FUNDACIÓN ACCENTURE accenture

INTRODUCCIÓN A JAVA

2. CONCEPTOS BÁSICOS DE JAVA

Adéntrate en los elementos más básicos de un programa en Java, conoce las nomenclaturas que incluye y los tipos de datos con los que trabaja.

TABLA DE CONTENIDOS

INTRODUCCIÓN A JAVA

2. CONCEPTOS BÁSICOS DE JAVA

2.1. ESTRUCTURA DE UNA CLASE JAVA

Todo programa Java se llama **clase**, tanto los pequeños programas que diseñamos en nuestros comienzos como programadores, como los grandes y complejos sistemas transaccionales que tiene un banco.

En este tema vamos a ver cuáles son los **componentes** que debe tener una clase Java para que esté correctamente diseñada. Todos estos componentes definirán la estructura de la clase.

COMPONENTES DE UNA CLASE/PROGRAMA JAVA:

PAQUETE (*PACKAGE*)

Los paquetes son una forma de organizar las clases. Indica la carpeta o directorio **dónde está guardada la clase**. Todas las clases que guardan cierta utilidad semejante se agrupan en un paquete, un paquete es la primera línea de código que debe estar en una clase Java. Sólo puede existir un paquete en una clase y cuando no le da un nombre específico, el compilador crea uno por defecto llamado default package.

Por ejemplo, imaginemos un curso de cocina, hoy vamos a aprender a hacer un bizcocho por tanto el paquete será la carpeta Repostería.

CLASE (*CLASS*)

El siguiente paso sería indicar el **nombre de la clase**.

Siguiendo nuestro ejemplo, la clase tomaría el nombre del tipo de receta a elaborar, por ejemplo, Bizcocho.

SENTENCIAS

Indica los **atributos o características de la clase**. Al final de cada sentencia se finalizará con punto y coma (;).

En nuestro caso, las sentencias serían los ingredientes: harina, yogurt, huevos, mantequilla...

BLOQUES

Los bloques son un **conjunto de sentencias**, que están delimitados por llaves ({}).

(Siguiendo con nuestra receta, podemos tener el bloque “lácteos”, que incluye leche y mantequilla)

MÉTODO

Una clase también implementa un comportamiento. El comportamiento de una clase se define con la creación de métodos, un método debe contener el **tipo de retorno + nombre del método**. Se recomienda que este siempre sea en verbo.

Para nuestro ejemplo, el método serán las instrucciones de elaboración: mezclar, batir, hornear...

Atributos: características que tendrán los objetos que crearemos a partir de esa clase.
Retorno: es el valor que queremos que nos devuelva el método al aplicarlo.

COMENTARIOS

Java permite incluir comentarios entre las líneas de código, para dejar más claras aún las instrucciones. Existen tres tipos de comentarios en Java:

Comentarios de una sola línea
// Esto es un comentario

Comentarios multilínea
/* Esto es un comentario de una o más líneas */

Comentarios de documentación para la herramienta Javadoc
/ Esto es un comentario para javadoc */**

Para nuestra receta podríamos además anotar alguna explicación más detallada del proceso antes de algún método o antes de las sentencias por si nuestra receta la coge un principiante en el mundo de la repostería, ejemplo: "importante usar yogur natural" o "preparamos el horno a 180º 5 minutos antes de introducirlo".

Vemos un ejemplo real de un programa Java, en este caso se trata de la clase vehículo:

```
package concesionario;

//A continuación la clase y sus aributos
class Vehiculo{
  String color;
  float motor;

  public void arrancar(){
  }
  public int frenar(){
   return 1;
          }
        }
```

Paquete

Comentario

Clase

Bloque de Sentencias

Método

Método

2.2. DENOMINACIONES: IDENTIFICADORES Y PALABRAS CLAVE

¿QUÉ ES UN IDENTIFICADOR?

Los identificadores son los **nombres** únicos que el programador da a los **componentes** que se están manejando en un programa, es decir, el nombre que ponemos por ejemplo a las clases, métodos y variables.

¿QUÉ PRINCIPIOS DEBE SEGUIR EL PROGRAMADOR A LA HORA DE ESTABLECER IDENTIFICADORES?

- Deben usarse nombres que sean **significativos**. Siguiendo el ejemplo del bizcocho, también podríamos haber llamado a la clase RecetaBizcocho, pero siempre algo que haga referencia al programa.
- Se recomienda **empezar por una letra**, aunque después se pueden emplear números.
- **No** se permite el uso de **caracteres especiales** como: espacio en blanco, +, %,*, etc.
- Los nombres de **clases comienzan con mayúsculas** y cuando son nombres compuestos cada palabra comienza con mayúsculas. Ej. public class RecetaBizcocho.
- Java distingue entre **mayúsculas y minúsculas**. Ej. bizcocho es distinto que Bizcocho.
- Los nombres de **constantes** se escriben **todo con mayúsculas** y si el nombre es compuesto se usa el carácter de **subrayado** para separar las palabras. Ej. TIEMPO_HORNEADO), pero para el nombre de la clase, métodos y atributos, no se usa el carácter de subrayado.
- **No** pueden coincidir con **palabras reservadas/clave** (las vemos en el siguiente punto).

¿QUÉ ES UNA PALABRA CLAVE?

También conocidas como **palabras reservadas**, son aquella que están definidas de manera exclusiva por el lenguaje de programación para un **propósito específico**. No se pueden utilizar estas palabras para darles otro comportamiento distinto al definido.

TIPO DE DATOS

boolean
dato verdadero/falso.

byte
número entero de 8 bits.

char
dato valor unicode de 16 bits.

double
número real en coma flotante 64 bits.

float
número real en coma flotante 32 bits.

int
número entero de 32 bits.

long
número entero de 64 bits.

short
número entero de 16 bits.

ESTRUCTURAS DE CONTROL

CONDICIONALES

if
para instrucciones de control condicionales simples (if) o múltiples (else if).

else
instrucción de control condicional doble que indica qué hacer en caso de que no se cumpla la condición del (if) o el (else if).

switch
instrucción de control condicional múltiple.

case
para indicar cada caso de una instrucción de control (switch).

default
indica el caso por defecto de una instrucción de control (switch).

BUCLES

for
para escribir bucles que se repiten un número determinado de veces.

while
para escribir bucles que se repiten mientras una condición sea cierta. La condición se valida al principio.

do
para escribir bucles (do while) que se repiten mientras una condición sea cierta. La condición se valida al final.

SALTO/RETORNO

break
interrumpe la ejecución de un bucle o de una instrucción.

continue
interrumpe la ejecución de un bucle, pero permite seguir realizando interacciones.

return
interrumpe la ejecución del método y puede devolver un valor de retorno.

EXCEPCIONES

try
indica un bloque de código donde se quieren atrapar excepciones.

catch
cláusula de un bloque (try) donde se especifica una excepción.

finally
permite especificar un bloque de código que siempre se ejecutará, se produzca o no una excepción.

throw
permite lanzar una excepción.

throws
indica las excepciones que un método puede lanzar.

MODIFICADORES

abstract
para definir clases y métodos abstractos.

final
indica que una variable no se puede modificar, un método no se puede redefinir o una clase no se puede heredar.

native
indica que un método está en un lenguaje de programación dependiente de la plataforma.

private
indica que un elemento es accesible únicamente desde la clase donde se ha definido.

protected
indica que un elemento es accesible desde la clase donde se ha definido, subclases de ella y otras clases del mismo paquete.

public
indica que un elemento es accesible desde cualquier clase.

static
indica que un elemento es único en una clase y que se puede acceder al él sin tener que crear una instancia u objeto de la clase.

synchronized
indica que un método o bloque de código es atómico.

transient
para especificar que un atributo no sea persistente.

volatile
indica que el valor de un atributo que está siendo utilizado por varios hilos esté sincronizado.

ESTRUCTURA DE DATOS

class
para definir una clase.

extends
permite indicar la clase padre de una clase.

implements
para definir la/s interfaces de una clase.

import
importa una clase que se encuentra en otro paquete o en la **API de Java**.

interface
para declarar una interfaz.

package
te permite agrupar un conjunto de clases e interfaces.

OTROS

assert
para afirmar que una condición es cierta.

const
se usaba para la definición de constantes, pero ya no se utiliza, ahora se usa (final).

enum
para definir tipos de datos enumerados.

goto
se usaba para moverse de un punto a otro del código, pero ya no se utiliza.

instanceof
permite saber si un objeto es una instancia de una clase concreta.

new
para crear un objeto nuevo de una clase.

override
sirve para sobreescribir un método.

strictfp
indica que se tienen que utilizar cálculos en coma flotante estricto.

super
permite invocar a un método o constructor de la superclase.

void
dato vacío (sin valor).

this
para referenciar al objeto e invocar al constructor.

API de Java: es una interfaz de programación de aplicaciones (API, por sus siglas del inglés: Application Programming Interface) provista por los creadores del lenguaje de programación Java, que da a los programadores los medios para desarrollar aplicaciones Java.

2.3. TIPOS DE DATOS

En el tema anterior (palabras clave) vimos la columna en la que identificábamos la denominación de algunos tipos de datos. En este caso vamos a entrar más a fondo en ellos.

Java trabaja con **dos tipos de datos**: primitivos/simples o de referencia/complejos, veamos sus diferencias.

1. DATOS PRIMITIVOS O SIMPLES

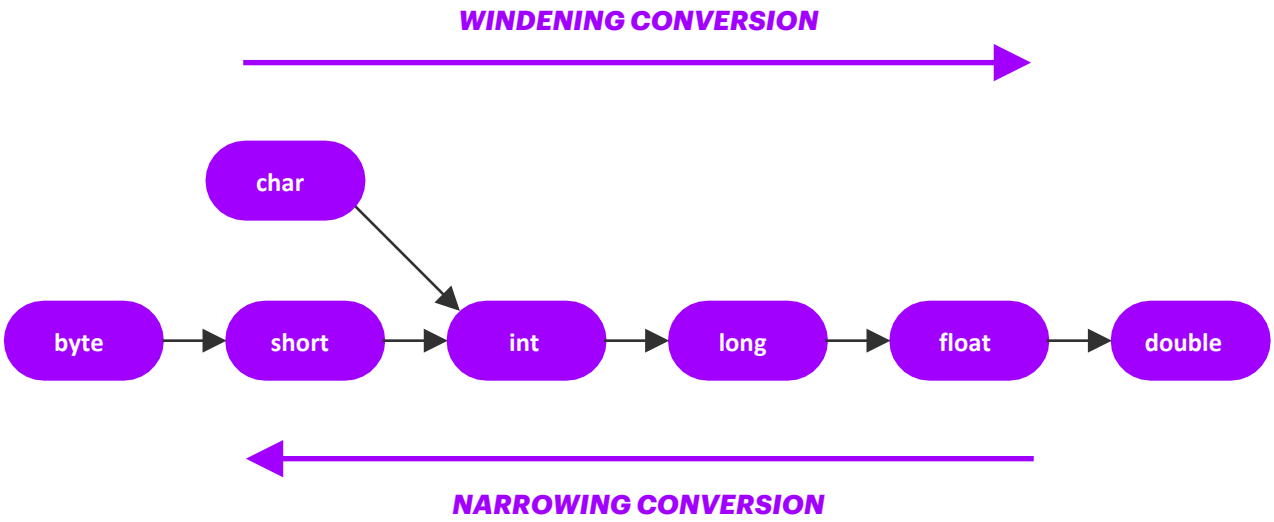
Son **entidades elementales** como números, caracteres, Verdadero/Falso. **No son objetos**, por tanto, no necesitan ser creados para usarse.

TIPO	NOMBRE	TAMAÑO	RANGO	VALOR DEFECTO
ENTERO	**byte**	8 bits	-128 a 127	0
	short	16 bits	-2^{15} a 2^{15} -1	0
	int	32 bits	-2^{31} a 2^{31} -1	0
	long	64 bits	-2^{63} a 2^{63} -1	0
DECIMAL	**float**	32 bits	± 3.40282347e^{+38}	0
			± 1.40239486e^{-45}	0
	double	64 bits	± 1.79769313486231570e^{+306}	0
			± 4.94065645841246544e^{-324}	0
CARÁCTER	**char**	16 bits	\u0000 a \uffff	'\u0000`(null)
LÓGICO	**boolean**	true / false	N.A.	False

Veamos varios ejemplos de los más usados en Java:

- **Int**: números de tipo entero (años, días de la semana)
- **Float**: números decimales de hasta 7 dígitos (altura, peso)
- **Double**: números decimales muy precisos, hasta 16 dígitos (área, longitud)
- **Char**: conjunto de caracteres (códigos con letras y números)
- **Boolean**: datos que sólo pueden ser verdadero o falso (par, primero)

Para este tipo de números, Java tiene una **función Casting**, que transforma un dato primitivo de un tipo a otro en función del tamaño que ocupa. La conversión puede ser de dos formas:



- **Casting implícito:** Lo realiza el compilador automáticamente, no tenemos que añadir ninguna línea de código. Una conversión implícita (widening conversion) se da cuando el contenedor es más grande que el valor a almacenar.

- **Casting explícito:** Lo realiza el programador y se tiene que declarar en qué tipo de dato se quiere transformar. En una conversión explicita (narrowing casting) se da cuando el contenedor donde se quiere guardar es más pequeño que el valor a almacenar. En este caso, puede que haya pérdida de datos.

2. DATOS REFERENCIA O COMPLEJOS

Son tipos de datos más complejos, que pueden estar formados por varias variables, otros datos complejos y/o comportamientos (funciones).

Son objetos, por tanto, necesitan ser creados por el programador en caso de usarse

GRUPO	DESCRIPCIÓN	VALOR DEFECTO
BIBLIOTECA ESTÁNDAR JAVA	Son las que nos proporcionan la librería estándar de JAVA. (String, Date, …)	null
PERSONALIZADOS/ DEFINIDOS POR EL DESARROLLADOR	cualquier clase creada por nosotros. clases customizadas/ personalizadas.	
ARRAYS	conjunto de elementos (arreglos). objeto que carece de métodos.	
LÓGICO	Byte Short Integer Long Float Double Character Boolean	

De ellos, el más utilizado es el tipo de dato **String** que permite manejar textos y cadenas de texto. Veamos varios ejemplos:

→ El operador "+" permite concatenar cadenas.

```
String str1 = "Hola";
String str2 = "Fernando";
String str3 = str1 + " " + str2; //str3 sería "Hola Fernando"
```

→ String ofrece un conjunto de métodos (formas de usarse)

Para obtener la **longitud** de la cadena (número de Strings)

```
str1.lenght() //devuelve el número 3 ya que hay 3 cadenas (str1,str2,str3)
```

Para obtener el **carácter** que está en la posición denotada por(int)

```
str1.charAt(0) //devuelve H, de la palabra Hola ya que tiene la posición 0.
```

2.4. VARIABLES Y CONSTANTES

VARIABLES Y CONSTANTES, ¿QUÉ SON?

Como vimos en el curso de PSEINT, las variables y constantes son espacios reservados para almacenar información y que, durante la ejecución del programa podrán ir cambiando (Variables) o permanecer fijas (Constantes):

VARIABLE

Objeto cuyo contenido **puede variar durante el proceso de ejecución** del algoritmo. Por ejemplo, datos para llevar a cabo una suma.

CONSTANTE

Objeto que **no experimenta cambios** durante todo el desarrollo del algoritmo

DECLARACIÓN DE UNA VARIABLE EN JAVA

Declarar es identificar con un **tipo y nombre** a la variable, y es necesario declararlas antes de usarlas. El tipo indica qué tipo de valores puede guardar. El nombre permite acceder a su contenido y es necesario escribirlo en minúsculas

```
<tipo dato> <nombre variable>;
```

Ejemplo: la edad, que será un dato que puede variar durante el programa

```
int edad;
```

INICIALIZAR VARIABLES

Se produce cuando a una variable se le da un valor por primera vez, antes de ser utilizada. A nivel de sintaxis la inicialización o asignación, se realiza usando el operador de asignación "=". Si no se inicia previamente, Java lo hace automáticamente.

Ejemplo: para los valores numéricos Java les da un valor inicial de cero.

```
int edad = 0;
```

ASIGNAR VARIABLES

Se produce cuando a una variable previamente inicializada se le asigna un nuevo valor durante la ejecución. Dependiendo del tipo de variables se asignan de una manera u otra.

- **Numéricos:** se asignan directamente a la variable. Ej. `int edad = 20;`
- **Caracteres:** los caracteres se ha de usar la comilla simple. Ej. `char género = 'M';`
- **Cadenas:** para las cadenas de textos se usa las comillas dobles. Ej. `String nombre = "Roberto";`
- **Booleanos:** se ha de poner true o false. Ej. `boolean casado = true;`

ÁMBITO DE UNA VARIABLE

El ámbito de una variable define su alcance de uso, o lo que es lo mismo, en que secciones de código una variable estará disponible. Fuera de este ámbito, una variable no podrá ser accedida (no existe). Además, las variables se han de definir en el ámbito más reducido posible. Hay tres tipos de ámbitos:

- **Local:** sólo están visibles dentro del bloque de código donde se han declarado (dentro de un método, un bucle, un condicional...), cuando éste finaliza, deja de existir.
- **De Instancia o Global:** son aquellas que pertenecen a cada instancia concreta de la clase donde han sido declaradas y sólo pueden ser accesibles desde la propia instancia a la que pertenecen.
- **De Clase o Estática:** la variable pertenece a la clase, no a la instancia, y se puede acceder a ella desde cualquier parte de la clase donde han sido declaradas.

Ejemplo de declaración:

```
public class ClaseAmbitos {
        static int numero; //Variable estática de la clase

        String nombre; //Variable global en cada instancia de la clase

        void Calcular() {
              float precio; //Variable local a este método
              …
              if (precio >= 10.95) {
                    boolean verdad; //Variable local a este if
                    …
              }
        }
}
```

CÓMO FUNCIONA UN ARRAY

Una vez visto qué es una variable, podremos entender mejor cómo funciona un Array.

Un **Array** es un tipo de dato estructurado que permite almacenar un conjunto de datos homogéneo, es decir, todos ellos del mismo tipo y relacionados. Cada uno de los elementos que componen un array pueden ser de tipo simple como caracteres, entero o real, o de tipo compuesto o estructurado como son otros arrays u objetos. Una vez declarado el tamaño de un array, éste no puede ser modificado. **El índice inicial de los arrays en Java es 0.**

Array de tamaño 5

Declaración del array:

```
<tipo dato> [ ] <nombre_array>;
<tipo dato> <nombre_array> [ ];
<tipo dato> [ ] <nombre_array> [ ];
```

Ejemplos:

```
int [ ] arrayEnteros; //Declaración array de enteros
String myStringArray [ ]; //Declaración array de strings
char [ ] myCharArray [ ]; //Declaración array de char
```

Los arrays se pueden **inicializar** de dos maneras, por tamaño o con valores:

Por tamaño:

```
<tipo dato> [ ] <nombre_array> = new <tipo dato> [tamaño];
```

Ejemplo: `int[ ] arrayEnteros = new int[5];`

Con valores:

```
<tipo dato> [ ] <nombre_array> = {val1, val2, … };
```

Ejemplo: `String[ ] arrayNombres = {"Pedro", "Juan", "María", "Susana"};`

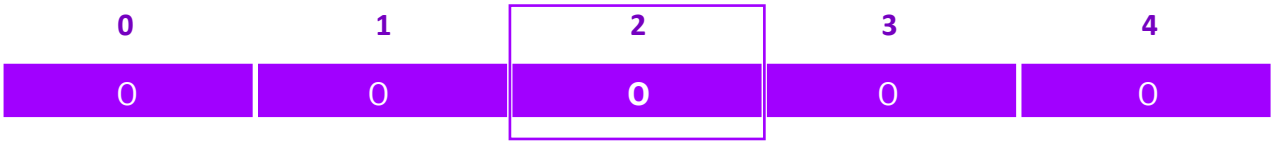
Para poder **recuperar o asignar** un valor a una posición del array, usaremos el índice:

Recuperar:

```
<tipo dato> <nombre_variable> = <nombre_array> [índice];
```

Ejemplo:

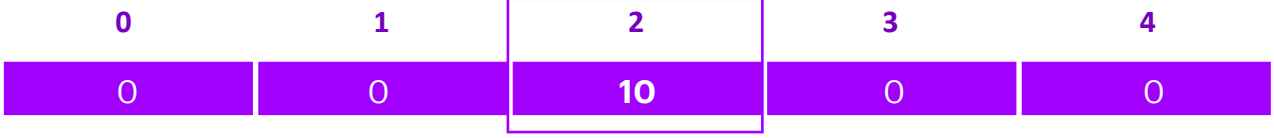
```
int numero = arrayEnteros [2];
```



Asignar

Ejemplo:

```
arrayEnteros [2] = 10;
```



Ejemplo:

```
arrayNombres [0] = "Sergio";
```

DECLARACIÓN DE CONSTANTES EN JAVA

- Se han de definir a nivel de **clase (global).**
- Para definirla se ha de hacer uso de dos palabras reservadas: **static y final.**
- Los nombres de las constantes se escriben en **mayúsculas.**

```
static final <tipo_dato> <NOMBRE_CONSTANTE> = valor;
```

Ejemplo: indicar el número de ruedas de un coche. Independientemente del modelo de coche, el número de ruedas siempre será cuatro.

```
static final int NUMERO_RUEDAS = 4;
```

2.5. PREGUNTAS FRECUENTES

- **¿Qué programa tengo que descargar para programar en Java?**

  El entorno de desarrollo que se suele utilizar para desarrollar Java es Eclipse ya que es un software de acceso libre (gratuito) y tiene en un único programa todas las herramientas para el desarrollo completo de una aplicación en sus diferentes etapas, desde la edición del código fuente inicial hasta la obtención del producto final.

- **¿Qué hay que tener en cuenta para nombrar una variable en Java?:**
  - Debe comenzar siempre con una letra (“edad” es válido, “3edad” no es válido)
  - Los nombres de las variables distinguen entre mayúsculas y minúsculas (una variable llamada “miCredito” y otra “MiCredito” son distintas)
  - Usa nombres descriptivos para nombrar tus variables (evita en la medida de lo posible usar abreviaturas o usar nombres poco claros)
  - El nombre de tu variable no debe coincidir con alguna palabra reservada.
  - Si tiene una sola palabra escríbela en minúscula y si tienes dos la segunda palabra empiézala por mayúscula “miCredito”.

FUNDACIÓN ACCENTURE

accenture