Tema 5 – Programación Orientada a Objetos (POO)

1º Ciclo Formativo Grado Superior - Desarrollo de Aplicaciones Web

Módulo: Programación

Prof. José de la Torre López (adaptación Antonio Hernández)

ÍNDICE

Introducción a POO

- A Java se le considera un lenguaje totalmente orientado a objetos.
- La Programación Orientada a Objetos (**POO**) es una técnica de programar aplicaciones ideada en los años setenta y que ha triunfado desde los ochenta, de modo que actualmente es el método habitual de creación de aplicaciones.
- Existen muchos otros tipos de paradigmas de programación.

Ejercicio en clase: Buscad información sobre otros paradigmas y algún lenguaje de programación que lo represente.

Programación estructurada

La **programación estructurada** impone una forma de escribir código que potencia la legibilidad del mismo. Cuando un problema era muy grande, aún que el código es legible, ocupa tantas líneas que al final le hacen inmanejable.

```
DEFINT I-N          'Declara entera toda variable que comience con letras I
iTrue = -1          'Flag en Verdadero
INPUT "¿Cuál es su nombre"; NombreUsuario$
PRINT "Bievenido al 'asterisquero',"; NombreUsuario$
DO
   PRINT
   INPUT "¿Con cuántos asteriscos inicia [Cero sale]:"; NroAsteriscos
   IF NroAsteriscos<=0 THEN EXIT DO
   Asteriscos$ = ""
   FOR I=1 TO NroAsteriscos
      Asteriscos$=Asteriscos$ + "*"
   NEXT I
   PRINT "AQUI ESTAN: "; Asteriscos$
   DO
      INPUT "Desea más asteriscos:";SN$
   LOOP UNTIL SN$<>""
   IF SN$<>"S" AND SN$<>"s" THEN EXIT DO      'Salida
   INPUT "CUANTAS VECES DESEA REPETIRLOS [Cero sale]:";iVeces
   IF iVeces<=0 THEN EXIT DO    'Salida
   FOR I = 1 TO iVeces
      PRINT Asteriscos$;
   NEXT I
   PRINT
LOOP WHILE iTrue
END
```

Programación modular

- La **programación modular** supuso un importante cambio, ya que el problema se descompone en módulos (en muchos lenguajes llamados funciones) de forma que cada uno se ocupa de una parte del problema.

```
01:  #include <stdio.h>
02:  #include <stdlib.h>
03:
04:  long i, n, iCount;
05:  long aiData[1000];
06:
07:  void insertionsort(long aiArray[], long iSize) {
08:     long i, j, iTemp;
09:
10:     for (i= 0; i< iSize; i++)
11:        for (j= i; j> 0; j--)
12:           if (aiArray[j-1]> aiArray[j]) {
13:              iTemp= aiArray[j-1];
14:              aiArray[j-1]= aiArray[j];
15:              aiArray[j]= iTemp;
16:              iCount++;
17:           }
18:           else break;
19:  }
20:
21:  int main(void) {
22:     while (scanf("%ld", &n)!= EOF) {
23:        for (i= 0; i< n; i++)
24:           scanf("%ld", &aiData[i]);
25:        iCount= 0;
26:        insertionsort(aiData, n);
27:        printf("Minimum exchange operations : %ld\n",iCount);
28:     }
29:
30:     return (0);
31:  }
```

Desventajas de la programación modular

- La separación de conceptos no es sencilla
- Los datos globales al final deben ser compartidos creando problemas de exclusión mutua
- La programación modular requiere más memoria y tiempo de ejecución
- No se dispone de algoritmos formales de modularidad, por lo que a veces los programadores no tienen claras las ideas de los módulos

Surge la Programación Orientada a Objetos

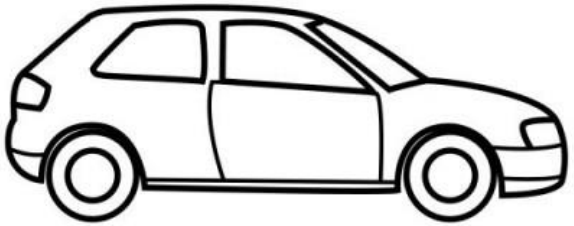
- Con la POO se intenta solucionar esta limitación ya que el problema se dividen en **objetos.**

- Un **objeto** es un elemento del programa que integra sus propios datos y su propio funcionamiento. Es decir un objeto está formado por datos (**atributos**) y por las funciones que es capaz de realizar el objeto (**métodos**).

- Una **clase** es lo que define a un tipo de objeto. Al definir una clase lo que se hace es indicar cómo funciona un determinado tipo de objetos. Luego, a partir de la clase, podremos crear objetos de esa clase.

Ejemplo que todos entendemos

- Una clase podría ser la clase **coche**, que representa un automóvil. Cuando se defina esta clases indicaremos las propiedades (como el color, modelo, marca, velocidad máxima,...) y los métodos (arrancar, parar, repostar,...). Todos los coches (es decir todos los objetos de clase coche) tendrán esas propiedades y esos métodos.



CLASE

OBJETOS

Propiedades de la POO (I)

- **Encapsulamiento**. Indica el hecho de que los objetos encapsulan datos y métodos. Una clase se compone tanto de variables (**propiedades**) como de funciones y procedimientos (**métodos**). De hecho no se pueden definir variables (ni funciones) fuera de una clase (es decir, en los programas orientados a objetos no hay **variables globales**).
- **Ocultación**. Durante la creación de las clases de objetos, hay métodos y propiedades que se crean de forma **privada**. Es decir, hay métodos y propiedades que sólo son visibles desde la propia clase, pero que no son accesibles desde otras clases. Cuando la clase está ya creada y compilada, esta zona privada permanece oculta al resto de clases. De esta forma se garantiza la independencia entre clases.

Propiedades de la POO (II)

- **Polimorfismo**. Cada método de una clase puede tener varias definiciones distintas. En el caso del parchís: ***partida.empezar(4)*** empieza una partida para cuatro jugadores, ***partida.empezar(rojo, azul***) empieza una partida de dos jugadores para los colores rojo y azul; estas son dos formas distintas de emplear el método ***empezar***, por lo tanto este método es polimórfico. Esto simplifica la programación y reutilización de clases.
- **Herencia**. Mediante la POO podemos definir clases que utilicen métodos y propiedades de otras (que hereden dichos métodos y propiedades). De esta forma podemos establecer organizaciones jerárquicas de objetos. la herencia se trata en la unidad siguiente.

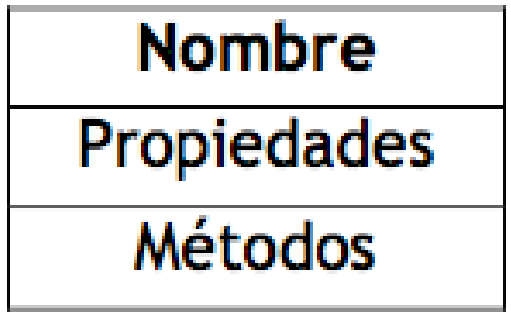
Introducción al concepto de objeto

- Un objeto es cualquier entidad representable en un programa informático, bien sea real (***ordenador***) o bien sea un concepto (***transferencia***). Un objeto en un sistema posee: una **identidad**, un **estado** y un **comportamiento**.
  - El **estado** 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 cómo 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).
  - La **identidad** es la propiedad que determina que cada objeto es único aunque tenga el mismo estado. No existen dos objetos iguales. Lo que sí existen son dos **referencias** al mismo objeto

Clases

- Una clase tiene:
  - Nombre
    - Evitar abreviaturas
    - Evitar nombres largos
    - Evitar nombres ya existentes o palabras reservadas
  - Atributos o propiedades (características)
  - Métodos (qué sabe hacer)
  - Constructor/es(qué ocurre al crearse)

Clases (diseño)



Ejercicio en clase

Diseñar las siguientes clases, escribiendo los atributos y métodos que creas convenientes:

- Coche
- Motocicleta
- Perro
- Transferencia bancaria

Clases en Java

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

Clases en Java - Ejemplo

Noria
radio:double
girar(velocidad:int) parar()

```
class Noria {
    double radio;
    void girar(int velocidad){
            ...//definición del método
    }
    void parar(){...
}
```

Ejercicio en clase

En un nuevo proyecto Java crear las clases que hemos modelado antes.

Objetos

- Los objetos son entidades en sí de la clase. Un objeto se crea utilizando el llamado **constructor** de la clase. El constructor es el método que permite iniciar el objeto.

- El objeto realmente es la información que se guarda en memoria, acceder a esa información es posible realmente gracias a una referencia al objeto.

Creación de objetos

Clase objeto = new Clase();

E.g.:
Noria miNoria = new Noria();

Acceso a los atributos

Objeto.propiedad;

E.g.:
miNoria.radio;

Acceso a los métodos

Objeto.método(parámetros);

E.g.:
miNoria.gira(5);

Ejercicio en clase

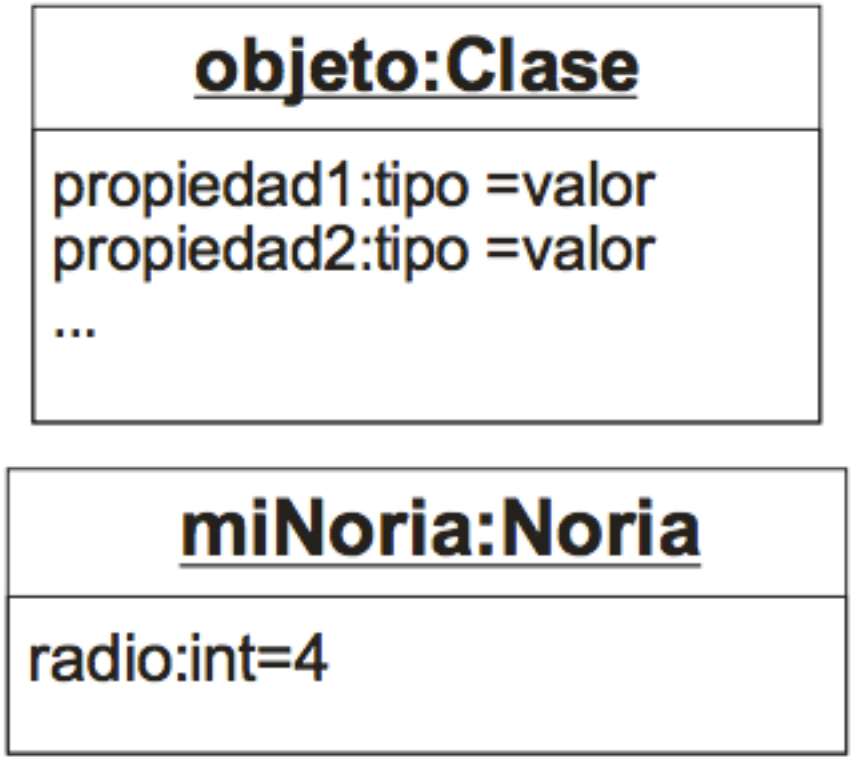
Crear un nuevo proyecto Java con un main e instanciar diferentes objetos de los que hemos creado sus clases antes. Se pide además cambiar los valores de los distintos objetos.

Herencia

- En la POO tiene mucha importancia este concepto, la herencia es el mecanismo que permite crear clases basadas en otras existentes. Se dice que esas clases *descienden* de las primeras. Así por ejemplo, se podría crear una clase llamada **vehículo** cuyos métodos serían mover, *parar*, *acelerar* y *frenar*. Y después se podría crear una clase **coche** basada en la anterior que tendría esos mismos métodos (les heredaría) y además añadiría algunos propios, por ejemplo ***abrirCapó*** o ***cambiarRueda***.

*La herencia se trata en profundidad en el tema siguiente.

Representación de objetos en UML



Asignación de distintos objetos

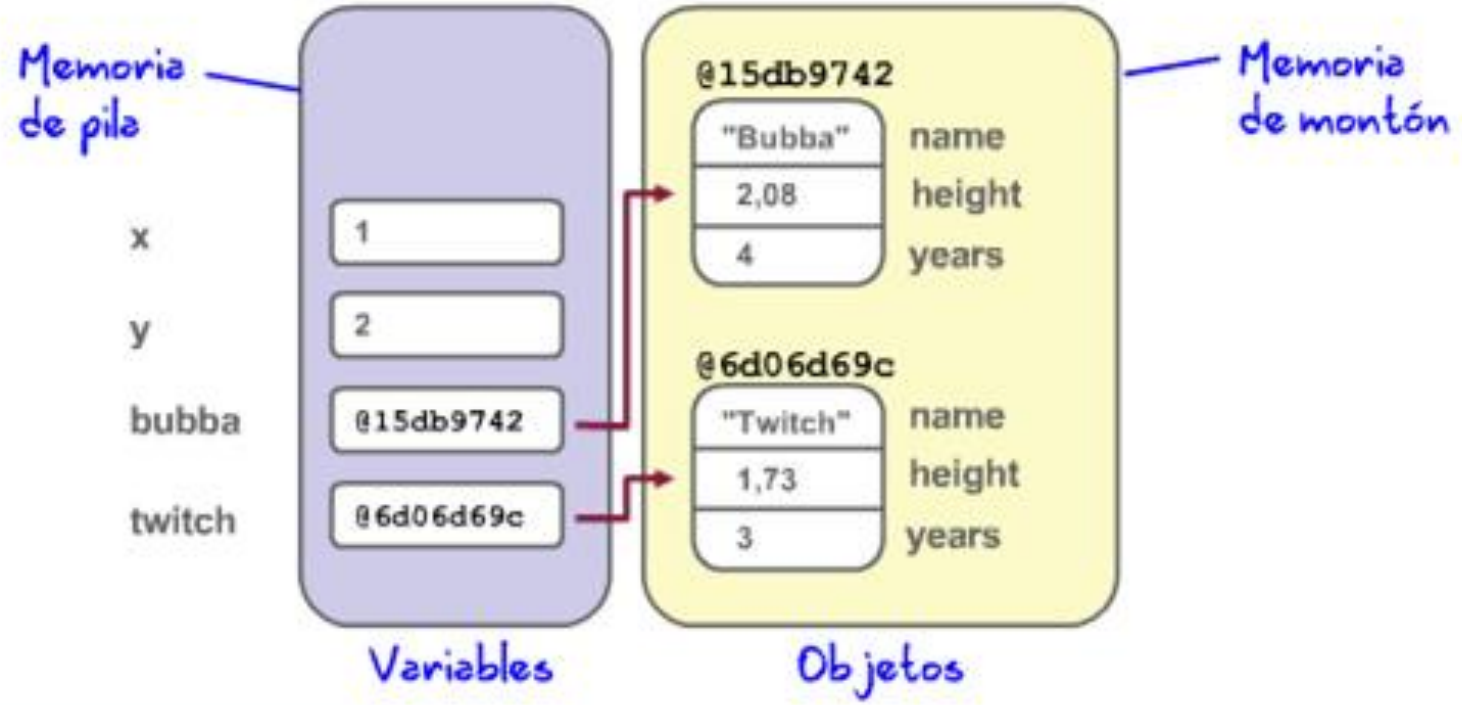
```
Noria n1=new Noria();
Noria n2;
n1.radio=9;
n2=n1;
n2.radio=7;
System.out.println(n1.radio); //sale 7
```

Tipos de memoria

- Stack (Pila)
  - Variables locales
  - Tipos de datos primitivos
  - parámetros y valores de retorno,
  - para llevar el control de la invocación y retorno de los métodos
  - Referencias a los objetos del heap
- Heap (Montículo o montón)
  - Objetos en sí
  - sus variables de instancia.

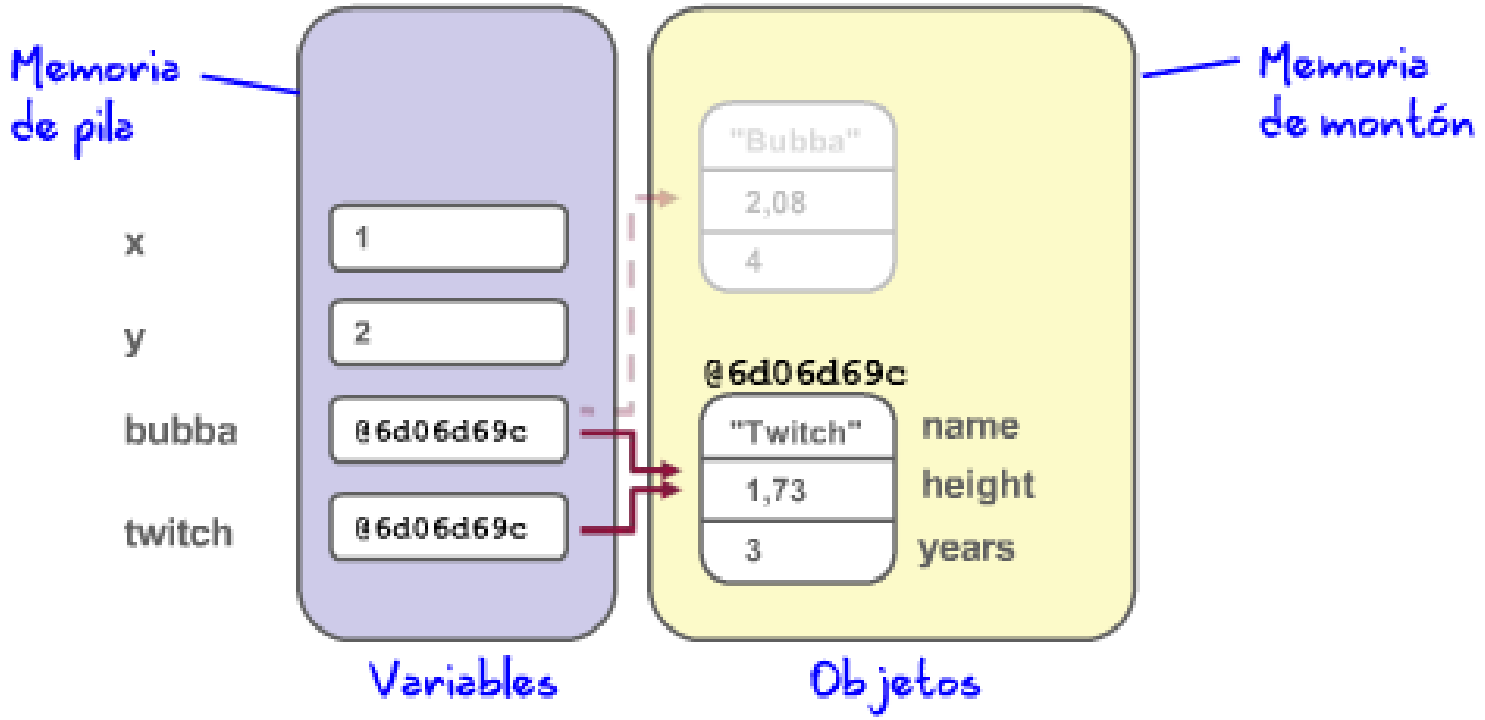
Referencias a variables

```
int x = 1;
int y = 2;
Prisoner bubba = new Prisoner();
Prisoner twitch = new Prisoner();
...
```



Referencias a variables

```
bubba = twitch;
```



Referencias a variables

- A partir de la línea 14, `bubba` y `twitch` hacen referencia al mismo objeto.
- Cualquier variable de referencia se podría utilizar para acceder a los mismos datos.

```
11 Prisoner bubba = new Prisoner();
12 Prisoner twitch = new Prisoner();
13
14 bubba = twitch;
15
16 bubba.name = "Bubba";
17 twitch.name = "Twitch";
19
20 System.out.println(bubba.name);        //Twitch
21 System.out.println(bubba == twitch); //true
```

Referencias a variables

- Las primitivas siempre son variables independientes.
- Los valores primitivos siempre ocupan ubicaciones distintas en la memoria de pila.
- La línea 14 hace brevemente que los valores primitivos `x` e `y` sean iguales.

```
11 int x;
12 int y;
13
14 x = y;
15
16 x = 1;
17 y = 2;
19
20 System.out.println(x);              //1
21 System.out.println(x == y);         //false
```

Ejercicio en clase

- Probar a:
  1. instanciar dos objetos (x e y) de la misma clase,
  2. asignar valores a un atributo de cada objeto,
  3. Imprimir por pantalla los valores de los atributos
  4. Asignar un objeto al otro, x = y
  5. Imprimir por pantalla de nuevo los valores de los atributos
  6. Hacer un cambio en y
  7. Imprimir en x para ver si el cambio en y ha surtido efecto

Modificadores de acceso

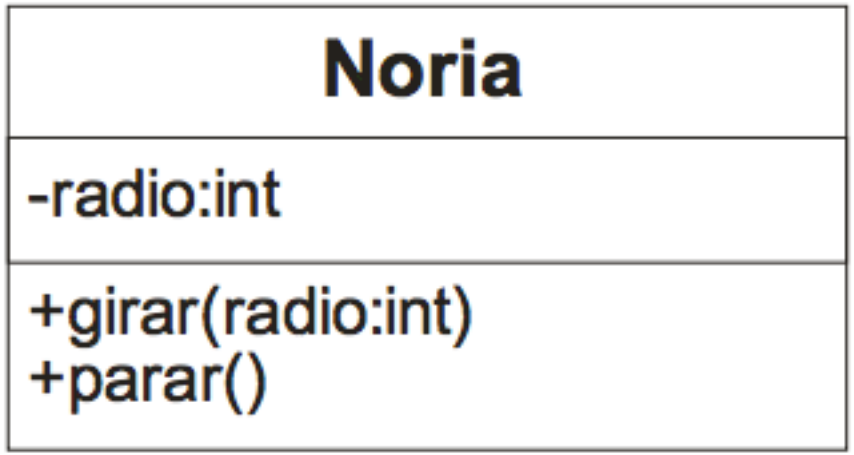
zona	private (privado)	sin modificador (*friendly*)	protected (protegido)	public (público)
Misma clase	X	X	X	X
Subclase en el mismo paquete		X	X	X
Clase (no subclase) en el mismo paquete		X		X
Subclase en otro paquete			X	X
No subclase en otro paquete				X

Modificadores de acceso (otra forma de verlo)

Visibilidad	Significado	Java	UML
Pública	Se puede acceder al miembro de la clase desde cualquier lugar.	`public`	+
Protegida	Sólo se puede acceder al miembro de la clase desde la propia clase o desde una clase que herede de ella.	`protected`	#
Por defecto	**Se puede acceder a los miembros de una clase desde cualquier clase en el mismo paquete**		~
Privada	Sólo se puede acceder al miembro de la clase desde la propia clase.	`private`	-

Modificadores de acceso

```
public class Noria {
    private double radio;
    public void girar(int velocidad){
            ...//definición del método
    }
    public void parar(){...
}
```



La importancia de la visibilidad

```
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
```

Ejercicio en clase

- Probar a crear un atributo con cada modificador de acceso, instanciar un objeto en el main e intentar acceder a los atributos.

- Hacer lo mismo con métodos.

Definiendo los comportamientos

1. Sus especificadores de alcance o visibilidad
2. El tipo de datos o de objeto que devuelve
3. El identificador del método
4. Los parámetros
5. El cuerpo del método
6. El resultado (si hay) con un return

Ejemplo de clase

```
public class Vehiculo {
   public int ruedas;
   private double velocidad=0;
   String nombre;
    public void acelerar(double cantidad) {
      velocidad += cantidad;
   }
    public void  frenar(double cantidad) {
      velocidad -= cantidad;
   }
   public double obtenerVelocidad(){
       return velocidad;
    }
    public static void main(String args[]){
         Vehiculo miCoche = new Vehiculo();
         miCoche.acelerar(12);
         miCoche.frenar(5);
         System.out.println(miCoche.obtenerVelocidad());
    } // Da 7.0
}
```

¿Y para qué los private? Getters y setters

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

    public String getNombre() {
            return nombre;
    }
    public void setNombre(String nombre) {
            this.nombre=nombre;
    }

    public int getEdad() {
            return edad;
    }
    public void setEdad(int edad) {
            this.edad=edad;
    }
}
```

Ventajas de usar Getters y setters (I)

- Encapsulación del comportamiento asociado con la obtención o configuración de la propiedad: esto permite que se agreguen más fácilmente funcionalidades adicionales (como la validación) más adelante.

- Aislar su interfaz pública del cambio, permitiendo que la interfaz pública permanezca constante mientras la implementación cambia sin afectar a los consumidores existentes.

Ventajas de usar Getters y setters (II)

- Proporcionar un punto de intercepción de depuración para cuando una propiedad cambia en tiempo de ejecución: depurar cuando y donde una propiedad cambió a un valor particular puede ser bastante difícil sin esto en algunos idiomas.
- Pueden permitir diferentes niveles de acceso, por ejemplo, la obtención puede ser pública, pero la modificación podría estar protegida
- ….

Sobrecarga

```
public class Matemáticas{
    public double suma(double x, double y) {
            return x+y;
    }
    public double suma(double x, double y, double z){
            return x+y+z;
    }
    public double suma(double[] array){
            double total =0;
            for(int i=0; i<array.length;i++){
                    total+=array[i];
            }
            return total;
    }
```

El método suma es **polimórfico**

La palabra mágica this

```
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;
    }
}
```

La palabra mágica this

```
public class Punto {
    int posX;
    int posY;
    ...
    /**Suma las coordenadas de otro punto*/
    public void suma(Punto punto2){
            posX = punto2.posX;
            posY = punto2.posY;
    }

    /** Dobla el valor de las coordenadas del punto*/
    public void dobla(){
            suma(this);
    }
```

Usos de this

- **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 .

Constructores

```
public class Ficha {
    private int casilla;

    public Ficha() { //constructor
            casilla = 1;
    }

    public void avanzar(int n) {
            casilla += n;
    }
    public int casillaActual(){
            return casilla;
    }
}

public class App {
    public static void main(String[] args) {
            Ficha ficha1 = new Ficha();
            ficha1.avanzar(3);
            System.out.println(ficha1.casillaActual());//Da 4
    }
}
```

Constructores

```
public class Ficha {
    private int casilla; //Valor inicial de la propiedad
    public Ficha(int n) { //constructor
            casilla = n;
    }
    public void avanzar(int n) {
            casilla += n;
    }
    public int casillaActual(){
            return casilla;
    }
}
public class App {
    public static void main(String[] args) {
            Ficha ficha1 = new Ficha(6);
            ficha1.avanzar(3);
            System.out.println(ficha1.casillaActual());//Da 9
    }
}
```

Ejercicio en clase

- Para las clases que hemos creado, implementar varios constructores y utilizarlos.

static

- Puede haber propiedades y métodos comunes a todos los objetos de una clase. En ese caso se consideran propiedades y métodos de la clase y no de los objetos. Comúnmente se les conoce como métodos y propiedades estáticas.

- Por ejemplo si todos los empleados tuvieran el mismo salario mínimo entonces se definiría esa propiedad como estática. Y para manipularla se utilizaría el nombre de la clase y no de los objetos.

static

Algunas consideraciones:

- Los atributos estáticos valen lo mismo para CUALQUIER instancia u objeto de una clase.
- Los métodos y atributos estáticos pueden ser llamados sin necesidad de instanciarlos.
  E.g.: Math

¿Por qué main es estático?

Arrays de objetos

1º) se crea el array solo con el número de objetos (como para tipos primitivos)

2º) se inicia cada elemento del array llamando al constructor correspondiente

```
//Creamos un array de objetos de la clase empleados
Empleado arrayObjetos[]=new Empleado[3];

//Creamos objetos en cada posicion
arrayObjetos[0]=new Empleado("Fernando", "Ureña", 23, 1000);
arrayObjetos[1]=new Empleado("Epi", "Dermis", 30, 1500);
arrayObjetos[2]=new Empleado("Blas", "Femia", 25, 1200);
```

Arrays de objetos

```
public class EmpleadoApp {

    public static void main(String[] args) {

        //Creamos un array de objetos de la clase empleados
        Empleado arrayObjetos[]=new Empleado[10];

        //Creamos 10 empleados por defecto
        int suma=0;
        for (int i=0;i<arrayObjetos.length;i++){
            arrayObjetos[i]=new Empleado();
        }

        //Lo recorremos y sumamos de nuevo los salarios (600*10)
        for (int i=0;i<arrayObjetos.length;i++){
            //Mostramos la direccion del objeto
            System.out.println(arrayObjetos[i]);
            suma+=arrayObjetos[i].getSalario();
        }
        System.out.println("La suma de salarios es "+suma);
    }

}
```

se puede iniciar cada elemento del array llamando al constructor usando un bucle