Tema 3: Programación orientada a objetos.

1. Objetos.

2. Clases

2.1. Formato de una clase en Java.

3. Instanciación de los objetos. Declaración y creación.

4. Métodos.

4.1. El método Main.

4.2. Constructores.

4.3. Métodos Set y Get.

4.4. Sobrecarga de métodos.

4.5. Métodos static, campos static y clase Math.

5. Referencias a miembros del objeto actual mediante this.

6. Modificadores de acceso.

7. Librerías de objetos: paquetes.

1. Objetos.

Entender qué es un objeto es la clave para entender cualquier lenguaje orientado a objetos.

Primero empecemos entendiendo qué es un objeto del mundo real. Un objeto del mundo real es cualquier cosa que vemos a nuestro alrededor. Digamos que para leer este tema lo hacemos a través del monitor y un ordenador, ambos son objetos, al igual que nuestro teléfono, un árbol o un coche.

Analicemos un poco más a un objeto del mundo real, como el ordenador. No necesitamos ser expertos en hardware para saber que un ordenador está compuesto internamente por varios componentes: la placa, el chip del procesador, un disco duro, una tarjeta gráfica, y otras partes más. El trabajo en conjunto de todos estos componentes hace trabajar a un ordenador.

Internamente, cada uno de estos componentes puede ser sumamente complicado y puede ser fabricado por diversas compañías con diversos métodos de diseño. Pero nosotros no necesitamos saber cómo trabajan cada uno de estos componentes, como saber qué hace cada uno de los chips de la placa, o cómo funciona internamente el procesador. Cada componente es una unidad autónoma, y todo lo que necesitamos saber es cómo interactúan entre sí los componentes, saber por ejemplo si el procesador y las memorias son compatibles con la placa, o conocer dónde se coloca la tarjeta gráfica. Cuando conocemos cómo interaccionan los componentes entre sí, podremos armar fácilmente el ordenador.

¿Qué tiene que ver esto con la programación? La programación orientada a objetos trabaja de esta manera. Todo el programa está construido en base a diferentes componentes (objetos), cada uno tiene un rol específico en el programa y todos los componentes pueden comunicarse entre ellos de formas predefinidas.

Podemos pensar que todo objeto del mundo real tiene 3 componentes: nombre, características y comportamiento.

Por ejemplo, a lo que damos nombre de automóvil tiene características (marca, modelo, color, velocidad máxima, etc.) y comportamiento (frenar, acelerar, retroceder, llenar combustible, cambiar llantas, etc.).

Los Objetos de Software, al igual que los objetos del mundo real, también tienen nombre, características y comportamientos. Un objeto de software tiene un identificador, mantiene sus características en una o más "variables" llamadas

atributos, e implementa su comportamiento en funciones o subrutinas llamadas métodos.

Imaginemos que tenemos un Ford Focus color azul que alcanza una velocidad de 260 km/h. Si pasamos ese objeto del mundo real al mundo del software, tendremos un objeto Automóvil con sus características predeterminadas:

Marca = Ford Modelo = Focus Color = Azul

Velocidad Máxima = 260 km/h

Cuando a las características del objeto le ponemos valores decimos que el objeto tiene estados. Las variables almacenan los estados de un objeto en un determinado momento.

El método describe los mecanismos que se encargan de realizar sus tareas; y oculta al usuario las tareas complejas que realiza. Por ejemplo, el pedal del acelerador de un automóvil oculta al conductor los complejos mecanismos para hacer que el automóvil vaya más rápido. Esto permite que las personas con poco o nada de conocimiento acerca de cómo funcionan los motores puedan conducir un automóvil con facilidad.

Definición teórica: Un objeto es una unidad de código compuesto de atributos y métodos relacionados.

En resumen:

Un objeto en un sistema posee: una identidad, un estado y un comportamiento.

La identidad es la propiedad que determina que cada objeto es único, aunque tenga el mismo estado. No existen dos objetos iguales.

El estado son los valores concretos que tiene un objeto en cada uno de sus atributos. Marca las condiciones de existencia del objeto dentro del programa. Lógicamente este estado puede cambiar. Un coche puede estar parado, en marcha, estropeado, funcionando, sin gasolina, etc. El estado lo marca el valor que tengan las propiedades del objeto.

El comportamiento determina como responde el objeto ante peticiones de otros objetos. Por ejemplo, un objeto conductor puede lanzar el mensaje arrancar a un coche. El comportamiento determina qué es lo que hará el objeto (es decir, qué ocurre cuando se arranca el coche). El comportamiento está relacionado por tanto con los métodos.

2. Clases.

Las clases son las plantillas para hacer objetos. Una clase sirve para definir una serie de objetos con propiedades (atributos), comportamientos (operaciones o métodos) y semántica comunes. Hay que pensar en una clase como un molde.

Supongamos que queremos conducir el automóvil del ejemplo anterior. Para hacer que aumente su velocidad, debe presionar el pedal del acelerador. ¿Qué debe ocurrir antes de que podamos hacer esto? Bueno, antes de poder conducir un automóvil, alguien tiene que diseñarlo. Por lo general, un automóvil empieza en forma de dibujos de ingeniería, similares a los planos de construcción que se utilizan para diseñar una casa.

Estos dibujos de ingeniería incluyen, por ejemplo, el diseño del pedal del acelerador para que el automóvil aumente su velocidad. A partir de los dibujos de ingeniería se pueden fabricar muchos automóviles del mismo modelo. Los dibujos de ingeniería del automóvil serían un símil de CLASE, y los automóviles que sacamos a partir de los dibujos serían un símil de OBJETOS.

Desafortunadamente, no se pueden conducir los dibujos de ingeniería de un automóvil. Antes de poder conducirlo, éste debe construirse a partir de los dibujos de ingeniería que lo describen.

Así como no podemos conducir un dibujo de ingeniería de un automóvil, tampoco podemos "conducir" una clase. De la misma forma que alguien tiene que construir un automóvil a partir de sus dibujos de ingeniería para poder conducirlo, también debemos construir un objeto de una clase para poder hacer que un programa realice las tareas que la clase le describe cómo realizar.

En resumen:

Una clase representa al conjunto de objetos que comparten una estructura y un comportamiento comunes. Una clase es una combinación específica de atributos y métodos y puede considerarse un tipo de dato de cualquier tipo no primitivo. Así, una clase es una especie de plantilla o prototipo de objetos: define los atributos que componen ese tipo de objetos y los métodos que pueden emplearse para trabajar con esos objetos. Aunque, por otro lado, una clase también puede estar compuesta por métodos estáticos que no necesitan de objetos (como las clases construidas en los temas anteriores que contienen un método estático main).

Antes de poder utilizar un objeto se debe definir la clase a la que pertenece, esa definición incluye:

- El nombre o identificador de clase. Debe empezar con letra mayúscula y seguirle letras minúsculas (si el nombre se compone de varias palabras, la inicial de cada palabra se deja en mayúscula).

- Sus atributos. Es decir, los datos miembros de esa clase. A los atributos también se les llama propiedades y campos. Los atributos son valores que poseerá cada objeto de la clase y por lo tanto marcarán el estado de los mismos. Se declaran con un tipo de dato correspondiente y un nombre identificador.

- Sus métodos. Las funciones miembro de la clase. Son las acciones (u operaciones) que puede realizar la clase. Sin duda es lo más complicado de programar y delimitar. Las clases se comunican entre sí invocando a métodos (a esto se le llama enviar mensajes).

El formato general para crear una clase en Java es:

```
[acceso] class NombreDeClase {
[acceso] [static] tipo atributo1;
 [acceso] [static] tipo atributo2;
 [acceso] [static] tipo atributo3;
...
[acceso] [static] tipo
nombreMétodo1([listaDeArgumentos]) {
...código del método...
}
[acceso] [static] tipo
nombreMétodo2([listaDeArgumentos]) {
...código del método...
}
...
}
```

La palabra opcional static sirve para hacer que el método o la propiedad a la que precede se pueda utilizar de manera genérica, los métodos o propiedades así definidos se llaman atributos de clase y métodos de clase respectivamente. Su uso se verá más adelante.

En Java cada clase debe ocupar un archivo, que además debe de tener el mismo nombre. Hasta ahora identificábamos archivo con programa; bien, en realidad no es así. En un archivo se declara y define una clase, que tiene que tener exactamente el mismo nombre que el archivo.

El método main, no se declara en cada archivo. La mayoría de clases no tienen métodos main, sólo disponen de él las clases que se pueden ejecutar (en cada proyecto real hay una).

3. Instanciación de los objetos. Declaración y creación.

Una instancia es un elemento tangible (ocupa memoria durante la ejecución del programa) generado a partir de una definición de clase. Todos los objetos empleados en un programa han de pertenecer a una clase determinada.

Un objeto, como variable que es, tiene un espacio de memoria asignado con el fin de guardar los datos que contiene la clase a que pertenece.

La instanciación de un objeto consiste, precisamente, en poder asignarle dicha memoria que le hace falta para poder guardar datos.

Para poder realizar esto, hay que llevar a cabo estos pasos:

Declarar objeto: ClaseDeObjeto nombreObjeto;

Crear objeto (instanciar): nombreObjeto = new ClaseDeObjeto();

Para crear un objeto se tiene que haber definido previamente la clase a la que pertenece el objeto. Para ello primero necesitamos declarar una referencia al objeto lo cual se hace igual que al declarar una variable, es decir se indica la clase de objeto que es y se indica su nombre, es decir la sintaxis es Clase objeto;

Ejemplo:

Automovil miAutomovil;

Esta instrucción declara miAutomovil, que será una referencia al tipo Automovil. La creación del objeto se hace con el identificador new: objeto= new Clase(); Ejemplo:

miAutomovil= new Automovil();

En un solo paso: Clase objeto= new Clase();

En un solo paso: Automovil miAutomovil =new Automovil();

4. Métodos.

Un método es una llamada a una operación sobre un determinado objeto. Al realizar esta llamada (también se le llama enviar un mensaje), el control del programa pasa a ese método y lo mantendrá hasta que el método finalice.

Hay métodos que devuelven un resultado (gracias a la palabra return). Cuando se define este tipo de método hay que indicar el tipo de dato que devuelve.

Si el método no devuelve ningún resultado se define en su cabecera como tipo void.

Dentro de cada método se pueden utilizar las estructuras de programación que hemos visto hasta ahora en el curso como: la estructura for, if, los operadores, las variables, etc.

Los métodos pueden admitir en su definición una serie de valores. A estos valores se les denomina parámetros o argumentos. Los parámetros pueden tener un tipo básico (char, int, boolean, etc.) o bien ser un objeto.

Cuando una clase ya tiene definidos sus métodos, es posible invocarlos utilizando los objetos definidos de esa clase. En esa invocación, se deben indicar los parámetros (o argumentos) que cada método requiere para poder realizar su labor.

Usar métodos de una clase: nombreObjeto.nombre_método();

Las llamadas a los métodos para realizar las distintas acciones se llevan a cabo separando los identificadores de la referencia y del método correspondiente con el operador punto.

Ejemplo: miAutomovil.getVelocidad(); miAutomovil.acelera(50);

Para los métodos hay que tener en cuenta lo siguiente:

1. Sus especificadores de alcance o visibilidad. Si el alcance es private, el método sólo se podrá utilizar dentro de otro método en la misma clase; si es public podrá ser invocado desde cualquier clase; si es protected desde la propia clase y sus descendientes y si no lleva especificador, desde clases que estén en el mismo paquete.

2. El tipo de datos o de objeto que devuelve. Si el resultado del método es un número entero, o un booleano, o un String o un objeto de una clase determinada, etc. Si el método no devuelve valor alguno se debe indicar como tipo el valor void.

3. El identificador del método. El identificador de un método debe empezar por una letra en minúscula, y si se enlaza más de una palabra, se unirán por guiones bajos .

4. Los parámetros. Los métodos pueden necesitar datos para realizar su tarea. Dichos parámetros en realidad son una lista de variables u objetos y los tipos o clases de los mismos. La existencia de esas variables u objetos está ligada a la del propio método; es decir, cuando el método finaliza, los parámetros se eliminan.

5. El cuerpo del método. Es decir, el código que permite al método realizar su tarea. Es lo más complicado. Dentro de ese código se pueden declarar variables, objetos y utilizar cualquier conjunto de instrucciones de Java, así como invocar a métodos de otras clases y objetos (si disponemos de visibilidad para ello). El valor resultado del método si lo hay se realiza mediante la instrucción return.

EJEMPLO: Supongamos la clase Vehículo con atributos ruedas y velocidad, y los métodos: acelerar, frenar y obtenerVelocidad. Esto podemos representarlo gráficamente y en código, de la forma:

CLASE VEHICULO

```
package Ejemplo;

public class

Vehiculo {

        public int ruedas;

        private double
```

```
        velocidad=0;public
        Vehiculo(){
        }
        public void acelerar(double
                cantidad) {velocidad +=
                cantidad;
        }
        public void frenar(double
                cantidad) {velocidad -=
                cantidad;
        }
        public double
                obtenerVelocidad(){
                return velocidad;
        }
  }
```

CLASE PRINCIPAL

```
package Ejemplo;
public class
Principal {
   public static void main(String
      args[]){Vehiculo miCoche =
      new Vehiculo();
      miCoche.acelerar(12);
      miCoche.frenar(5);
      System.out.println(miCoche.obtenerVelocidad());
   }
```

}

4.1 El método Main.

El método main es el método principal de cualquier programa en Java. Es el primero que se ejecuta y siempre debe declararse como static y void.

El método main se declara con el siguiente encabezado:

```
public static void main( String args[ ] )
```

Cualquier clase puede contener un método main, pero hay que tener en cuenta que la JVM invoca solo a un método main por ejecución.

Algunos programadores aprovechan que se puede crear un método main en cualquier clase para crear un pequeño programa de prueba para la propia clase.

4.2 Métodos constructores.

Cada clase que se declare puede proporcionar un constructor, el cual puede utilizarse para inicializar un objeto de una clase al momento de crear ese objeto. De hecho, Java requiere una llamada al constructor para cada objeto que se crea.

Hay dos tipos de métodos constructores:

a.- El método constructor por defecto: Es el que no tiene parámetros.

Si el programador no lo escribe, Java lo crea automáticamente e inicializa los valores de losatributos. Las variables de los tipos char, byte, short, int, long, float y double se inicializan con 0, las variables de tipo boolean se inicializan con false, y las variables de tipo por referencia se inicializan con null.

Si el programador sí lo escribe, puede hacerlo así:

```
public Pajaro() {
}
```

donde los atributos se inicializarían como en el caso anterior; o puede especificar:

```
public Pajaro() {
```

```
        nombre="Pepe";
        posX=3;
        posY=5;
    }
```

Donde en este caso cualquier objeto creado con este constructor recibiría el nombre"Pepe", la posiciónX 3 y la posiciónY 5.

b.-El método constructor que tiene parámetros: Tiene argumentos que se utilizarán paradar valor a los atributos del objeto que se está creando.

```
public Pajaro(String nombre, int posX, int posY){
    this.nombre=nombre;
    this.posX=posX;
     this.posY=posY;
    }
```

NOTA: Una función constructora, no puede devolver ningún parámetro (ni siquiera puedeser void).

Para CREAR OBJETOS La palabra clave **new** llama al constructor de la clase para realizar la inicialización. La llamada al constructor se indica mediante el nombre de la clase,seguido de paréntesis; el constructor debe tener el mismo nombre que la clase.

Pasos:

1.- Primero se declara el objeto:

```
Pajaro  loro;
```

2.- Luego se hace una instanciación del objeto, y se hace una

```
inicialización:loro = new Pajaro("Lucy",50,50) ;
```

Nota: o en una vez `Pajaro  loro= new Pajaro("Lucy",50,50) ;`

4.3 Métodos Set y Get.

En Java es habitual la práctica de ocultar los atributos y trabajar exclusivamente con los métodos. La razón es que no es recomendable que los atributos sean visibles desde fuera de la clase, por ello se declaran con una visibilidad private (o protected).
Siendo así ¿cómo pueden las otras clases modificar el valor de una propiedad? Mediante métodos que permitan la lectura y escritura de esas propiedades, son los métodos get (obtener) y set (ajustar).

Los get sirven para leer el valor de un atributo, nunca llevan parámetros y el nombre delmétodo es la palabra *get* seguida del nombre de la propiedad (por ejemplo, *getEdad*).

Los *set* permiten variar el valor de una propiedad. Por lo que como parámetro reciben el nuevo valor y son métodos siempre de tipo void.
Hacer métodos get y set para acceder a los atributos de la clase es una buena práctica de programación, ya que se permite la encapsulación u ocultación de la información. De tal manera que sólo se tenga acceso a los atributos de la clase a través de los métodos. Con esta práctica de programación nos evitamos problemas, si el usuario de nuestro programa intenta poner valores no permitidos a los atributos de nuestra clase.

Ejemplo:

```
public class
Persona {
private String
nombre;
private int
edad;

public String getNombre() {
        return nombre;
}

public void
        setNombre(String
        nombre) {
        this.nombre=nomb
        re;
}
public int getEdad() {
        return edad;
}
public void
        setEdad(int
        edad) {
        this.edad=e
        dad;
}
}
```

4.4 Sobrecarga de métodos.

Pueden declararse métodos con el mismo nombre en la misma clase, siempre y cuando tengan distintos conjuntos de parámetros (determinados en base al número, tipos y orden de los parámetros). A esto se le conoce como sobrecarga de métodos. Cuando se hace una llamada a un método sobrecargado, el compilador de Java selecciona el método apropiado mediante un análisis del número, tipos y orden de los argumentos en la llamada. Por lo general, la sobrecarga de métodos se utiliza para crear varios métodos con el mismo nombre que realicen la misma tarea o tareas similares, pero con distintos tipos o distintos números de argumentos. Por ejemplo, los métodos abs, min y max de Math se sobrecargan con cuatro versiones cada uno:

1. Uno con dos parametros double.
2. Uno con dos parametros float.
3. Uno con dos parametros int.
4. Uno con dos parametros long.

4.5 Métodos static, campos static y clase Math.

En este apartado se va a ver cuándo un método o atributo debe ser estático y cuándo no. Cuando un método o atributo se define como static quiere decir que se va a crear para esa clase solo una instancia de ese método o atributo. En el siguiente ejemplo se ve como se ha creado un atributo numpajaros que contará el número de pájaros que se van generando. Si ese atributo no fuera estático sería imposible contar los pájaros, puesto que en cada instancia del objeto se crearía una variable numpajaros. De la misma manera, el método nuvopajaro(), muestrapajaros() o el método main tienen sentido que sean estáticos.

Ejemplo:

```
class Pajaro
{
private static int
numpajaros=0;
private char color;
private int edad;

        static void
        nuevopajaro() {
```

```
        numpajaros++;
        }

        Pajaro(
        ) {
        color=
        'v';
        edad=
        0;
        nuevo
        pajaro
        ();
        }
        Pajaro(char c, int e) {color=c; edad=e; nuevopajaro();}
        static void muestrapajaros()
        {System.out.println(numpajaros);}public
        static void main (String[] args) {

        pajaro
        p1, p2;
        p1=new
        pajaro();
        p2=new
        pajaro('a',
        3);
        p1.muestra
        pajaros();
        p2.muestra
        pajaros();


        }
}
```

El atributo numpajaros y los métodos nuevopajaro(), muestrapajaros() y main() se comparten por todos los objetos creados de la clase Pajaro.

Es frecuente que las clases contengan métodos static convenientes para realizar tareas comunes. Por ejemplo, utilizamos el método static pow de la clase Math para elevar un valor a una potencia.
Para declarar un método como static, se coloca la palabra clave static antes del tipo de valor de retorno en la declaración del método. Se puede llamar a cualquier método static especificando el nombre de la clase en la que está declarado el método, seguido de un punto (.) y del nombre del método, como sigue: NombreClase.nombreMétodo (argumentos);

Aquí, utilizaremos varios métodos de la clase Math para presentar el concepto de los métodos static. La clase Math cuenta con una colección de métodos que nos permiten realizar cálculos matemáticos comunes. Por ejemplo, podemos calcular la raíz cuadrada de 900.0 con una llamada al siguiente método static:

```
Math.sqrt( 900.0 )
```

La expresión anterior se evalúa como 30.0. El método sqrt recibe un argumento de tipo double y devuelve un resultado del mismo tipo. Para imprimir el valor de la llamada anterior al método en una ventana de comandos, podríamos escribir la siguiente instrucción:

```
System.out.println( Math.sqrt( 900.0 ) );
```

En esta instrucción, el valor que devuelve sqrt se convierte en el argumento para el método println. Observe que no hubo necesidad de crear un objeto Math antes de llamar al método sqrt. Observe también que todos los métodos de la clase Math son static; por lo tanto, cada uno se llama anteponiendo al nombre del método el nombre de la clase Math y el separador punto (.).
La clase Math también declara dos campos que representan unas constantes matemáticas de uso común: Math.PI y Math.E. Estos campos se declaran en la clase Math con los modificadores public, final y static. Al hacerlos public, otros programadores pueden utilizar estos campos en sus propias clases. Cualquier campo declarado con la palabra clave final es constante; su valor no puede modificarse después de inicializar el campo. Tanto PI como E se declaran como final, ya que sus valores nunca cambian. Al hacer a estos campos static, se puede acceder a ellos mediante el nombre de clase Math y un separador de punto (.), justo igual que los métodos de la clase Math.

5. Referencias a los miembros del objeto actual mediante this.

Cada objeto puede acceder a una referencia a si mismo mediante la palabra clave this (también conocida como referencia this). Se utiliza dentro del código de las clases para obtener una referencia al objeto actual y permitir evitar ambigüedad y realizar llamadas a métodos que de otra forma serían complicadas.

```
public class Punto {
int posX;
int posY;

void modificarCoords(int posX, int posY){

/* Hay ambigüedad ya que posX es el nombre de uno de

* los parámetros, y además el nombre de una de las
* propiedades de la clase Punto
```

```
*/
        this.posX=posX; //this permite evitar la ambigüedad
        this.posY=posY;
}
}
```

En definitiva, los posibles usos de this son:

- this. Referencia al objeto actual. Se usa por ejemplo pasarle como parámetro a un método cuando es llamado desde la propia clase.
- this.atributo. Para acceder a una propiedad del objeto actual.
- this.método(parámetros). Permite llamar a un método del objeto actual con los parámetros indicados.
- this(parámetros). Permite llamar a un constructor del objeto actual. Esta llamada sólo puede ser empleada en la primera línea de un constructor.

En realidad, cuando desde un método invocamos a un método de la propia clase o a una propiedad de la misma, se usa this de forma implícita, es decir que, aunque no escribamos this, el compilador lo sobreentiende. Por eso en la práctica sólo se indica si es imprescindible.

6. Modificadores de acceso.

Se trata de una palabra que antecede a la declaración de una clase, método o propiedad de clase. Hay tres posibilidades: public, protected y private. Una cuarta posibilidad es no utilizar ninguna de estas tres palabras; entonces se dice que se ha utilizado el modificador por defecto (friendly).

Los especificadores determinan el alcance de la visibilidad del elemento al que se refieren. Referidos por ejemplo a un método, pueden hacer que el método sea visible sólo para la clase que lo utiliza (private), para éstas y las heredadas (protected), para todas las clases del mismo paquete (friendly) o para cualquier clase del tipo que sea (public).

En la siguiente tabla se puede observar la visibilidad de cada especificador:

zona	private (privado)	sin modificador (*friendly*)	protected (protegido)	public (público)
Misma clase	X	X	X	X
Subclase en el mismo paquete		X	X	X
Clase (no subclase) en el mismo paquete		X		X
Subclase en otro paquete			X	X
No subclase en otro paquete				X

Ejemplo:

En el siguiente código se declara el atributo euros con el modificador private.

```
/**
* Ejemplo de declaración de la clase PrecioPrivado
* double da() --> devuelve el valor almacenado en euros
* void pone( double x ) --> almacena valor en euros
* euros --> Atributo de acceso privado
*/

public class PrecioPrivado {

// Atributo o variable miembro
private double euros;


// Métodos publicos
public double da() {
return euros;
}
public void pone(double x) {
euros=x;
}
}
```

Si se construye otro código que intente utilizar directamente el atributo euros se producirá un error de compilación ya que el atributo euros sólo es accesible a través de los métodos de la clase da y pone.:

```
/**
* Ejemplo de uso de la clase PrecioPrivado
* double da() --> devuelve el valor almacenado en euros
* void pone( double x ) --> almacena valor en euros
* euros --> Atributo de acceso privado
*/
public class
PruebaPrecioPrivado {
public static void main
(String [] args ) {
        precioPrivado p = new precioPrivado();
        // Crea instanciap.pone(56.8); // Asigna
        56.8 a euros System.out.println("Valor =
        " + p.da());
        p.euros=75.6; // Asigna 75.6 a euros -
        ERROR System.out.println("Valor = " +
        p.euros); // Tambien ERROR
}
}
```

La utilización del modificador private sirve para implementar una de las características de la programación orientada a objetos: el ocultamiento de la información o encapsulación.

Estrictamente hablando, la declaración como público de un atributo de una clase no respeta este principio de ocultación de información. Declarándolos como privados, no se tiene acceso directo a los atributos del objeto fuera del código de la clase correspondiente y sólo puede accederse a ellos de forma indirecta a través de los métodos proporcionados por la propia clase. Una de las ventajas prácticas de obligar al empleo de un método para modificar el valor de un atributo es asegurar la consistencia de la operación. Por ejemplo, un método que asigne valor al atributo euros de un objeto de la clase Precio puede garantizar que no se le asignará un valor negativo.

EN LAS CLASES

Cuando se definen los datos de una determinada clase, se debe indicar el tipo de propiedad que es (String, int, double, int[][],...) y el especificador de acceso (public, private,...).

Ejemplo:

```
public class Persona {
public String nombre;//Se puede acceder desde cualquier clase
private int contraseña;//Sólo se puede acceder desde la
//clase Persona
protected String dirección;//Acceden a esta propiedad
//esta clase y sus descendientes
}
```

Propiedades finales

Los atributos de una clase pueden utilizar el modificador final, para que se conviertan en valores no modificables en el objeto. De ser así, se debe iniciar el valor del atributo en la construcción del objeto.

7. Librerías de objetos: paquetes.

Un paquete o package es un conjunto de clases relacionadas entre sí, las cuales están ordenadas de forma arbitraria. Las clases que forman parte de un paquete no derivan todas ellas de una misma superclase. Por ejemplo, el paquete java.io agrupa las clases que permiten a un programa realizar la entrada y salida de información. Un paquete también puede contener a otros paquetes. Con el uso de paquetes se evitan conflictos, como llamar dos clases con el mismo nombre (si existen, estarán cada una en paquetes diferentes). Al estar en el mismo paquete, las clases de dichos paquetes tendrán un acceso privilegiado a los miembros de dato y métodos de otras clases del mismo paquete.

La sentencia import:

Imaginemos que queremos utilizar una clase contenida en algún paquete. La forma de utilizar dicha clase es generalmente utilizando la sentencia import. Podemos importar una clase individual, como, por ejemplo:

import java.lang.System; //Se importa la clase System

O bien podemos importar todas las clases de un paquete:

import java.awt.*;

En este caso podremos utilizar todas las clases del paquete. Un ejemplo de esto sería el siguiente:

import java.awt.*;
….
Frame fr= new Frame("Panel ejemplo");

También es posible utilizar la clase sin utilizar la sentencia import:

java.awt.Frame fr= new java.awt.Frame("Panel ejemplo");

En la siguiente tabla se muestran las librerías más importantes de Java:

Paquete o librería	**Descripción**
java.io	Librería de Entrada/Salida. Permite la comunicación del programa con ficheros y periféricos.
java.lang	Paquete con clases esenciales de Java. No hace falta ejecutar la sentencia importpara utilizar sus clases. Librería por defecto.
java.util	Librería con clases de utilidad general para el programador.
java.applet	Librería para desarrollar applets.
java.awt	Librerías con componentes para el desarrollo de interfaces de usuario.
java.swing	Librerías con componentes para el desarrollo de interfaces de usuario. Similar al paquete awt.
java.net	En combinación con la librería java.io, va a permitir crear aplicaciones que realicen comunicaciones con la red local e Internet.
java.math	Librería con todo tipo de utilidades matemáticas.
java.sql	Librería especializada en el manejo y comunicación con bases de datos.
java.security	Librería que implementa mecanismos de seguridad.
java.rmi	Paquete que permite el acceso a objetos situados en otros equipos (objetos remotos).
java.beans	Librería que permite la creación y manejo de componentes javabeans. Javabean es un componente que encapsula varios objetos en uno solo.

EJEMPLO:

Crea una clase llamada Pajaro, con los atributos nombre, posX y posY los métodos que realizan las siguientes acciones:

Pajaro(). Constructor por defecto. En este caso, el constructor por defecto no contiene ninguna instrucción, ya que Java inicializa de forma automática las variables miembro, si no le damos ningún valor.

Pajaro(String nombre, int posX, int posY). Constructor que recibe como argumentos una cadena de texto y dos enteros para inicializar el valor de los atributos.

volar(int posX, int posY). Método que recibe como argumentos dos enteros: posX y posY, y devuelve un valor de tipo double como resultado, usando la palabra clave return. El valor devuelto es el resultado de aplicar la fórmula: desplazamiento= √(posX*posX+posY*posY)

Diseña un programa que utilice la clase Pajaro, cree una instancia de dicha clase y ejecute sus métodos.

CLASE PAJARO

```
package ejemplo;
public class Pajaro {
//Declaración de los atributos
    String nombre;
   int posX, posY;
// Creación de métodos constructores
    public Pajaro() {  //constructor por defecto
   }
  public Pajaro(String nombre, int posX, int posY) {
  //constructor con argumentos
        this.nombre=nombre;
        this.posX=posX;
        this.posY=posY;
        //referencia this     para    utilizar variables ya declaradas
   }
 // Creación de métodos
   double volar (int posX, int posY) {
        double desplazamiento = Math.sqrt( posX*posX + posY*posY );
       this.posX = posX;
       this.posY = posY;
       return
       desplazamiento;
   }
}
```

CLASE PRINCIPAL

```
package ejemplo;
public class Principal {
   public static void main(String[] args) {
// Creación de un objeto
      Pajaro loro = new Pajaro("Lucy",50,50) ;
// Aplicación de un método al objeto
      creadodouble d = loro.volar(50,50);
      System.out.println("El desplazamiento ha sido " + d);
   }}
```