Sesión 2:
Tipos de datos y variables

Índice

Documentación

- Especificación del lenguaje:

  https://docs.oracle.com/javase/specs/index.html

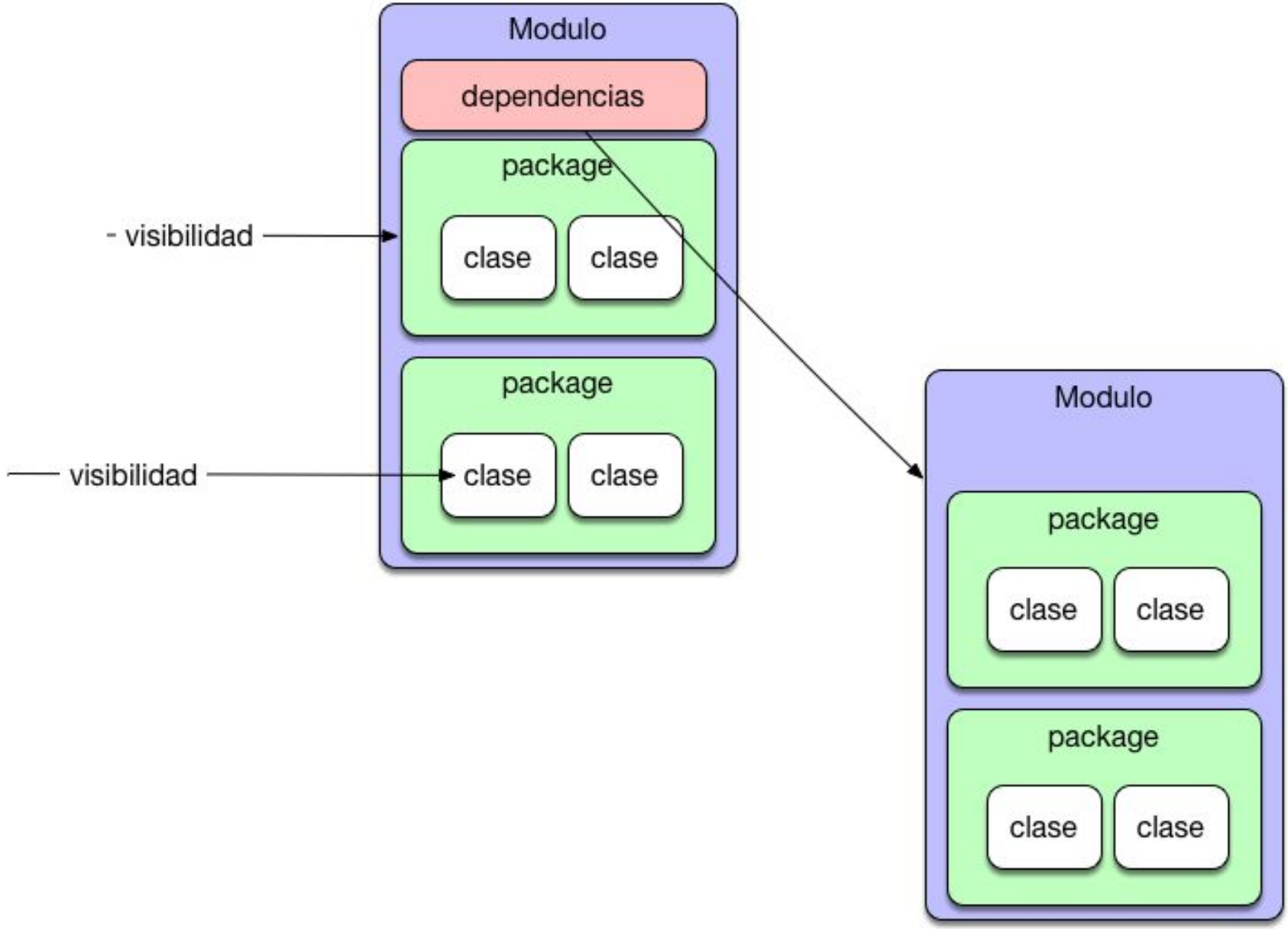
  Version 21:

  https://docs.oracle.com/en/java/javase/21/index.html

- Versión 21 en HTML:

  https://docs.oracle.com/javase/specs/jls/se21/html/index.html

Paquetes, módulos y clases



Documentación de la API

Es una biblioteca de paquetes que vienen con el JDK.

- Especificación de la versión 21 de la API:

  https://docs.oracle.com/en/java/javase/21/docs/api/index.html

- Módulo java.base:

  https://docs.oracle.com/en/java/javase/21/docs/api/java.base/module-summary.html

Léxico de Java

Identificadores

Se usan para los nombres de las clases, los métodos y las variables.

- Secuencia descriptiva de caracteres Unicode: letras mayúsculas o minúsculas, números, barra baja o símbolo de dólar ($).
- No debe empezar nunca con un número
- No puede coincidir con ninguna de las palabras reservadas del lenguaje.
- Java distingue mayúsculas y minúsculas -> VALOR != valor.

Ejemplos

- Correctos: TempMedia cuenta a4 $prueba esto_es_correcto
- Incorrectos: 2cuenta Temp-alta No/Correcto

Léxico de Java

Comentarios

Existen 3 tipos de comentarios.

- // -> Para comentar una sóla línea
  Ejemplo:

```
int variable = 100 //Esto es un comentario
```

- /* … */ -> Para comentar múltiples líneas
  Ejemplo:

```
/*
Demostración de la sentencia if
*/
```

Etiquetas standard de JAVADOC

TAG	DESCRIPCIÓN	COMPRENDE
@author	Nombre del desarrollador.	Nombre autor o autores
@version	Versión del método o clase.	Versión
@return	Informa de lo que devuelve el método, no se aplica en constructores o métodos "void".	Descripción del valor de retorno
@param	Definición de un parámetro de un método, es requerido para todos los parámetros del método.	Nombre de parámetro y descripción
@throws	Indica la excepción que puede generar	
@see	Hace referencia a otro método o clase.	Referencia cruzada referencia (#método(); clase#método(); paquete.clase; paquete.clase#método()).
@deprecated	Indica que el método o clase es obsoleto (propio de versiones anteriores) y que no se recomienda su uso.	Descripción

Léxico de Java

Comentarios

- /** … */ -> Comentarios de documentación, permiten introducir información sobre el programa dentro del propio programa. *(metadatos)*

Ejemplo:

```
/**
Esta clase dibuja un gráfico de barras
* @author Alfredo
  @version 3.2
*/
```

NOTA: Los comentarios no se anidan.

https://docs.oracle.com/en/java/javase/21/javadoc/javadoc.html

Léxico de Java

Separadores

En Java hay 12 tokens separadores. Estos son los más usados:

- ( )Paréntesis: Contener parámetros al definir o llamar a un método.
  También para contener expresiones, operaciones…
- { }Llaves: Para arrays inicializados automáticamente
  También para definir bloques de código, clases y métodos
- [ ] Corchetes: Para declarar tipos de arrays.
  También para hacer referencia a valores de array
- ; Punto y coma: Separador de sentencias o lineas
- , Coma: Separa nombre al declarar variables consecutivas.
  También para encadenar sentencias en un bucle for
- . Punto: Para separar nombres de paquetes de subpaquetes y clases
  También para hacer referencia a una variable de método.

Palabras clave de Java

Existen 51 palabras clave reservadas en el lenguaje Java. No se pueden utilizar como identificadores.

abstract continue for new switch assert default if

package synchronized boolean do goto private this break

double implements protected throw byte else import public

throws case enum instanceof return transient catch extends

int short try char final interface static void class

finally long strictfp volatile const float native super while

_ (underscore)

Además, Java reserva las siguientes: **true**, **false** y **null**.

Tipos de datos

- Java es un lenguaje fuertemente tipado. Parte de su robustez y seguridad se debe a ello:
    - Cada variable y expresión tienen un tipo de dato, definido de forma estricta.
    - En todas las asignaciones, ya sean explícitas o al pasar parámetros a un método se comprueba la compatibilidad de los datos.
- El compilador comprueba que todos los tipos son compatibles.
- Cualquier incompatibilidad da lugar a errores.
- Se manejan como variables, constantes y literales.

Los tipos simples o primitivos

Java define 8 tipos simples de datos clasificados en 4 grupos:

- Enteros: números enteros con signo (negativos y positivos)

Nombre	Anchura (bits)	Rango
long	64	-2^{63} a $2^{63}-1$
int	32	-2^{31} a $2^{31}-1$ (-2.147.483.648 a 2.147.483.647)
short	16	-2^{15} a $2^{15}-1$ (-32.768 a 32.767)
byte	8	-2^{7} a $2^{7}-1$ (-128 a 127)

Los tipos simples o primitivos

- Coma flotante: también conocidos como números reales. Expresiones que requieren precisión decimal.

Nombre	Anchura (bits)	Rango
double	64	$1.7x10^{-308}$ a $1.7x10^{308}$ (16 lugares posición)
float	32	$1.4x10^{-45}$ a $3.4x10^{38}$ (8 lugares posición)

Los tipos simples o primitivos

- Caracteres

  char es el tipo de de dato para almacenar caracteres. Ocupa 16 bits.

  Java utiliza Unicode (conjunto internacional de caracteres) para representar caracteres. Los 127 primeros caracteres de Unicode se corresponde con el código ASCII.

  Podremos utilizar números para representar caracteres en ASCII o Unicode.

- Booleanos

  boolean permite almacenar valores lógicos en Java.

  Sólo puede tomar dos valores: **true** o **false**.

Los tipos simples o primitivos

Literales

Los literales son valores constantes que no están definidos en una variable, pero Java nos deja utilizarlos entre el código.

- Enteros: 100 -- Java lo crea como tipo int
- Largos: 8456l, 33456L
- Coma flotante: 98.5 -- Por defecto, se crea un double. Podemos forzar el tipo:
  - 98.5f será un float
  - 98.5d será un double.
- Booleanos: true/false
- Caracteres: ‘X’
- Cadena de caracteres: “Esto es una prueba”

Los tipos simples o primitivos

Secuencia de escape

Secuencia de escape	Descripción
\ddd	Carácter octal (ddd)
\uxxxx	Carácter UNICODE hexadecimal (xxxx)
\’	Comilla simple
\”	Comilla doble
\\	Barra invertida
\r	Retorno de carro
\n	Nueva línea o salto de línea
\f	Comienzo de páginas
\t	Tabulador
\b	Retroceso

char ch = ‘\141’; es ‘a’ en ASCII - char ch1 = ‘\u0061’; es a en ASCII

char ch2 = ‘\ua432’; es el caracter japonés □

Cadenas de caracteres

La clase String

String no es un tipo de datos simple. Nos va a permitir guardar en un objeto cadenas de caracteres.

```
System.out.println("Esto es una cadena de caracteres");

String miCadena = "Esto es una cadena de caracteres";

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

Estudiaremos la clase con más detalle más adelante.

Variables

- La variable es la unidad básica de almacenamiento en un programa Java.

- Se define mediante un tipo, un identificador, y un inicializador opcional.

tipo identificador [= valor_inicial]

```
int d = 5;              //declara 1 entero y lo inicializa

int a, b, c;            //declara 3 enteros a, b y c

int e = 3, f = 4, g; //declara 3 enteros, e inicializa e y f

byte z = 22;            //declara z de tipo byte y la inicializa

char x = 'x'            // la variable x tiene el valor 'x'
```

Variables

Inicialización dinámica

Java permite la inicialización dinámica de una variable mediante cualquier expresión válida en el instante en el que se declara la variable.

```
//Estas variables se inicializan al principio
double a = 3.0, b = 4.0
//c y d se están inicializando dinámicamente
double c = a * b;
double d = Math.sqrt(a*a + b*b);
```

Las variables se pueden declarar en cualquier punto, pero sólo son válidas después de ser declaradas.

```
count = 100 //error, no se puede usar antes de declararla
int count;
```

Constantes

- Una variable se puede declarar como constante precediendo su declaración con la etiqueta final:

```
final int NUM_BARCOS = 10;
```

- Se define mediante un tipo, un identificador, y un inicializador opcional.

```
final double PI = 3.14159    // declara una aproximación de PI
final float EURO = 166.386
```

- La inicialización de una variable *final* se puede hacer en cualquier momento posterior a su declaración. Cualquier intento de cambiar el valor de una variable declarada como *final* después de su inicialización produce un error en tiempo de compilación.

- Por convención irán en mayúsculas. Si son compuestos, las palabras se separan con subrayados. **final** int CTE_GRAVITACIÓN;

Operadores aritméticos

Operador	***Significado***	***Ejemplo***
+	Suma	4 + 2
-	Resta	5 - 5
*	Multiplicación	43 * 1
/	División	5 / 2
%	Módulo (resto división)	resto = 5 % 3
++	Incremento	a = 1 a++ // a vale 2

Operadores aritméticos

Operador	***Significado***	***Ejemplo***
--	Decremento	a = 100 a-- // a vale 99
+=	Suma y asignación	a = 4 a += 2 // a aquí vale 6
-=	Resta y asignación	b = 5 b -=2 // b aquí vale 3
*=	Multiplicación y asignación	c = 2 c *= 2 // c vale 4
/=	División y asignación	d = 6 d /= 2 // d vale 3
%=	Módulo y asignación	e = 6 e %= 2 // e vale 0

Operadores aritméticos

- Los operandos de los operadores deben ser tipo numérico. No se pueden utilizar con el tipo **boolean**. Sí se puede usar sobre **char**, que ya sabemos que se le pueden asignar números.

- El símbolo **menos (-)** también puede negar variables. Ejemplo:

```
a = 5;  b = -a;  //b vale -5;
```

- Los **operadores con asignación** permiten ahorrar escritura y es implementado mejor por el intérprete Java. Por eso, se suelen emplear en programas profesionales.

- Los operadores de **incremento** y **decremento**, aumentan o disminuyen en una unidad su operando.

```
x++;  //es lo mismo que escribir x = x + 1;
```

- Los operadores de incremento o decremento pueden aparecer como prefijo del operando

```
x = 40;
y = ++x;
// y valdrá 41. x también vale 41
```

```
x = 40;
y = x++;
// y valdrá 40 y x será 41
```

Operadores relacionales

Determinan la relación de un operando con otro. El resultado es un valor **boolean**

Operador	***Significado***	***Ejemplo***
==	Igual a	4 == 3 //false
!=	Distinto de	4 != 3 //true
>	Mayor que	4 > 2 //true
<	Menor que	5 < 8 //true
<=	Menor o igual que	5 <= 5 //true
>=	Mayor o igual que	6 >= 0 //true

Operadores lógicos

Sólo operan sobre operandos de tipo **boolean**.. El resultado es un valor **boolean**

Operador	***Significado***	***Ejemplo***
&	AND lógico	A & B // **true** cuando A y B son verdaderos. Evalúa ambos operandos.
|	OR lógico	A | B // **true** cuando A o B son verdaderos. Evalúa ambos operandos.
^	XOR lógico	A ^ B // **true** cuando A y B son distintos.
||	OR en cortocircuito	A || B // **true** cuando A o B son verdaderos. Evalúa condicionalmente
&&	AND en cortocircuito	A && B // **true** cuando A y B son verdaderos. Evalúa condicionalmente
!	NOT lógico	!A // **true** si A es falso.

Operadores

Operadores lógicos

Operador	*Significado*
==	Igual a
!=	Distinto de
?:	if-else

A	B	A | B	A & B	A ^ B	!A
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE

Operadores

Operadores lógicos en cortocircuito

- La diferencia de usar los operadores lógicos & y | con utilizar los operadores lógicos en cortocircuito && y ||, es que con los primeros se evalúan ambas condiciones. Sin embargo, con los operadores en cortocircuito, Java no evaluará el operando de la derecha si al evaluar el de la izquierda obtiene ya el resultado de la operación.
- Ésto es útil cuando el operando de la derecha depende del de la izquierda.

- Por ejemplo, si tenemos:

```
int x = 0; int y = 2;
boolean p = ( x != 0 ) & ( ( y / x ) != 0 );
```

- Esto dará un error, porque si x es 0, evalúa también el de la derecha, al ser 0 y/x, da error en la operación. Utilizando && , al no cumplirse el primero, ya no se evalúa el segundo.

- Lo habitual y recomendable es utilizar las formas en cortocircuito de AND y OR, es decir && y ||.

Java: Segundo programa de ejemplo

```
class Ejemplo2 {
    public static void main(String args[]) {
        int num;     //Declara una variable llamada num

        num = 100;      //Asigna a num el valor 100
        System.out.println("Esto es un num: " + num);

        num = num * 2;
        System.out.print("El valor de num * 2 es: ");
        System.out.println(num);
    }
}
```

Variables

Tiempo de vida de las variables

Hasta ahora hemos declarado todas las variables dentro del bloque main. Las llamamos ***variables globales***.

Java permite la declaración de variables dentro de un bloque (código entre llaves {}). Se llaman ***variables locales*.**

```
public static void main (String args[]) {
    int x = 10;          //variable conocida en todo el bloque main
    if (x == 10) {
        int y = 20;  //variable sólo conocida dentro del bloque if
        System.out.print("x vale " + x + ". y vale " + y);
    }
    // y = 100           //Error, aquí no se conoce y
    System.out.print("x vale " + x); //pero si se conoce x
 }
```

Variables

Conversión de tipos

Conversiones compatibles

- Si dos tipos de variables son compatibles, se puede asignar un valor de un tipo a una variable de otro tipo.
- Por ejemplo, es posible asignar una variables de tipo **int** a una variable **long**
- **Java** realiza la conversión automáticamente

Para ello, se deben de cumplir dos condiciones:

- Los dos tipos son compatibles
- El tipo de destino es más amplio que el tipo fuente.

- Por ejemplo, el tipo **long** es lo suficientemente amplio para almacenar valores de tipo **int** o tipo **byte.**
- Los tipos enteros y de coma flotante son compatibles entre ellos
- Los tipo **char** o **boolean** no son compatibles entre sí, ni con los tipos enteros o de coma flotante.

Variables

Conversión de tipos

Conversiones incompatibles

- ¿Qué pasa con los tipos que no son compatibles? Por ejemplo, transformar de **double** a **byte**.
- Para crear una conversión entre dos tipos incompatibles, se debe usar un ***casting***.
- Es una conversión de tipos explícita y tiene la siguiente forma:

  (tipo) valor

  donde tipo indica el tipo al que se desea convertir el valor especificado.

- También permite realizar conversiones a tipos más pequeños. Por ejemplo, de un **int** a un **byte**. Esto se denomina *estrechamiento*.
-

```
int a = 20;
byte b;
b = (byte) a;
```

Variables

Conversión de tipos

Promoción automática de tipos a expresiones

- Las conversiones de tipo también pueden tener lugar en expresiones. Si tenemos:

```
byte a = 40, b = 50, c = 100;
int d = a * b / c;
```

- El resultado de **a * b** excede del rango del tipo **byte**. Por tanto, Java lo transforma a entero de forma automática.

```
byte b = 50;
b = b * 2;  //Error, lo correcto sería hacer un cast
b = (byte) (b * 2)
```

- **50 * 2** se puede almacenar perfectamente en un **byte**. Sin embargo, al evaluar la expresión los operandos son promocionados automáticamente al tipo **int**. Por tanto, hay que hacer un **cast**

Variables

Reglas en la conversión de tipos en las expresiones

- Los valores **byte** y **short** son promocionados a **int**.
- Además, si uno de los operandos es del tipo **long**, la expresión completa es promocionada a **long**
- Si un operando es del tipo **float**, la expresión completa es promocionada al tipo **float**.
- Si cualquiera de los operandos es **double**, el resultado será un **double**

```
byte b = 42;
char c = ‘a’;
    short s = 1024;
    int i = 50000;
    float f = 5.66f;
    double d = .1234;
    double total = (f * b) + (i / c) - (d * s)
```

f * b será un ___, i / c será ____, d * s será un ____, float + int es ____ y el total es ____

Java: Tercer programa de ejemplo

```
class Ejemplo3 {
    public static void main(String[] args) {
        int x = 2;
        int y = 7;
        double division;

        division = (double) y / (double) x;

        // si no hacemos casting vemos que el resultado cambia
        // division = y / x;

        System.out.println("El resultado de la división es: " + division);
    }
}
```