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 2019 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.**

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}	$(2-2^{-23})\cdot 2^{127}$	From 3.402,823,5 E+38 to 1.4 E-45	`float f = 65f;`
	`double`	64	2^{-1074}	$(2-2^{-52})\cdot 2^{1023}$	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`	--	--	--	--	--

fuente: 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	!

Evaluación

Completa vs En cortocircuito

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

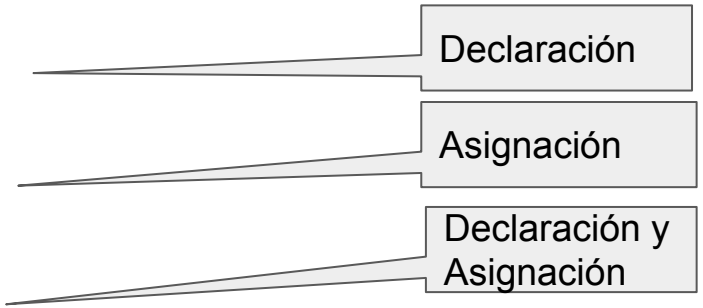
```
<ident> = <expresion>;

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

boolean estado;

estado = true;

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



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”);

    }

}
```

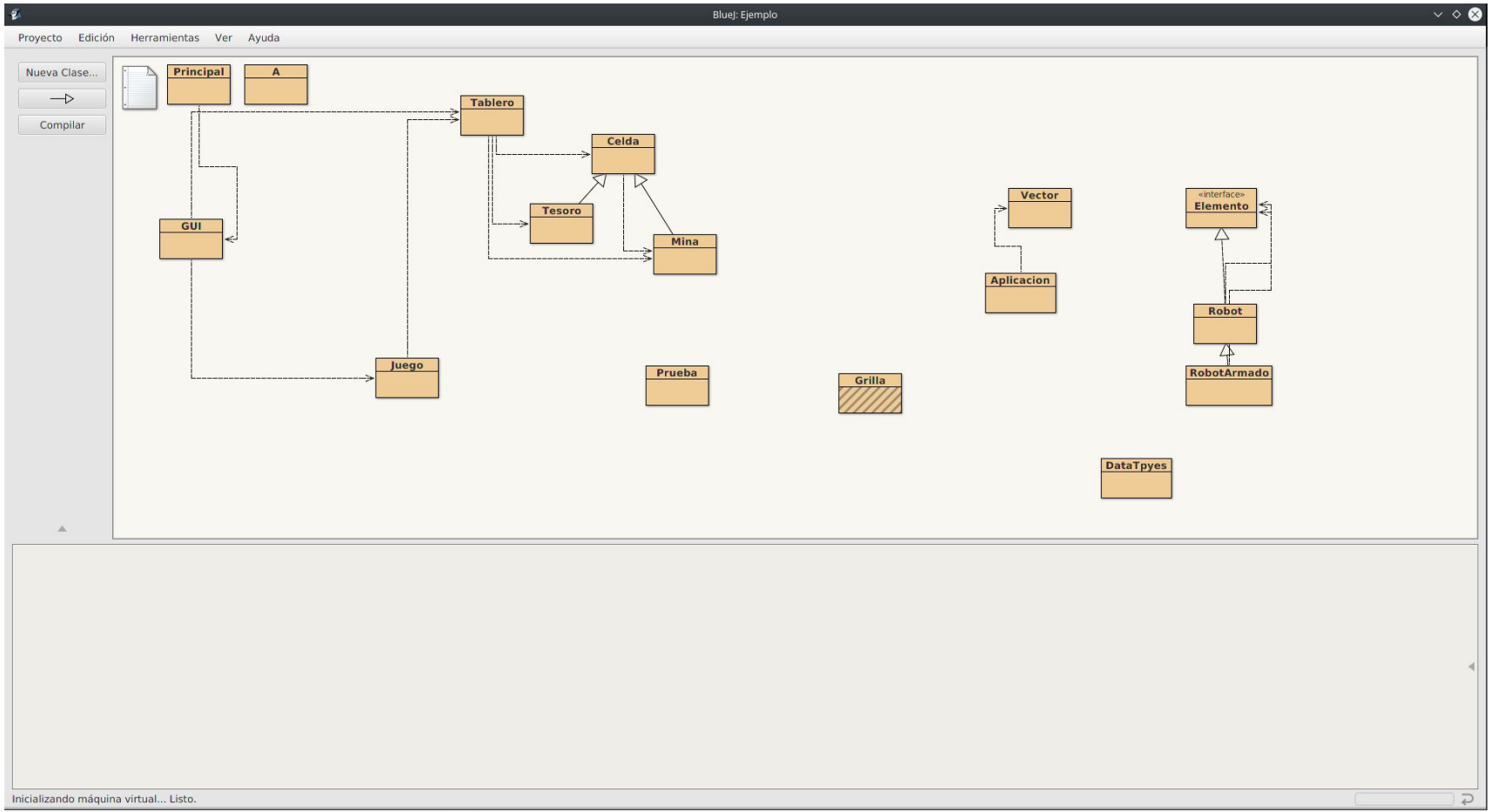
definición de una clase

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.

BlueJ


BlueJ: Ejemplo
Proyecto
Edición
Herramientas
Ver
Ayuda
Nueva Clase...
Compilar
Principal
A
Tablero
Celda
Tesoro
Mina
GUI
Juego
Prueba
Grilla
Vector
Aplicacion
«interface»
Elemento
Robot
RobotArmado
DataTpyes
Inicializando máquina virtual... Listo.

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{

    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;
    }
    ...
}
```

Clases como unidad de programación

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
            es un dígito del número n */
            boolean esta = false;
            while (n>0) && !esta {
                if (d == n % 10)
                    esta = true;
                n = n/10;
            }
            return esta;
        }
}
```

Tipo elemental **boolean**

Condicional **if**

Operadores relacionales y lógicos.

Comentario de múltiples líneas.

¿Preguntas?