  
finally {  
  ...  
}  
<next statement>
```

Propagación de Excepciones

- En vez de capturar una excepción a través de una sentencia try-catch statement, podemos propagarla hacia el método llamador.
- En este caso la cabecera del método debe incluir la palabra reservada throws.

```
public int getAge() throws NumberFormatException{  
    ...  
    int age=Integer.parseInt(inputStr);  
    ...  
    return age;  
}
```

Propagando Excepciones

- Podemos escribir un método que propague la excepción directamente
- Para crear excepciones, def por el usuario, se usa throw para crear una nueva instancia de la excepción.
- La cabecera del método debe incluir la palabra throws.

```
public void doWork(int num) throws Exception{  
    ...  
    if (num!=val)    throw new Exception("Invalid val");  
    ...  
}
```

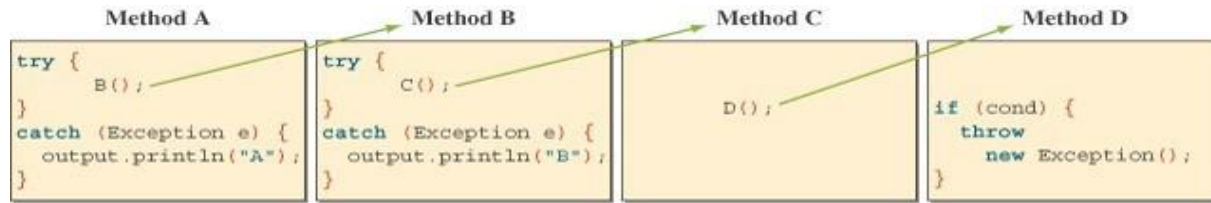
Propagando Excepciones

- Cuando un método puede propagar una excepción, directa o indirectamente, se llama lanzador de excepción
- Cada excepción lanzada puede ser de uno de los dos tipos,
 - captada (catcher)
 - propagada

Tipos de lanzadores de Excepciones

- Un atrapador de *excepciones* es un lanzador de excepciones que incluye un bloque catch que maneja la excepción lanzada, es decir la captura.
- Un propagador de excepciones no contiene ningún bloque catch que coincida con esa excepción
- Un método puede atrapar unas excepciones y propagar otras.

Llamados



Call Sequence



Stack Trace

