Unidad V

Implementación Orientada a Objetos.

5.1 Estructura de una clase.

Una clase consiste en:

algunas_palabras class nombre_de_la_clase [algo_más] {
[lista_de_atributos]
[lista_de_métodos]
}

Lo que está entre [y] es opcional

Ya veremos qué poner en "algunas_palabras" y "algo_más", por ahora sigamos un poco más.

La lista de atributos (nuestras viejas variables locales) sigue el mismo formato de C: se define primero el tipo y luego el nombre del atributo, y finalmente el ";".

public final static int MAX_VALUE

;

También tenemos "algunas_palabras" adelante, pero en seguida las analizaremos.

En cuanto a los métodos, también siguen la sintaxis del C; un ejemplo:

```
public int incContador() { // declaración y apertura de {
cnt++; // instrucciones, separadas por ";"
return(cnt);
} // cierre de }
```

Finalmente, se aceptan comentarios entre /* y */, como en C, o bien usando // al principio del comentario (el comentario termina al final de la línea).

Veamos un ejemplo:

```
// Implementación de un contador sencillo
// GRABAR EN UN ARCHIVO "Contador.java" (OJO CON LAS MAYUSCULAS!)
// COMPILAR CON: "javac Contador.java" (NO OLVIDAR EL .java!)
// ESTA CLASE NO ES UNA APLICACION, pero nos va a servir enseguida

public class Contador { // Se define la clase Contador

// Atributos
int cnt; // Un entero para guardar el valor actual

// Constructor // Un método constructor…
```

```
public Contador() { // …lleva el mismo nombre que la clase
cnt = 0; // Simplemente, inicializa (1)
}

// Métodos
public int incCuenta() { // Un método para incrementar el contador
cnt++; // incrementa cnt
return cnt; // y de paso devuelve el nuevo valor
}
public int getCuenta() { // Este sólo devuelve el valor actual
return cnt; // del contador
}
}
```

Cuando, desde una aplicación u otro objeto, se crea una instancia de la clase Contador, mediante la instrucción:

```
new Contador()
```

el compilador busca un método con el mismo nombre de la clase y que se corresponda con la llamada en cuanto al tipo y número de parámetros. Dicho método se llama Constructor, y una clase puede tener más de un constructor (no así un objeto o instancia, ya que una vez que fue creado no puede recrearse sobre sí mismo).

En tiempo de ejecución, al encontrar dicha instrucción, el intérprete reserva espacio para el objeto/instancia, crea su estructura y llama al constructor.

O sea que el efecto de new Contador() es, precisamente, reservar espacio para el contador e inicializarlo en cero.
En cuanto a los otros métodos, se pueden llamar desde otros objetos (lo que incluye a las aplicaciones) del mismo modo que se llama una función desde C.

Por ejemplo, usemos nuestro contador en un programa bien sencillo que nos muestre cómo evoluciona:

```
// Usemos nuestro contador en una mini-aplicación
// GRABAR EN UN ARCHIVO "Ejemplo1.java" (OJO CON LAS MAYUSCULAS!)
// COMPILAR CON: "javac Ejemplo.java" (NO OLVIDAR EL .java!)
// EJECUTAR CON: "java Ejemplo1" (SIN el .java)

import java.io.*; // Uso la biblioteca de entradas/salidas

public class Ejemplo1 { // IMPORTANTE: Nombre de la clase
// igual al nombre del archivo!
// entero para asignarle el valor del contador e imprimirlo
// aunque en realidad no me hace falta.
static int n;
// y una variable tipo Contador para instanciar el objeto…
static Contador laCuenta;

// ESTE METODO, MAIN, ES EL QUE HACE QUE ESTO SE COMPORTE
// COMO APLICACION. Es donde arranca el programa cuando ejecuto "java Ejemplo1"
// NOTA: main debe ser public & static.
```

```
public static void main ( String args[] ) {
System.out.println ("Cuenta… "); // Imprimo el título
laCuenta = new Contador(); // Creo una instancia del Contador
System.out.println (laCuenta.getCuenta()); // 0 - Imprimo el valor actual (cero!)
n = laCuenta.incCuenta(); // 1 - Asignación e incremento
System.out.println (n); // Ahora imprimo n
laCuenta.incCuenta(); // 2 - Lo incremento (no uso el valor…
System.out.println (laCuenta.getCuenta()); // …de retorno) y lo imprimo
System.out.println (laCuenta.incCuenta()); // 3 - Ahora todo en un paso!
}
}
```

5.2 Elementos de una clase.

Atributos o Propiedades
Tipo de características y propiedades que las entidades puedan obtener. Los atributos distinguen un objeto de los restantes (tamaño, posición, color, ...). Cada propiedad tendrá un determinado valor. Las propiedades de un objetos pueden ser heredadas por sus descendientes.
SINTAXIS
(ACCESO)

- Public
- Private
- protected

(MODIFICADOR)

- Static
- Final

tipoDato

- Tipos básicos
    1. byte, int, float, double
    2. char, boolean
- Wrapper
    1. Integer, String, Double, Float
    2. Clase definidas por el programador

TIPOS DE ATRIBUTOS
De Instancia :
Estos atributos permiten almacenar los datos particulares de un objeto. Se denominan de instancia porque se estructuran con el objeto cuando este cerca.
Ejemplo:

```
public class Triangulo{
        private int base;
        private int altura;
        private int area:
```

}
De Clase:
Estos atributos permiten almacenar datos que van a ser compartidos por muchos objetos. Estos atributos, no se instancia con el objeto cuando este es cerrado.
Ejemplo:

```
public class Casa{
        private static String empresaEnergia;
        private static String empresaAgua;
}
```

Constantes:
Estos atributos representan valores constantes de los objetos.
Ejemplo:

```
public class Circulo{
    private final double PI= 3.1416;
}
public class Circulo{
    private static final double PI= 3.1416;
}
```

OPERACIONES
Es una acción que el objeto puede realizar. para implementar este concepto en Lenguaje de Programación Java, debemos recurrir al concepto de función (subprograma que realiza una tarea concreta) conjunto de cosas que puede hacer un objeto (estudiar, caminar, trabajar, rotas, volar, etc.). Un método es un procedimiento o función que altera el estado de un objeto o hace que el objeto envíe un mensaje, es decir, que devuelva los valores.
SINTAXIS

```
[acceso] tipoDevuelto nombreDeLaOperación( [parámetros] ){
                Definición de la operación (método)
}
```

TIPOS DE OPERACIONES
Constructoras:
Estas operaciones se encargan de inicializar los atributos de un objeto cuando este se está creando:

- Tiene el mismo nombre de la clase
- No retornan valor (no se le especifican tipoDevuelto)
- Se llaman automáticamente cuando el objeto se crea (llamado implícito)

5.3 Clase principal.

Lo habitual en una aplicación medianamente seria es que no tengamos una sola "clase", como hasta ahora, sino que realmente existan varios objetos de distintas clases, que se relacionan entre sí.

En Java podemos definir varias clases dentro de un mismo fichero, con la única

condición de que sólo una de esas clases sea declarada como "pública". En un caso general, lo más correcto será definir cada clase en un fichero. Aun así, vamos a ver primero un ejemplo que contenga dos clases en un solo fichero

```
// DosClases.java
// Primer ejemplo de una clase nuestra
// que accede a otra también nuestra,
// ambas definidas en el mismo fichero
// Introducción a Java

class Principal {



public static void main ( String args[] ) {

Secundaria s = new Secundaria ();
 s. saluda (); // Saludo de "Secundaria"
Saluda (); // Saludo de "Principal"
}
public static void saluda() {
System.out.println (“Saludando desde <Principal>“);
}
}
// ----------------------------------------------------
class Secundaria {
```

```
public void saluda () {
System.out.println (“Saludando desde <Secundaria>“);
}



}
```

5.4 Crear objetos.

En Java, se crea un objeto mediante la creación de un objeto de una clase o, en otras palabras, ejemplarizando una clase. Aprenderás cómo crear una clase más adelante en **Crear Clases**.

Hasta entonces, los ejemplos contenidos aquí crean objetos a apartir de clases que ya existen en el entorno Java.

Frecuentemente, se verá la creación de un objeto Java con un sentencia como esta.

```
Date hoy = new Date();
```

Esta sentencia crea un objeto Date (Date es una clase del paquete java,util). Esta sentencia realmente realiza tres acciones: declaración, ejemplarización e inicialización.

Date hoy es una declaración de variable que sólo le dice al compilador que el nombre **hoy** se va a utilizar para referirse a un objeto cuyo tipo es Date, el operador **new** ejemplariza la clase Date (creando un nuevo objeto Date), y **Date()** inicializa el objeto.

Declarar un Objeto

Ya que la declaración de un objeto es una parte innecesaria de la creación de un objeto, las declaraciones aparecen frecuentemente en la misma línea que la creación del objeto. Como cualquier otra declaración de variable, las declaraciones de objetos pueden aparecer solitarias como esta.

```
Date hoy;
```

De la misma forma, declarar una variable para contener un objeto es exactamente igual que declarar una variable que va a contener un tipo primitivo.

tipo nombre
donde tipo es el tipo de dato del objeto y nombre es el nombre que va a utilizar el objeto. En Java, las clases e interfaces son como tipos de datos. Entonces tipo puede ser el nombre de una clase o de un interface.

Las declaraciones notifican al compilador que se va a utilizar nombre para referirse a una variable cuyo tipo es tipo. **Las declaraciones no crean nuevos objetos. Date hoy** no crea un objeto Date, sólo crea un nombre de variable para contener un objeto Date. Para ejemplarizar la clase Date, o cualquier otra clase, se utiliza el operador **new**.

Ejemplarizar una Clase

El operador **new** ejemplariza una clase mediante la asignación de memoria para el objeto nuevo de ese tipo. **new** necesita un sólo argumento: una llamada al método constructor. Los métodos constructores son métodos especiales proporcionados por cada clase Java que son reponsables de la inicialización de los nuevos objetos de ese tipo. El operador **new** crea el objeto, el constructor lo inicializa.

Aquí tienes un ejemplo del uso del operador **new** para crear un objeto Rectangle (Rectangle es una clase del paquete java.awt).

new Rectangle(0, 0, 100, 200);
En el ejemplo, **Rectangle(0, 0, 100, 200)** es una llamada al constructor de la clase Rectangle.

El operador **new** devuelve una referencia al objeto recien creado. Esta referencia puede ser asignada a una variable del tipo apropiado.

Rectangle rect = new Rectangle(0, 0, 100, 200);
(Recuerda que una clase esencialmente define un tipo de dato de referencia. Por eso, Rectangle puede utilizarse como un tipo de dato en los programas Java. El valor de cualquier variable cuyo tipo sea un tipo de referencia, es una referencia (un puntero) al valor real o conjunto de valores representado por la variable.