Una vez finalizadas todas las Unidades de Trabajo, se detallan los contenidos teóricos básicos a dominar.

JAVA

INSTALACIÓN

Para instalar JAVA es necesario seguir dos pasos:

1.- INSTALAR JAVA DEVELOPMENT KIT (JDK)

2.- CONFIGURAR VARIABLES DE ENTORNO DEL SISTEMA

INSTALAR JAVA DEVELOPMENT KIT (JDK)

Para instalar JAVA el primer paso será ir a la dirección:

https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Desde ahí se descargará el entorno Java Standar Edition Development Kit más actual seleccionando el sistema operativo que el usuario utilice.

Una vez terminada la descarga se procederá a su instalación mediante un doble click, tras la cual, se observarán en el directorio correspondiente dos carpetas: jdk y jre.

- La carpeta jdk alberga el compilador e intérprete de JAVA.
- La carpeta jre alberga la máquina virtual de JAVA.

CONFIGURAR VARIABLES DE ENTORNO DEL SISTEMA

Para poder empezar a usar JAVA se requiere una previa configuración que consiste en configurar dos variables de entorno del sistema.

Primera variable: JAVA_HOME

Esta variable de entorno del sistema informa al sistema operativo sobre la ruta donde se encuentra instalado JAVA. Hay que crearla:

- Se accede a: Equipo/Propiedades/Configuración avanzada/Variables de entorno
- Se pulsa en Nueva en la zona de Variables del sistema
- Se escribe lo siguiente:

  Nombre: JAVA_HOME
  Valor: La ruta de la carpeta jdk. Por ejemplo: C:\Programas\Java\jdk9

Segunda variable: PATH

Esta variable de entorno del sistema informa al sistema operativo sobre la ruta de distintos directorios esenciales para el funcionamiento del ordenador. Hay que editarla:

- Se accede a: Equipo/Propiedades/Configuración avanzada/Variables de entorno
- Se pulsa en Editar sobre la variable PATH en la zona de Variables del sistema
- Se pulsa en Nuevo y se escribe lo siguiente: %JAVA_HOME%\bin

PROBAR INSTALACIÓN DE JAVA

Para comprobar que se han seguido los dos pasos anteriores correctamente, es recomendable probar mediante consola a compilar y ejecutar un pequeño programa.

1. En un editor de texto se escribiría lo siguiente:

```
public class Ejemplo {   <- Es importante introducir aquí y en el nombre del archivo el mismo nombre
        public static void main (String[] arg) {
                System.out.println("Prueba");
}       }
```

2. El guardado debe realizarse utilizando el nombre de archivo: Ejemplo.java

3. En una consola, tres situarse en el directorio donde se encuentra el archivo, se escribiría lo siguiente:

Javac Ejemplo.java <- Se encarga de compilar el código fuente
Java Ejemplo <- Se encarga de interpretar el bytecode

JAVA vs OTROS LENGUAJES DE PROGRAMACIÓN

En un lenguaje de programación, el usuario desarrolla un programa escribiendo un código que solo él conoce, denominado código fuente, traduciéndose éste mediante la ***compilación*** a un código que solo la máquina entiende, denominado código màquina.

CÓDIGO FUENTE (.C .PHP ...) -> CÓDIGO MÁQUINA (.EXE)

En JAVA, el usuario desarrolla un programa escribiendo un código que solo él conoce, denominado código fuente, traduciéndose éste mediante la ***compilación*** a un código intermedio denominado bytecode, que mediante la ***interpretación*** se traduce a un código que solo la máquina entiende, denominado código màquina.

CÓDIGO FUENTE (.JAVA) -> BYTECODE (.CLASS) -> CÓDIGO MÁQUINA (.EXE)

ENTORNO DE TRABAJO

Programar en JAVA requiere un entorno de desarrollo. Hay varios a elegir: Eclipse, NetBeans, BlueJ, etc.

INICIO Y FIN DE UN PROGRAMA

```
public class NombreDelPrograma {
        public static void main (String[] arg) {
                ..............
}       }
```

COMENTARIOS

```
/*

.....

*/
```

FINALIZACIÓN DE LAS INSTRUCCIONES

Todas las instruccions finalizan con punto y coma.

CONSTANTES

Es aconsejable escribir las constantes en mayúsculas.

- Cuando cada objeto tiene su espacio de memoria con un contenido invariable:

  static final TipoDeLaConstante NombreConstante;

  Ejemplo: static final float PI = 3.1416f;

- Cuando cada clase tiene su espacio de memoria con un contenido invariable, ese epacio sería único y sería compartido por todos los objetos de la classe:

  final NombreConstante;

  Ejemplo: final double PI = 3.1416;

VARIABLES

Es aconsejable escribir las variables en minúsculas.

No llevan $ delante como en PHP.

Es necesario declararlas para poder usarlas, esto se lleva a cabo en las primeras líneas del programa y consiste en añadirles a la izquierda el tipo de datos del valor que albergan.

Ejemplo: int numero;

Es necesario inicializarlas para poder usarlas, esto se lleva a cabo en las primeras líneas del programa y consiste en darles el valor 0 en el caso de variables numéricas o el valor vacío (“”) en cadenas de texto.

Ejemplo: numero = 0;

TIPOS DE DATOS PRIMITIVOS

int – entero
long – entero muy grande
float – decimal
double – decimal muy grande
char – carácter
boolean – valor true o false

TIPOS DE DATOS OBJETO

Integer – entero
Long – entero muy grande
Float – decimal
Double – decimal muy grande
Character – carácter
Boolean – valor true o false
String – cadenas de texto

arrays – vector de elementos
tipos definidos por el programador

OPERADORES / OPERACIONES

Suma: +
Resta: -
Multiplicación: *
División: /
Resto: %
Igualdad: ==
Distinto: !=
Comparación: <, <=, >, >=
And: &&
Or: ||
Not: !
Asignación: =
Concatenación de cadenas de texto: +
Escribir: System.out.println("texto con salto de línea"); System.out.print("texto sin salto de línea");

ESTRUCTURAS DE CONTROL

CONDICIONALES

IF:

```
if (condición) {

.....

} else {

.....

}
```

SWITCH:

```
switch (variable) {

case valor1:
        .....
        break;
case valor2:
        .....
        break;
default:
        .....
        break;
}
```

REPETITIVAS

WHILE:

```
while (condición) {

.....

}
```

DO...WHILE:

do {

.....

} while (condición);

FOR:

for (i=valorinicial; i<valorfinal; i++){

.....

}

Incremento 1: i++ | Decremento 1: i-- | Incremento X: i+=X | Decremento X: i-=X

ENTRADA DE DATOS

Para introducir datos se necesita hacer uso de la clase que a continuación se detalla:

```
import java.util.Scanner;

public class EntradaDeTeclado {

        private String entradaTeclado;
        private int entradaTeclado2;

public EntradaDeTeclado() {

        entradaTeclado = "";
        entradaTeclado2 = 0;
}

public void pedirEntradaTexto() {

        Scanner entradaEscaner = new Scanner(System.in);
        entradaTeclado = entradaEscaner.nextLine();
}

public void pedirEntradaNumero() {

        Scanner entradaEscaner = new Scanner(System.in);
        entradaTeclado2 = entradaEscaner.nextInt();
}

public String getEntrada() {

        return entradaTeclado;
}

public int getEntrada2() {

        return entradaTeclado2;
}
}
```

Los pasos a seguir para utilitzar el código anterior serían los siguientes:

1. Crear una nueva clase que se llame EntradaDeTeclado y escribir el código expuesto.
2. Crear una nueva clase que se llame como el usuario quiera que será el programa a desarrollar, lo que se conoce como clase principal.

El funcionamiento se aprecia en el siguiente ejemplo:

```
public class NombreDelPrograma {

        public static void main (String[] arg) {

                EntradaDeTeclado entrada = new EntradaDeTeclado();

                System.out.println("Esto es un programa para probar la entrada de datos");

                System.out.println("Introduzca un texto: ");
                entrada.pedirEntradaTexto();
                System.out.println("La entrada de texto ha sido: " + entrada.getEntrada() );

                System.out.println("Introduzca un número: ");
                entrada.pedirEntradaNumero();
                System.out.println("La entrada numérica ha sido: " + entrada.getEntrada2() );
}
}
```

ORGANIZACIÓN EN LA API DE JAVA

Para programar en JAVA se tendrá que recurrir constantemente a la consulta de la documentación del API de JAVA. Esta documentación está disponible en la siguiente dirección:

https://docs.oracle.com/javase/8/docs/api/

Las clases de JAVA ser organizan en paquetes y hay que conocer dicha organización para encontrar las clases que se necessiten rápidamente.

Este sería un esquema orientativo:

java.lang – Contiene las clases fundamentales del lenguaje. Se carga automáticamente
java.util – Contiene las clases relacionades con colecciones de datos (Ejemplo: ArrayList)
java.io – Contiene las clases relacionades con la entrada y salida de datos
java.math – Contiene las clases que permiten operaciones y cálculos matemáticos
java.awt – Contiene las clases útiles para crear interfaces gráficas y para dibuixar figuras e imágenes
java.sql – Contiene las clases que permiten gestionar datos con bases de datos
java.net – Contiene las clases que permiten gestionar el trabajo en red y con Internet.

Para utilizar clases de estos paquetes en un código hay que importar dichos paquetes y dichas clases, mediante la sentencia import y en las primeras líneas del programa.

Ejemplo:

import java.util.Scanner; Se importa la clase Scanner del paquete util del API de JAVA.

import java.lang.;* Se importa el paquete lang del API de JAVA. El asterisco conlleva que se importen todas las clases contenidas en el paquete.

ORIENTACIÓN A OBJETOS

CLASE

Abstracción que **define** un tipo de objeto especificando qué propiedades (atributos) y operaciones disponibles va a tener (métodos). La clase define qué se puede hacer y qué no se puede hacer.

A las diferentes clases se les ponen nombres que comiencen por mayúsculas.

OBJETO

Entidad que **tiene** unas propiedades (atributos) y unas operaciones disponibles (métodos).

Un objeto es una instancia de una clase.

Cuando solo se tiene definida la clase no existen los objetos, éstos hay que crearlos para que existan.

A los diferentes objetos se les ponen nombres que comiencen por minúsculas.

ESTRUCTURA DE UNA CLASE

La estructura de una clase consta siempre de tres partes claramente diferenciadas, mediante el siguiente ejemplo se muestra la parte 1 (atributos), la parte 2 (constructor) y la parte 3 (métodos):

```
public class Taxi {

String Ciudad;
String matricula;
int tipoMotor;

públic Taxi() {

ciudad = "Madrid";
matricula = "";
tipoMotor = 0;

}

public void setMatricula (String valorMatricula) {

matricula = valorMatricula; }

public void setTipoMotor (int valorTipoMotor) {

matricula = valorMatricula; }

public String getMatricula () {

return matricula; }

public String getTipoMotor () {

return tipoMotor; }

}
```

ATRIBUTOS

Son las características que va a tener el objeto y se representan con variables.

CONSTRUCTOR

Es el encargado de inicializar las variables utilizadas como atributos a unos valores determinados. Esta acción se lleva a cabo cada vez que se crea un objeto.

Una clase puede disponer de cero, uno o más constructores. En el momento de crear el objeto se utiliza el que coincida en número y tipo de parámetros con el requerido en la instrucción para crear el objeto.

MÉTODOS

La ejecución de un método se denomina invocación del método. Los métodos disponibles para un objeto los define la clase, pero se invocan sobre cada objeto en particular.

nombreDelObjeto.nombreDelMetodo();

Se distinguen dos tipos de métodos: setters y getters.

GETTERS

Son métodos que devuelven algo. Un método es tipo función si comienza con un tipo que significa que devuelve una variable de ese mismo tipo. La devolución del resultado se expresa con la palabra reservada *return* seguida del dato a devolver.

SETTERS

Son métodos que realizan ciertas operaciones sin devolver un valor concreto. Un método es tipo procedimiento si comienza con la palabra reservada *void*.

ÁMBITO DE LAS VARIABLES

Los métodos o los constructores pueden requerir parámetros con tipos y nombres concretos. Estos parámetros tienen un ámbito limitado al método o constructor, es decir, tienen **ámbito local**.

Por otro lado, los atributos se pueden usar en toda la clase, es decir, tienen **ámbito global**.

SIGNATURA DE UN MÉTODO

Esto es el encabezado de un método y permite extraer cierta información relevante:

- Si el método es tipo función o tipo procedimiento
- El tipo del valor devuelto (si es un método tipo función)
- El nombre del método
- Los parámetros requerides y sus tipos

Esta información que indica lo que hace el método pero no cómo lo hace, se denomina interfaz.

SOBRECARGA DE NOMBRES

La sobrecarga de nombres se da cuando se tiene alguna variable local de un método o constructor, o algún parámetro de un método o constructor, con un nombre idéntico al de algun atributo de la clase.

Usando la palabra reservada *this* se puede evitar que los parámetros o variables locales tapen a las variables globales. Escribiendo *this* delante del nombre de la variable se le indica a JAVA que se hace referencia al atributo de la clase en vez de a una variable local o parámetro.

this.nombreDeLaVariable

Repetir los nombres de atributos como parámetros o variables locales queda a elección del programador.

CREACIÓN DE UN OBJETO

Para crear un objeto de una determinada clase se utiliza la siguiente sentencia:

NombreClase NombreObjeto = new NombreClase();

CLASE PRINCIPAL

Al llevar a cabo un programa siempre se creará una clase concreta denominada principal. En ella se incluirá el siguiente código y se sustituirán los puntos suspensivos por las líneas de código del programa:

```
public class Principal {

públic static void main(String[] arg) {

.....

}
```

Por otro lado, se crearán todas aquellas clases que el programador considere necesarias y se crearán objetos de dichas clases en la clase principal siempre que se necesiten.

Ejemplo de clase principal utilizando la clase Taxi creada con anterioridad:

```
public class Principal {

public static void main (String[] arg) {

Taxi taxi1 = new Taxi();
matricula = "1234 BCD";

taxi1.setMatricula(matricula);

System.out.println(taxi1.getMatricula());

}
}
```

DIFERENCIA ENTRE IGUALDAD E IDENTIDAD ENTRE OBJETOS

La comparación usando == no se debe usar para comparar el contenido de los objetos, sino únicamente para comparar la información de las variables apuntadoras. Usar == para comparar objetos es un error muy común.

Para comparar el contenido de los objetos hemos de usar, en general, un método especial del que disponen todos los objetos, denominado *equals*.

La comparación de identidad entre objetos requiere que se defina en base a qué va a considerar que dos objetos son iguales. Por ejemplo en un objeto de la clase String está claro, sería en base al contenido de las cadenas de texto.

cadena1.equals(cadena2) -> devuelve true si la cadena1 tiene el mismo contenido que la cadena2

Sin embargo, en un objeto de la clase Persona, podria ser en base al dni, al nombre, al número de la seguridad social, etc.

Los tipos primitivos se comparan de la forma habitual mediante el operador ==.

ASIGNACIÓN ENTRE OBJETOS

No se puede usar el operador de asignación = entre objetos de la misma forma en que se utiliza con tipos primitivos.

La manera de hacer dicha asignación sería la siguiente:

Se parte de un objeto con cierto contenido para a continuación crear otro objeto y establecer el contenido de los atributos de éste usando los métodos setters y asignando los valores devueltos por los métodos getters del primer objeto.

objeto1.setNombre(objeto2.getNombre());

CADENAS DE TEXTO

El tipo de la cadena de texto en JAVA es la clase String. Dicha clase tiene muchos métodos para operar con las cadenas de texto tal y como se puede observar en la API de JAVA.

Algunos ejemplos de estos métodos se muestran a continuación:

Extraer subcadenas de texto de cadenas de texto: substring

Ejemplo de uso: *"pruebas".substring(1,5)* -> devolvería "rueba"

Obtener la longitud de una cadena de texto: length

Ejemplo de uso: "pruebas".length() -> devolvería 7

Separar una cadena de texto en caracteres: charAt

Ejemplo de uso:

```
String cadena = "pruebas";
char letra = cadena.charAt(1);
```

Este método devuelve el caracter de la posición especificada, en este caso 'r'.

COLECCIONES DE DATOS

Existen dos tipos de colecciones para almacenar datos:

- De tamaño fijo: Permiten almacenar tanto objetos como tipos primitivos. **Arrays**
- De tamaño flexible: Sólo permiten almacenar objetos. **ArrayList**

Las colecciones de tamaño flexible permiten modificar dinámicamente el número de ítems que contienen, es decir, ampliarlo o reducirlo. Son arrays dinámicos, permiten crear colecciones de tamaño variable que se pueden agrandar o empequeñecer en función de las necesidades.

Las colecciones de tamaño fijo se utilizan cuando se conoce el número de elementos que va a contener la colección y éste va a ser invariable.

CLASE ARRAYS

Un array en JAVA es una clase dentro del paquete util: java.util.Arrays. Se utiliza de la siguiente forma:

Declaración: TipoPrimitivoUobjeto [] nombreArray;

Creación: nombreArray = new TipoPrimitivoUobjeto [longitud]

Declarar y crear en la misma línea: Tipo [] nombreArray = {valor1, valor2, valor3};

Ejemplo: int [] array1 = {1, 2, 3}

Se **asignan** valores a las diferentes casillas de la manera habitual: array1[0] = 1

Se **accede** a los valores de las diferentes casillas de la manera habitual: array[0]

Se conoce la **longitud** de un array mediante el atributo length: array1.length -> devolvería 3

Se **comparan** arrays mediante el método equals: Arrays.equals(array1, array2)

Se **copia** un array 1 en otro array 2 de forma automática mediante el método copyOf:

array2 = Arrays.copyOf (array1, array1.length)

Se **rellena** un array con un valor determinado mediante el método fill:

Arrays.fill (array1,10) -> El array queda relleno con todas sus casillas puestas a 10

Se **transforma** un array en un arraylist mediante el método asList: Lista1 = Arrays.asList(array1)

Se **pasa un array a cadena de texto** mediante el método toCharArray:

```
String cadena = "pruebas";
char[] array1;
char[] array1 = cadena.toCharArray();
```

Se **pasa una cadena de texto a array** mediante el método valueOf:

```
char array1[] = { 'p', 'r', 'u', 'e', 'b', 'a', 's' };
String cadena = String.valueOf(array1);
```

CLASE ARRAYLIST

Una lista en JAVA es una clase dentro del paquete util: java.util.ArrayList. Un objeto de tipo ArrayList será una lista redimensionable en la que se tendrán disponibles los métodos más habituales para operar con los elementos de dicha lista. Los elementos de una lista tienen un orden, que es el orden en el que se insertaron en la misma.

Muy importante acordarse de que son colecciones de objetos, por lo que un ArrayList no puede ser una lista de enteros como tipo primitivo (int) pero si una lista de enteros objeto (Integer).

Se utiliza de la siguiente forma:

Declaración: ArrayList<TipoDeObjetosEnLaColeccion> NombreArrayList;

Creación: NombreObjeto = new ArrayList<TipoDeObjetosEnLaColeccion>();

Operaciones:

Añadir objeto al final: NombreArrayList.add(Objeto);

Reemplazar objeto existente: NombreArrayList.set(posicion, objeto);

Extraer un objeto de cierta posición: NombreArrayList.get(posicion);

Obtener número de objetos de la lista: NombreArrayList.size();

CLASE RANDOM

Generar números aleatorios en JAVA se lleva a cabo mediante una clase que se encuentra dentro del paquete util: java.util.Random. Se crea un objeto de tipo Random y luego se invoca un método sobre dicho objeto para que devuelva el número aleatorio.

Se utiliza de la siguiente forma:

Declaración: Random numeroAleatorio;

Creación: numeroAleatorio = new Random();

Operaciones:

aleatorio.nextInt(valor);

Este código genera un número entero aleatorio comprendido entre 0 (incluido) y valor (excluido).

aleatorio.nextBoolean(valor);

Este código genera un valor true o false de forma aleatoria.

CONVERSIÓN DE TIPOS

Transformar el tipo de una variable u objeto en otro diferente al original con el que fue declarado se realiza colocando el nuevo tipo entre paréntesis a la izquierda de la variable que se quiere convertir. No todos los tipos se convertiran de forma segura.

Tipo VariableNueva = (NuevoTipo) VariableAntigua;

ORGANIZACIÓN EN PAQUETES

Los proyectos en JAVA se suelen organizar en paquetes. El paquete no es más que un contenedor que contiene en su interior elementos relacionados entre sí.

Un package es una agrupación de clases afines.

Ejemplo de uso:

```
package TiposBasicos;

public class Persona {

.....

}

public class Animal {

.....

}
```

```
package Conductores;

Import TiposBasicos.*;

public class taxista {

Persona taxista1;

}
```

JAVADOC

Documentar un proyecto es algo fundamental de cara a su futuro mantenimiento. Cuando se programa una clase, se debe generar documentación lo suficientemente detallada sobre ella como para que otros programadores sean capaces de usarla sólo con su interfaz.

No debe existir necesidad de leer o estudiar su implementación, de la misma manera que para usar una clase del API de JAVA no se lee ni se estudia su código fuente.

Javadoc es el estándar para documentar clases de JAVA.

Al documentar una clase se debe incluir:

Nombre de la clase / **Descripción general** / **Número de versión** / **Nombre de autores** / **Documentación de cada constructor o método** incluyendo: Nombre del constructor o método, Tipo de retorno, Nombres y tipos de parámetros si los hay, Descripción general, Descripción de parámetros si los hay, Descripción del valor de retorno.

Para que javadoc sea capaz de generar documentación automáticamente han de seguirse estas reglas:

- La documentación para javadoc ha de incluirse entre los siguientes símbolos:

  ```
  /**
  * Esto es un comentario para javadoc
  * Esto es un comentario para javadoc
  */
  ```

- La ubicación le define a javadoc qué representa el comentario: si está incluido justo antes de la declaración de clase se considerará un comentario de clase y si està incluido justo antes de la cabecera de un constructor o método se considerará un comentario de ese constructor o método.

- Para alimentar javadoc se usan ciertas palabras reservades que van precedidas del carácer @:

  @author: Nombre del desarrollador

  @deprecated: Indica que el método o clase es obsoleto

  @param: Define un parámetro de un método

  @return: Informa de lo que devuelve el método

  @see: Asocia con otro método o clase

  @version: Versión del método o clase

- Las etiquetas @author y @version se usan para documentar clases e interfaces, por tanto no son válidas en cabeceras de constructores ni métodos.

- La etiqueta @param se usa para documentar constructores y métodos.

- La etiqueta @return se usa solo en métodos de tipo función.

- Dentro de los comentarios se admiten etiquetas HTML.

POLIMORFISMO

JAVA permite declarar variables sin especificar la clase concreta a la que van a pertenecer, sino únicamente diciendo qué interfaz van a cumplir.

Por ejemplo si un método pide como parámetro un tipo List significa que admite que se le pase como parámetro tanto un ArrayList como otro objeto que cumpla con la interfaz. El parámetro sería polimórfico porque admite distintas formas de objeto.

Ejemplo:

```
ArrayList<List> lista1 = new ArrayList<List>();
ArrayList<Integer> lista2 = new ArrayList<Integer>();
ArrayList<String> lista3 = new ArrayList<String>();
lista1.add(lista2);
lista1.add(lista3);
```

lista1 es una variable polimórfica porque contiene objetos que no son solo de un tipo, sino de varios.

HERENCIA

La herencia permite definir una clase como extensión de otra, es decir, la clase hija tiene todas las características de la clase padre y además sus características particulares.

Todo lo que es común a ambas clases queda comprendido en la clase superior (superclase), mientras lo que es específico queda restringido a las clases inferiores (subclases).

La aportación de la herencia es que permite evitar la duplicidad de código, es decir, aquello que es común a varias clases se escribe solo una vez (en la clase padre).

En JAVA, que una clase derive de otra se indica mediante la palabra reservada *extends*.

Muchas subclases pueden heredar de una misma superclase y a su vez una subclase puede convertirse en superclase de otra.

Los atributos privados de una superclase no son accesibles por una subclase directamente. Para acceder a ellos tendrá que hacer uso de métodos de acceso o modificación. Una subclase puede invocar a cualquier método público de su supercalse como si fuese propio.

Ejemplo:

```
public class Persona {

private String nombre;
private int edad;

public Persona (String valornombre, int valoredad) {

nombre = valornombre;
edad = valoredad;

}

public String getNombre() {return nombre;}
public int getEdad() {return edad;}

}

public class Profesor extends Persona{

private String codigoEmpleado;

public Profesor (String valornombre, int valoredad) {

super(valornombre, valoredad);    <- El constructor de la subclase invoca al constructor de la superclase
codigoEmpleado="";

}

public void setcodigoEmpleado(String valorcodigoEmpleado){codigoEmpleado = valorcodigoEmleado;}
public getcodigoEmpleado(){return codigoEmpleado;}

}
```

MODIFICADORES DE ACCESO

En JAVA existen cuatro modificadores de acceso: public, protected, private y no especificar ninguno.

Estos modificadores se pueden aplicar a todos los miembros de una clase, es decir, a atributos, métodos y constructores.

La visibilidad de los elementos según el modificador de acceso aplicado sería la siguiente:

MODIFICADOR	CLASE	PACKAGE	SUBCLASE	TODOS
public	si	si	si	si
protected	si	si	si	no
private	si	no	no	no
no especificado	si	si	no	no