

NOTAS – REPASO – EDITS:   
[UT1, UT3 Generación de números aleatorios] status: updated  
Falta: [UT4 Programación Orientada a Objetos, final]

# UT4 Programación Orientada a Objetos

**1. Programación Orientada a Objeto (POO)**

La programación Orientada a Objetos es una metodología que basa la estructura de los programas en torno a los objetos.

Los lenguajes de POO ofrecen medios y herramientas para describir los objetos manipulados por un programa. Más que describir cada objeto individualmente, estos lenguajes proveen una construcción (Clase) que describe a un conjunto de objetos que poseen las mismas propiedades.

**2. Clase.**

Una clase es un tipo definido por el usuario (plantilla) que describe los atributos y los métodos de los objetos que se crearán a partir de ella.

El estado de un objeto viene determinado por sus **atributos,** y su comportamiento está definido por sus **métodos.**

Dentro de las clases también se encuentran los **constructores** que permiten inicializar un objeto.

Los atributos y los métodos se denominan en general **miembros de la clase**.

La definición de una clase consta de dos partes: el nombre de la clase precedido por la palabra reservada **class** y el cuerpo de la clase entre llaves. La sintaxis queda:

|  |
| --- |
| **class nombre-clase {** |
| **cuerpo de la clase** |
| **}** |

Dentro del cuerpo de la clase se puede encontrar los atributos y métodos.

**Ej:**

class Circunferencia {

private double x, y, radio;

public Circunferencia(){ }

public Circunferencia(double cx, double cy, double r){

x=cx;

y=cy;

radio=r;

}

public void ponRadio(double r){

radio=r;

}

public double longitud(){

//No hay que pasarle el radio, coge el radio de la circunf

return 2\*Math.PI\*radio;

}

}

En el ejemplo, se define la **clase Circunferencia**, que puede ser usada dentro de un programa de la misma manera que cualquier otro tipo. Un objeto de esta clase tiene tres atributos (coordenadas del centro y valor del radio), dos constructores y dos métodos.

**Los constructores se distinguen fácilmente porque tienen el mismo nombre que la clase.**

Los atributos se declaran de la misma manera que cualquier variable. En una clase, cada atributo debe tener un nombre único.

Siguiendo las recomendaciones de la Programación Orientada a Objetos, cada clase se debe implementar en un fichero .java, de esta manera es más sencillo modificar la clase.

El fichero que contiene la clase debe llevar el nombre de la clase pública con la extensión .java. ***A veces tendremos varias clases en un .java, pero sólo una puede ser pública***

**3. Métodos**

Los métodos forman lo que se denomina interfaz de los objetos, definen las operaciones que se pueden realizar con los atributos. Desde el punto de vista de la Programación Orientada a Objetos, el conjunto de métodos se corresponde con el conjunto de mensajes a los que los objetos de una clase pueden responder.

**Los métodos implementan la funcionalidad asociada al objeto**. Los métodos son el equivalente a las funciones en Programación Estructurada. Se diferencian de ellos en que es posible **acceder** a las **variables de la clase de forma implícita** (atributos).

Cuando se desea realizar una acción sobre un objeto, se dice que se le manda un mensaje invocando a un método que realizará la acción.

Los métodos permiten al programador modularizar sus programas.

Todas las variables declaradas en las definiciones de métodos son variables locales, solo se conocen en el método que las define. Casi todos los métodos tienen una lista de parámetros que permiten comunicar información.

La sintaxis para definir un método es:

|  |
| --- |
| **<modificador-acceso> <modificador > tipoR nombre-método(<parámetros>){** |
| **<cuerpodelmétodo>** |
| **}** |

<modificador-acceso> indica cómo es el acceso a dicho método: public, private, protected y sin modificador.

<modificador > indica si es estática o no.

<tipoR> es el tipo de dato que retorna el método. Es obligatorio indicar un tipo. Para los métodos que no devuelven nada se utiliza la palabra reservada *void*.

Nombre-método es cómo el programador quiere llamar a su método y <parámetros> son los datos que se van a enviar al método para trabajar con ellos, no son obligatorios.

Para devolver el valor se utiliza el operador **return**. Una vez que se ejecute return, el control se devuelve al código que lo llamó y la ejecución del método termina. Los métodos que no lleven return, tendrán tipo void.

**Ej:**

int suma(int x, int y) {

return x+y;

}

En este ejemplo el método recibe 2 parámetros de tipo int, realiza su suma y devuelve esta suma, que es un valor de tipo int.

void imprimir(){

System.out.println(“Este método no devuelve nada y tampoco

recibe parámetros”);

}

En este ejemplo el método no devuelve ningún valor, ni recibe parámetros, aun así es necesario poner los paréntesis vacíos.

**3.1 Métodos Estáticos o de Clase – hay otro word**

Van precedidos del modificador **static**.

Para invocar a un **método** estático no se necesita crear un objeto de la clase en la que se define. Si **se invoca desde la clase** en la que se encuentra definido, basta con escribir su nombre.

Si **se le invoca desde una clase distinta**, debe anteponerse a su nombre, el de la clase en que se encuentra seguido del operador punto (.) La sintaxis es:

**< NombreClase> .metodoEstatico( );**

Suelen emplearse para realizar operaciones comunes a todos los objetos de la clase. No afectan a los estados de los objetos.

Por ejemplo, si se necesita un método para contabilizar el número de objetos creados de una clase, se define estático ya que su función, aumentar el valor de una variable entera, se realiza independientemente del objeto empleado para invocarle. Esa variable entera, sería única para todos los objetos.

Otra razón por la que tendríamos que usar métodos estáticos es si se utilizan fuera del contexto de cualquier instancia

Los métodos estáticos sólo pueden llamar a otros métodos static directamente, y no se pueden referir a **this** o **super** de ninguna manera.

No es conveniente usar muchos métodos estáticos, pues si bien se aumenta la rapidez de ejecución, se pierde flexibilidad, no se hace un uso efectivo de la memoria y no se trabaja según los principios de la POO.

Las **variables** también se pueden declarar como static, y es equivalente a declararlas como variables globales, que son accesibles desde cualquier fragmento de código.

Se puede declarar un bloque static que se ejecuta una sola vez si se necesitan realizar cálculos para inicializar las variables static.

Ej:

class Estatica {

static int a=3,b;

static{

System.out.println(“Bloque static inicializado“);

b=a\*4;

} **// Si no es static no compilará**

static void metodo(int x){

System.out.println(“x= “+x);

System.out.println(“a= “+a);

System.out.println(“b= “+b);

}

public static void main(String[] args){

metodo(42);

}

}

En el ejemplo la clase que tiene dos variables static, un bloque de inicialización static y un método static. La salida del programa es:

Bloque static inicializado:

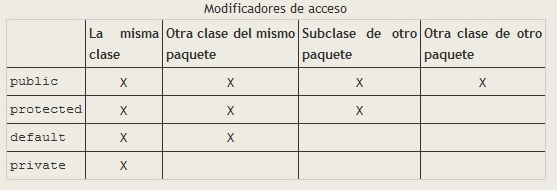
x = 42

a = 3

b = 12

**3.2 Modificadores de acceso**

*Se profundizará sobre los modificadores de acceso cuando veamos Herencia y Paquetes en Java*. De momento, pondremos que un atributo/método es privado (**private**) si solo lo utilizamos dentro de su clase, si lo queremos usar fuera lo pondremos público (**public**).



**4. Objetos**

Es una entidad (tangible o intangible) que posee características y acciones que realiza por sí solo o interactuando con otros objetos.

Un **objeto** es una entidad caracterizada por sus atributos propios y cuyo comportamiento está determinado por las acciones o funciones que pueden modificarlo (métodos), así como también las acciones que requiere de otros objetos.

Un objeto constituye una unidad que **oculta** tanto datos como la descripción de su manipulación. Puede ser definido como una **encapsulación** y una **abstracción**: una encapsulación de atributos y servicios, y una abstracción del mundo real.

**Para el contexto del Enfoque Orientado a Objetos (EOO) un objeto es una entidad que encapsula datos (atributos) y acciones o funciones que los manejan (métodos). También para el EOO un objeto se define como una instancia o particularización de una clase.**

Los objetos de interés durante el desarrollo de software no solo son tomados de la vida real (objetos visibles o tangibles), también pueden ser abstractos. En general, son entidades que juegan un rol bien definido en el dominio del problema. Un libro, una persona, un coche, un polígono son algunos **ejemplos** de objeto.

Cada objeto puede ser considerado como un proveedor de servicios utilizados por otros objetos que son sus clientes. Cada objeto puede ser a la vez proveedor y cliente. De ahí que un programa pueda ser visto como un conjunto de relaciones entre proveedores clientes. Los servicios ofrecidos por los objetos son de dos tipos:

1.- Los datos, que llamamos **atributos**.

2.- Las acciones o funciones, que llamamos **métodos**.

Un objeto consta de una estructura interna (los atributos) y de una interfaz que permite acceder y manipular dicha estructura (los métodos). Para **construir un objeto de una clase** cualquiera hay que llamar a un método de iniciación, **el constructor**. Para ello, se utiliza el operador **new**. La sintaxis es la siguiente:

|  |
| --- |
| **Nombre-clase nombre-objeto = new Nombre-clase (<valores>);** |

Donde **Nombre-clase** es el nombre de la clase de la cual se quiere crear el objeto, **nombre-objeto** es el nombre que el programador quiere dar a ese objeto y <valores> son los valores con los que se inicializa el objeto. Dichos valores son opcionales y aparecen en el constructor.

**Ej:** Circunferencia circ = new Circunferencia();

Se crea el objeto circ, de la clase Circunferencia, con los valores predeterminados.

Cuando se crea un objeto, Java hace lo siguiente:

* **Asignar memoria** al objeto por medio del operador ***new***.
* **Llamar al constructor** de la clase para inicializar los atributos de ese objeto con los valores iniciales o con los valores predeterminados por el sistema: los atributos numéricos a cero, los alfanuméricos a nulos y las referencias a objetos a null.

Si no hay suficiente memoria para ubicar el objeto, el operador new lanza una excepción OutOfMemoryError.

Para acceder desde un método de una clase a un miembro de un objeto de otra clase diferente se utiliza la sintaxis: **objeto.miembro**. Cuando el miembro accedido es un método se entiende que el objeto ha recibido un mensaje, el especificado por el nombre del método y responde ejecutando ese método. En este caso, después del nombre del método siempre se pone paréntesis, aunque no haya parámetros.

***Características Generales:***

* Un objeto posee **estados**. El estado de un objeto está determinado por los valores que poseen sus atributos en un momento dado.
* Un objeto tiene un **conjunto de métodos**. El comportamiento general de los objetos dentro de un sistema se describe o representa mediante sus operaciones o métodos. Los métodos se utilizarán para obtener o cambiar el estado de los objetos, así como para proporcionar un medio de **comunicación** entre objetos.
* Un objeto tiene un **conjunto de atributos**. Los atributos de un objeto contienen valores que determinan el estado del objeto durante su tiempo de vida. Se implementan con variables, constantes y estructuras de datos (similares a los campos de un registro).

**4.1 Asignación**

Una vez el objeto está creado, se tiene una **referencia a ese objeto en la variable** donde se asigna. Si se realiza la asignación de un objeto a otro los dos harán referencia al mismo objeto.

**Ej**:

Circunferencia c1 = new Circunferencia(0, 0, 15);

Circunferencia c2 = c1;

c1.ponRadio(25);

System.out.println(c2.radio); // Es el mismo objeto

En el ejemplo, la variable c1 apunta a un objeto de la clase Circunferencia. Debido a la asignación, la variable c2 apunta al mismo objeto. Después se modifica el valor del radio del objeto apuntado por c1.

Y por último, se visualiza c2 que será el valor 25.

**4.2 Igualdad**

Creamos dos objetos iguales (con los mismos valores de sus variables miembro), al preguntar si las variables que apuntan a esos objetos son iguales nos devolverá false, pues aunque tengan el mismo valor no son el mismo objeto.

**Ej:**

Circunferencia c1 = new Circunferencia(0,0,15);

Circunferencia c2 = new Circunferencia(0,0,15);

if (c1==c2) **/\* Esto es falso\*/**

**4.3 Constructor**

Un Constructor es un método especial en Java empleado para inicializar valores en instancias de objetos. A través de este tipo de métodos es posible generar distintos tipos de instancias para la clase en cuestión.

Los métodos constructores tienen las siguientes **características**:

* Se llaman igual que la clase.
* No devuelven nada, ni siquiera void.
* Puede haber varios constructores, que deberán distinguirse por el tipo de valores que reciban.
* De entre los que existan, solo uno se ejecutará al crear el objeto.
* El código de un constructor, generalmente, suele ser inicializaciones de variables y objetos, para conseguir que el objeto sea creado con dichos valores iniciales.
* Si no se define ningún constructor el compilador crea uno **por defecto** sin parámetros que, al ejecutarse, inicializa el valor de cada atributo de la nueva instancia a 0, false o null, dependiendo de si el atributo es numérico, alfanumérico o una referencia a otro objeto respectivamente, pero dicho constructor desaparece en el mismo momento en que se defina otro constructor, por lo que **si se quiere tener el de por defecto habrá que definirlo**.

La **declaración de constructores** sigue la siguiente sintaxis:

|  |
| --- |
| **<modificador deVisibilidad> NombredelaClase ( <argumentos> ) {** |
| **< declaraciones>** |
| **}** |

Donde <modificadorvisibilidad> es el tipo de modificador de acceso del constructor, <Nombredelaclase> es el nombre del constructor y debe coincidir con el de la clase y <argumentos> son las variables que recibe el constructor y contienen los valores con los que se inicializaran los atributos.

**Ej:**

class Circunferencia{

private double x, y, radio;

public Circunferencia(){ }

public Circunferencia(double cx, double cy, double r){

x=cx; y=cy; radio=r;

}

}

En este ejemplo existen 2 constructores, el primero que es el de por defecto y el segundo que inicializa los atributos x, y y radio con los valores cx, cy y r, respectivamente.

**4.4 Referencia this**

Java incluye un valor de referencia especial llamado **this**, que se utiliza dentro de cualquier método para referirse al objeto actual. **El valor this se refiere al objeto sobre el que ha sido llamado el método actual.**

Se puede utilizar this siempre que se requiera una referencia a un objeto del tipo de una clase actual. Si hay dos objetos que utilicen el mismo código, seleccionados a través de otras instancias, cada uno tiene su propio valor único de this.

Normalmente, dentro del cuerpo de un método de un objeto se puede referir directamente a las variables miembros del objeto. Sin embargo, algunas veces no se querrá tener ambigüedad sobre el nombre de la variable miembro y uno de los argumentos del método que tengan el mismo nombre y usaremos la referencia this.

**Ej:**

class Circunferencia {

private double x, y, radio;

public Circunferencia(double x, double y, double radio){

this.x=x;

this.y=y;

this.radio=radio;

}

}

En el ejemplo, el constructor de la clase inicializa las variables con los argumentos pasados al constructor. Se debe utilizar this en este constructor para evitar la ambigüedad entre los argumentos y los atributos miembro.

También se puede utilizar this para llamar a uno de los métodos del objeto actual. Esto solo es necesario si existe alguna ambigüedad con el nombre del método y se utiliza para intentar hacer el código más claro.

**UT4 Objetos y clases: Introducción**

**El concepto de clase**

Si pensamos en las **clases de objetos** que vemos, tenemos coches, bares, semáforos, personas…

Si simplificamos esta definición de clases de objetos por simplemente clases:

¿Qué **clases** vemos? Coches, bares, semáforos, personas…

Y ya estaremos usando terminología de la **programación orientada a objetos**:

Tenemos la **clase** Coche, la clase Bar, la clase Semáforo, la clase Persona…

No nos importa si esa persona es Marta, está en el bar “Casa Manolo” (Bar Casa Manolo sería un objeto de clase Bar), si tiene un Seat Panda o si el semáforo está en rojo. Estamos pensando en términos **clasificatorios**.

Vamos a pensar ahora en cada una de estas clases:

¿Cómo son los coches? ¿Qué **características** pueden diferenciar uno de otro? Por ejemplo, podríamos indicar la marca, el modelo, el color, la cilindrada, etc.

A las características las vamos a denominar **atributos**, término muy utilizado en la **programación orientada a objetos**.

Atributo: variable dentro del objeto

|  |
| --- |
| ¿Qué atributos tiene la clase Bar? Todo bar tiene un nombre, una ubicación, un estado (si está abierto o cerrado), una lista de precios,… |
| Como atributos de la clase Semáforo podríamos indicar su ubicación y estado (rojo, verde o ámbar). |
| La clase Persona podría definir como atributos el nombre, sexo, edad, estado civil, altura, etc. |

A continuación vamos a pensar en el **comportamiento** de estas clases. Nos preguntaremos qué cosas hacen, qué tipo de acciones pueden realizar...

|  |
| --- |
| Un coche puede arrancar, detenerse, girar a la izquierda, acelerar, frenar, encender sus luces.  Método: funciones que cambian el estado, a veces cambian atributos |
| Un semáforo puede cambiar de estado. |
| Un bar puede abrir, cerrar, servirte una cerveza, cobrarla, modificar la lista de precios.  Una persona puede hablar, dormir, conducir un coche, tomarse una cerveza en un bar. |

En terminología de la **programación orientada a objetos**, a estas funciones que determinan el comportamiento de una clase se las conoce como **métodos**.

|  |
| --- |
| Sabemos lo que es una **clase** y que está compuesta de **atributos** y **métodos**. Nuestra labor ahora consistirá en **modelar** en Java estas clases. |

Modelar no es otra cosa sino crear una **abstracción** que represente de algún modo una determinada realidad.

Para crear en Java la clase Coche procederíamos del siguiente modo:

|  |
| --- |
| class Coche |
| { |
| } |

Entre las llaves introduciremos los atributos y métodos de que consta la clase. Usamos para ello la palabra reservada **class**.

El nombre de la clase Coche lo escribimos con la primera letra en mayúsculas, pues es de común acuerdo entre los programadores en Java que los nombres de clases empiecen así.

Introduzcamos algunos atributos:

|  |
| --- |
| class Coche |
| { |
| String marca; |
| String modelo; |
| String color; |
| int numeroDePuertas; |
| int cuentaKilometros; |
| int velocidad; |
| boolean arrancado; |
| } |

**Declarar atributos** es algo similar a declarar una variable normal. El nombre del atributo se precede por su tipo. Así, en el ejemplo, tenemos tres atributos de tipo String que contendrán cadenas de caracteres; otros tres de tipo int para almacenar valores enteros; finalmente, el atributo arrancado, que utilizaremos para indicar si el coche está en marcha o no, es de tipo boolean, admitiendo como posibles valores true o false.

Los tipos int y boolean forman parte de los tipos básicos de Java, conocidos como tipos **primitivos**. Los presentaremos formalmente a su debido momento; por ahora es suficiente con que conozcamos su existencia y cómo los utilizamos.

El tipo String, que escribimos con la primera letra en mayúsculas (lo que debería darte una pista), no es más que otra clase, como lo son las clases Coche y Persona. Es una clase muy importante en Java que encapsula un buen conjunto de métodos para trabajar con cadenas de caracteres.

El concepto importante que debes entender es que los atributos no necesariamente son siempre de tipos básicos, sino que también pueden ser de cualquier clase, incluso de una propia que nosotros mismos hayamos creado.

Por ejemplo, podríamos definir un nuevo atributo de la clase Coche, llamado conductor, en el que figure la persona (de tipo Persona) que lo conduce:

|  |
| --- |
| Persona conductor; |
|  |

Fíjate también en la convención utilizada para nombres de variables compuestos de varias palabras. Se escriben todas juntas, pero iniciando cada palabra en mayúsculas, a excepción de la primera. Esto forma parte también del estilo de escritura Java.

Ten en cuenta que Java distingue mayúsculas de minúsculas. No es lo mismo numeroDePuertas que numerodepuertas.

Definamos ahora los métodos de la clase Coche:

|  |
| --- |
| class Coche |
| { |
| String marca; |
| String modelo; |
| String color; |
| int numeroDePuertas; |
| int cuentaKilometros; |
| int velocidad; |
| boolean arrancado; |
|  |
| void arrancar() |
| { |
| } |
|  |
| void parar() |
| { |
| } |
|  |
| void acelerar() |
| { |
| } |
|  |
| void frenar() |
| { |
| } |
|  |
| void pitar() |
| { |
| } |
|  |
| int consultarCuentaKilometros() |
| { |
| } |
| } |

El nombre de método viene seguido por un par de paréntesis, pues en ocasiones los métodos podrán recibir argumentos que luego se utilizarán en el cuerpo del método. Aunque el método no requiera argumentos los paréntesis son absolutamente necesarios.

Por otro lado, el nombre del método va precedido por el tipo del valor que devuelve. Hay que indicarlo incluso si el método no devuelve explícitamente ningún valor. Ese caso se indica con el tipo **void**.

Entre el par de llaves { } introduciremos el cuerpo del método, las instrucciones que indican su operatividad.

Escribamos algo de código básico en cada uno de ellos:

|  |
| --- |
| class Coche |
| { |
| String marca; |
| String modelo; |
| String color; |
| int numeroDePuertas; |
| int cuentaKilometros; |
| int velocidad; |
| boolean arrancado; |
|  |
| void arrancar() |
| { |
| arrancado = true; |
| } |
|  |
| void parar() |
| { |
| arrancado = false; |
| } |
|  |
| void acelerar() |
| { |
| velocidad = velocidad + 1; |
| } |
|  |
| void frenar() |
| { |
| velocidad = velocidad - 1; |
| } |
|  |
| void pitar() |
| { |
| System.out.println("Piiiiiiiiiiiiiiiiii"); |
| } |
|  |
| int consultarCuentaKilometros() |
| { |
| return cuentaKilometros; |
| } |
| } |
|  |

Los métodos arrancar() y parar() establecen el atributo booleano arrancado a true y false, respectivamente. Los métodos acelerar() y frenar() incrementan y decrementan en una unidad, respectivamente, el atributo velocidad. El método pitar() imprime en consola una cadena de caracteres. El método consultarCuentaKilometros() devuelve **a quien lo invoca** (fíjate en el return) lo que contiene el atributo cuentaKilometros. Observa que es el único que devuelve explícitamente un valor (de tipo entero); los restantes, aunque algunos modifican los atributos de la misma clase, no devuelven ningún valor a quien los llama.

Podríamos modelar la clase Persona del siguiente modo:

|  |
| --- |
| class Persona |
| { |
| char sexo; |
| String nombre; |
| int edad; |
| Coche coche;   // El coche que conduce esa persona |
|  |
| void saludar() |
| { |
| System.out.println("Hola, me llamo " + nombre); |
| } |
|  |
| void dormir() |
| { |
| System.out.println("Zzzzzzzzzzz"); |
| } |
|  |
| int obtenerEdad() |
| { |
| return edad; |
| } |
| } |

Fíjate que uno de los atributos es precisamente de la clase Coche que acabamos de definir:

|  |
| --- |
| Coche coche |
|  |

**El concepto de objeto**

Una vez hemos entendido el concepto de **Clase** en el paradigma de la Programación Orientada a Objetos, es momento de matizar qué es un **Objeto** en su sentido más práctico.

*Una clase es una plantilla a partir de la cual vamos a crear objetos.*

Una **Clase**, como hemos visto, no es más que una especificación que define las características y el comportamiento de un determinado tipo de objetos. Piensa en ella como si se tratara de una plantilla, molde o esquema a partir del cual podremos construir **objetos concretos**.

Consideremos la clase Coche que definimos en el artículo anterior:

Con esta plantilla, vamos a crear coches **concretos** de la marca, modelo y color que nos apetezca. Cada uno que creemos será un **objeto**, o **instancia**, de la clase Coche.

Fíjate, en esta terminología, en la equivalencia de los términos objeto e instancia para referirnos, ahora sí, a **entidades concretas (objetos)** de una determinada clase.

Fabriquemos, entonces, nuestro primer coche…

Cada objeto, como todas las variables en Java, ha de ser **declarado** antes de ser utilizado:

|  |
| --- |
| Coche coche1 |
|  |

Con esta instrucción declaramos la variable coche1 de tipo Coche. En cuanto creemos, con el comando que escribiremos a continuación, el objeto concreto, coche1 contendrá una **referencia** a ese objeto, es decir, almacenará la dirección de memoria en la que realmente se halla el objeto propiamente dicho. **Esto es muy importante**: coche1 no contendrá el objeto en sí, sino una dirección de memoria que apunta a él.

Materialicemos nuestro primer coche del siguiente modo:

|  |
| --- |
| coche1 = new Coche() |
|  |

La palabra reservada **new** se emplea para **crear nuevos objetos**, instancias de una determinada clase que indicamos a continuación, seguida de un par de paréntesis.

Veremos esto a su debido momento, pero por ahora tenemos bastante con retener que se está invocando a un método especial que tienen todas las clases, que sirve para **construir** el objeto en cuestión facilitándole sus valores iniciales. A este método se le conoce como **constructor** de la clase.

Podríamos haber realizado la declaración y la creación en una sola instrucción:

|  |
| --- |
| Coche coche1 = new Coche() |

Ahora, vamos a decir de qué marca y modelo es:

|  |
| --- |
| coche1.marca = "Seat" |
| coche1.modelo = "Panda" |

*(entramos dentro del objeto coche1 y cambiamos los atributos)*

Consulta el cuadro de arriba con la definición de la clase y observa que tanto marca como modelo son dos atributos de tipo String, por lo que referencian cadenas de caracteres que escribimos entre comillas.

Observa con cuidado la **notación punto**. Separamos el nombre de la variable que referencia al objeto del atributo empleando un **punto como separador**.

Pintemos de azul nuestro vehículo:

|  |
| --- |
| coche1.color = "Azul" |

Además tiene tres puertas, el cuenta kilómetros indica 250.000 y su velocímetro refleja 0 Km/h. Valores correspondientes a los atributos numeroDePuertas, cuentaKilometros y velocidad, respectivamente, todos de tipo entero.

|  |
| --- |
| coche1.numeroDePuertas = 3 |
|  |
| coche1.cuentaKilometros = 250000 |
|  |
| coche1.velocidad = 0 |
|  |

Su motor está detenido, hecho que representamos a través de la variable booleana arrancado:

|  |
| --- |
| coche1.arrancado = false |

Ahora vamos a experimentar con los métodos. Arranquemos el Panda:

|  |
| --- |
| coche1.arrancar() |

De nuevo, empleamos también la notación punto para separar la variable del método.

Si observas el código verás que este método se limita a hacer que la variable booleana arrancado valga ahora true. Podrías decir que hubiéramos logrado el mismo resultado actuando sobre el atributo directamente, en lugar de invocar al método:

|  |
| --- |
| coche1.arrancado = true |

En efecto, es así; pero, como comprenderás más adelante, no suele ser buena idea dejar que los programas campen a sus anchas y modifiquen arbitrariamente los atributos de un objeto, siendo preferible que sean los métodos los que se ocupen de esa labor. Imagina que el programa intenta hacer que el cuenta kilómetros marque una cantidad negativa, o que el número de puertas sea igual a cincuenta. Un método correctamente diseñado podría gestionar que los valores estuvieran dentro del rango adecuado y asegurarse de que se cumplen las condiciones que permitirían la modificación del atributo.

La Programación Orientada a Objetos implementa un mecanismo, denominado **encapsulación**, que nos permite **ocultar** determinadas facetas de nuestro objeto y dejar solo accesibles aquellas partes que nos interesen. Pero vayamos por orden, todo a su momento…

Vamos a acelerar nuestro coche:

|  |
| --- |
| coche1.acelerar() |
|  |

Lo que provocará, si consultas el código, que el velocímetro incremente en una unidad su valor.

|  |
| --- |
| coche1.pitar() |

Lo que ocasionará un pitido.

Puedes crear todos los objetos de la clase Coche que desees:

|  |
| --- |
| Coche coche2 = new Coche() |

Cada uno con su propia colección de atributos:

|  |
| --- |
| coche2.marca = "Ford" |
| coche2.modelo = "Fiesta" |
| coche2.color = "Negro" |

Incluso podrías crear otros Seat Pandas, por supuesto. Aunque, en un instante dado, compartieran los mismos valores en sus atributos, se trataría de objetos distintos, **ubicados en direcciones de memoria diferentes** y cada uno podría seguir su propia trayectoria vital.

Extraído de:

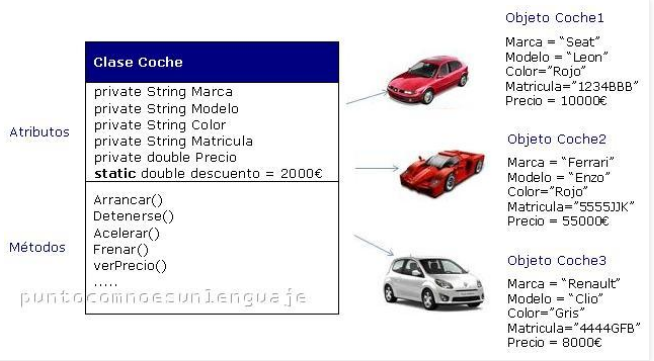
<http://elclubdelautodidacta.es/wp/indice-java/>

**UT4 Java static. Atributos y métodos estáticos o de clase**

Los atributos y métodos estáticos también se llaman **atributos de clase** y **métodos de clase**. Se declaran como **static**.

Supongamos una clase Coche sencilla que se utiliza en un programa de compra-venta de coches usados y de la que se crean 3 objetos de tipo Coche.

La clase contiene 6 atributos: marca, modelo, color, matrícula, precio y descuento. Supongamos que el descuento es una cantidad que se aplica a todos los coches sobre el precio de venta. Como este dato es el mismo para todos los coches y es un valor que se puede modificar en cualquier momento no debe formar parte de cada coche sino que es un dato que deben compartir todos. Esto se consigue declarándolo como static.



**Atributos static:**

Son **propios** únicamente **de la clase** **y no de los objetos que pueden crearse de la misma**, por lo tanto, sus valores son compartidos por todos los objetos de la clase. Van **precedidos del modificador static**.

Un atributo static:

* No es específico de cada objeto. Solo hay una copia del mismo y su valor es compartido por todos los objetos de la clase.
* Podemos considerarlo como una variable global a la que tienen acceso todos los objetos de la clase.
* Existe y puede utilizarse aunque no existan objetos de la clase.

Para invocar a una variable estática **no se necesita crear un objeto de la clase en la que se define**:

* Si se invoca **desde la clase** en la que se encuentra definido, basta con escribir su nombre.
* Si se le invoca **desde una clase distinta**, debe anteponerse a su nombre, el de la clase en la que se encuentra definido seguido del operador punto (.) **<NombreClase>.variableEstatica**.

**Ejemplo1:**

public class SerHumano{

String nombre;

String colorOjos;

int edad;

/\*

\* Declaración e inicialización de una variable de instancia estática

\* Tiene sentido declararla estática pues todos los objetos

\* de la clase, teniendo en cuenta que esta modela a un ser humano,

\* habitan en el mismo planeta

\*/

static String planeta="Tierra";

void mostrarCaracteristicas(){

System.out.println(nombre+" tiene "+edad+" años");

System.out.println("Sus ojos son "+colorOjos);

System.out.println("Su planeta es "+planeta);

}

void esMayorEdad(){

if(edad>=18){

System.out.println(nombre+" es mayor de edad");

System.out.println("Tiene "+edad+" años");

}

else{

System.out.println(nombre+" es menor de edad");

System.out.println("Tiene "+edad+" años ");

}

}

public static void main(String args[]){

SerHumano sh1=new SerHumano ();

sh1.nombre="Jesus";

sh1.colorOjos="azules";

sh1.edad=28;

sh1.mostrarCaracteristicas();

sh1.eresMayorEdad();

System.out.println("---------------------");

SerHumano sh2=new SerHumano ();

sh2.nombre="Rebeca";

sh2.colorOjos="verdes";

sh2.edad=27;

sh2.mostrarCaracteristicas();

sh2.esMayorEdad();

System.out.println("---------------------");

System.out.println("FIN DEL PROGRAMA");

}

}

**Métodos static:**

Van **precedidos del modificador** **static**.

Para invocar a un método estático **no se necesita crear un objeto de la clase en la que se define**:

* Tiene acceso solo a los **atributos estáticos de la clase**.
* No es necesario instanciar un objeto para poder utilizarlo.

Si se le invoca desde una clase distinta, debe anteponerse a su nombre, el de la clase en la que se encuentra seguido del operador punto (.)

<NombreClase>.metodoEstatico

**Ejemplo:**

Vamos a escribir una clase Persona que contendrá un atributo contadorPersonas que indique cuántos objetos de la clase se han creado.

contadorPersonas debe ser un atributo de clase ya que no es un valor que se deba guardar en cada objeto persona que se crea, por lo tanto se debe declarar static:

public static int contadorPersonas;

Si lo declaramos como privado:

private static int contadorPersonas;

Desde fuera de la clase Persona solo podremos acceder al atributo a través de métodos static:

public static void incrementarContador(){

contadorPersonas++;

}

public static int getContadorPersonas() {

return contadorPersonas;

}

En este caso un ejemplo de uso puede ser:

System.out.println(Persona.getContadorPersonas());

Cada vez que se crea una persona se incrementará su valor.

Si no es private, desde otra clase podemos hacerlo así:

Persona.contadorPersonas++;

Si es private, desde otra clase debemos incrementarlo así:

Persona.incrementarContador();

//Clase Persona

public class Persona {

private String nombre;

private int edad;

private static int contadorPersonas;

public Persona() {

}

public Persona(String nombre, int edad) {

this.nombre = nombre;

this.edad = edad;

}

public void setNombre(String nom) {

nombre = nom;

}

public String getNombre() {

return nombre;

}

public void setEdad(int ed) {

edad = ed;

}

public int getEdad() {

return edad;

}

public static int getContadorPersonas() {

return contadorPersonas;

}

public static void incrementarContador() {

contadorPersonas++;

}

}

//Clase Principal

public class Estatico1 {

public static void main(String[] args) {

Persona p1 = new Persona("Tomás Navarra", 22);

Persona.incrementarContador();

Persona p2 = new Persona("Jonás Estacio", 23);

Persona.incrementarContador();

System.out.println("Se han creado: " + Persona.getContadorPersonas() + " personas");

}

}

En lugar de utilizar el método incrementarContador() cada vez que se crea un objeto, podemos hacer el incremento de la variable estática directamente en el constructor.

El código de la clase Persona y de la clase principal quedaría ahora así:

//Clase Persona

public class Persona {

private String nombre;

private int edad;

private static int contadorPersonas;

public Persona() {

contadorPersonas++;

}

public Persona(String nombre, int edad) {

this.nombre = nombre;

this.edad = edad;

contadorPersonas++;

}

public void setNombre(String nom) {

nombre = nom;

}

public String getNombre() {

return nombre;

}

public void setEdad(int ed) {

edad = ed;

}

public int getEdad() {

return edad;

}

public static int getContadorPersonas() {

return contadorPersonas;

}

}

//Clase Principal

public class Estatico1 {

public static void main(String[] args) {

Persona p1 = new Persona("Tomás Navarra", 22);

Persona p2 = new Persona("Jonás Estacio", 23);

System.out.println("Se han creado: " + Persona.getContadorPersonas() + " personas");

}

}

*No conviene usar muchos métodos estáticos, pues si bien se aumenta la rapidez de ejecución, se pierde flexibilidad, no se hace un uso efectivo de la memoria y no se trabaja según los principios de la Programación Orientada a Objetos*.

|  |
| --- |
| **NOTA:** muchas clases de la API disponen de métodos estáticos. Por ejemplo, la **clase Math** del paquete java.lang cuenta con multitud de ellos. Estos métodos se emplean para realizar operaciones matemáticas. La **clase Thread**, del mismo paquete, cuenta con varios: uno que se emplea para retardar la ejecución de código es “**void sleep(long retardo)**”. Consultar la API. Lo importante de estos métodos es que para su utilización no es necesario instanciar un objeto de las clases en las que se encuentran ya que son estáticos. |

public class MetodosEstaticosAPI{

public static void main(String args[]){

int num=100;

System.out.println("La raiz cuadrada de "+num+" es "+Math.sqrt(num));

//Bloque try ... catch. Se estudiará más adelante. A nivel de

//ejecución no afecta.

//Se introduce un retardo en la ejecución del código de 3 sg

try{

Thread.sleep(3000); //espera 3000ms

}catch(InterruptedException e){}

System.out.println("La potencia de 2 elevado a 8 es "+Math.pow(2,8));

}

}

|  |
| --- |
| **NOTA**: **this** no se puede utilizar en un método estático |

# UT4 Ejercicios básicos de Java

Ejercicio 1.-

Define una clase Persona considerando los siguientes atributos de clase: nombre (String), apellidos (String), edad (int), casado (boolean), numeroDocumentoIdentidad (String). Define un constructor y los métodos para poder establecer y obtener los valores de los atributos.

Ejercicio 2.-

Considera que estás desarrollando un programa Java donde necesitas trabajar con objetos de tipo DiscoMusical. Define una clase DiscoMusical considerando los siguientes atributos de clase: titulo (String), autor (String), añoEdicion (int), formato (String), digital (boolean). Define un constructor y los métodos para poder establecer y obtener los valores de los atributos.

Ejercicio 3.-

Define una clase Profesor considerando los siguientes atributos de clase: nombre (String), apellidos (String), edad (int), casado (boolean), especialista (boolean). Define un constructor que reciba los parámetros necesarios para la inicialización y otro constructor que no reciba parámetros. Crea los métodos para poder establecer y obtener los valores de los atributos.

Ejercicio 4.-

Define una clase Bombero considerando los siguientes atributos de clase: nombre (String), apellidos (String), edad (int), casado (boolean), especialista (boolean). Define un constructor que reciba los parámetros necesarios para la inicialización y los métodos para poder establecer y obtener los valores de los atributos.

Ejercicio 5.-

Se pide definir una clase Medico (que representa a un médico de un hospital) con los siguientes atributos de clase: nombre (String), apellidos (String), edad (int), casado (boolean), numeroDocumentoIdentidad (String), especialidad (String). Definir un constructor que permita asignar valores de defecto a los atributos y los métodos para poder establecer y obtener los valores de los atributos. En cada método, incluye una instrucción para que se muestre por consola un mensaje informando del cambio. Por ejemplo si cambia la especialidad del médico, debe aparecer un mensaje que diga: "Ha cambiado la especialidad del médico de nombre … . La nueva especialidad es: …".

Ejercicio 6.-

Dada la clase Medico del ejercicio anterior, añade un método de nombre "calculoParaMultiploEdad" que no recibe parámetros y que permita determinar cuántos años faltan para que la edad del médico sea múltiplo de 5 y mostrar un mensaje informativo por pantalla. Por ejemplo si el médico tiene 22 años deberá en primer lugar obtener el resto de la división de 22 entre 5, que es 2. Ahora obtendrá los años que faltan para que el médico tenga una edad múltiplo de 5, que serán 5-2 = 3 años. A continuación deberá mostrar un mensaje por consola del tipo: "El médico de nombre … con especialidad … tendrá una edad múltiplo de 5 dentro de … años".

Ejercicio 7.-

Diseñar una clase Rueda que permita representar una rueda de un vehículo. Sus atributos de clase serán: tipo (String), grosor (double), diametro (double), marca (String). Define un constructor asignando unos valores de defecto a los atributos y los métodos para poder establecer y obtener los valores de los atributos. Crea un método denominado **comprobarDimensiones** donde a través de condicionales if realices las siguientes comprobaciones:

a) Si el diámetro es superior a 1.4 debe mostrarse por consola el mensaje "La rueda es para un vehículo grande". Si es menor o igual a 1.4 pero mayor que 0.8 debe mostrarse por consola el mensaje "La rueda es para un vehículo mediano". Si no se cumplen ninguna de las condiciones anteriores debe mostrarse por pantalla el mensaje "La rueda es para un vehículo pequeño".

b) Si el diámetro es superior a 1.4 con un grosor inferior a 0.4, ó si el diámetro es menor o igual a 1.4 pero mayor que 0.8, con un grosor inferior a 0.25, deberá mostrarse por consola el mensaje "El grosor para esta rueda es inferior al recomendado".

Ejercicio 8.-

Diseñar una clase Motor que representa el motor de una bomba para mover fluidos. Define la clase Motor considerando los siguientes atributos de clase: tipoBomba (int), tipoFluido (String), combustible (String). Define un constructor asignando unos valores de defecto a los atributos y los métodos para poder establecer y obtener los valores de los atributos.

Añade un método tipo procedimiento denominado **dimeTipoMotor()** donde a través de un condicional switch hagas lo siguiente:

a) Si el tipo de bomba es 0, mostrar un mensaje por consola indicando "No hay establecido un valor definido para el tipo de motor".

b) Si el tipo de bomba es 1, mostrar un mensaje por consola indicando "El motor es un motor de agua".

c) Si el tipo de bomba es 2, mostrar un mensaje por consola indicando "El motor es un motor de gasolina".

d) Si el tipo de bomba es 3, mostrar un mensaje por consola indicando "El motor es un motor de hormigón".

e) Si el tipo de bomba es 4, mostrar un mensaje por consola indicando "El motor es un motor de pasta alimenticia".

f) Si no se cumple ninguno de los valores anteriores mostrar el mensaje "No se puede clasificar el motor.

Ejercicio 9.-

Dada la clase del ejercicio anterior Motor, diseña un método tipo función que devuelva un booleano (true o false) denominado **dimeSiMotorEsParaAgua()** donde se cree una variable local booleana motorEsParaAgua de forma que si el tipo de motor tiene valor 1 tomará valor true y si no lo es tomará valor false. El método debe devolver la variable local booleana motorEsParaAgua.

# UT4 Ejercicios de clases 1

**PRIMEROS EJERCICIOS DE CLASES**

**1.-** Indica la salida al siguiente programa.

class Ejercicio{

public static void main(String[] args){

Clase1 obj1=new Clase1();

obj1.imprimir(24.3,5);

}

}

class Clase1{

//objeto

private double valor=9.8;

private int x=7;

public void imprimir(double valor, int x){

System.out.print(valor + ” ” + this.x);

//valor se refiere al de Ejercicio

//this.x se refiere al de la Clase1

}

}

*Salida Solucion:*

*24.3 7*

**2.-** Indicar la salida al siguiente programa.

class Ejercicio {

public static void main(String[] args){

Clase1 obj1=new Clase1(5,4);

System.out.print(obj1.modificar(4)+” “);

Clase1 obj2=new Clase1(5,4);

System.out.print(obj2.modificar(5)+” “);

obj2=obj1;

System.out.print(obj2.modificar(5)+” “);

}

}

class Clase1{

int p1,p2;

public Clase1(int i, int j){

p1=i;

p2=j;

}

public int modificar(int i){

p1=p1+i;

p2=p2+i;

System.out.print(p2+” “);

return p1;

}

}

//SERIA UTIL PARA DEPURAR

*Salida solucion:*

*1er print*

*P1 = 5+4*

*P2 =4+4*

*Print 8*

*Print 9*

*2do print*

*Print 9*

*Print 10*

*3er print*

*8+5=13  
9+5=14*

**3.-** Crea una clase que represente a un círculo. Debe tener tres atributos, las coordenadas x e y de su centro y su radio.

Tendrá tres constructores, sin parámetros (el centro será 0,0 y el radio 1), pasándole sólo el radio (el centro será 0,0) y pasándole el radio y las coordenadas X e Y.

También tendrá tres métodos uno que calcule el área, otro que calcule la longitud y otro que escriba los resultados.

Después, haz una clase principal en la que se creen 3 objetos círculos y se prueben los métodos.

**4.-.** Crea una clase Punto que modele un punto en un espacio bidimensional. Tendrá dos atributos, x e y, que guardan las coordenadas. Habrá un constructor sin parámetros que crea un punto en (0, 0) y otro al que se le pueden pasar las coordenadas del punto. También habrá métodos para obtener las coordenadas y para imprimir el punto con el formato (x,y).

**5.-** Realiza un programa en Java con la creación de una clase llamada Coche con los atributos color, marca, matricula, número de puertas. Crea un constructor que inicialice el objeto con estos valores y otro que funcione como un constructor por defecto. Escribe métodos para devolver cada uno de los valores de los atributos (dameMatricula, dameColor,...) y un método que simule la operación de pintar el coche cambiando su color. Crea un método main que implemente la solución

**6.-** Realiza un programa en Java que permita crear cuentas bancarias pidiendo la cantidad inicial al usuario, así como realizar operaciones de ingresar y sacar dinero de esas cuentas. Si la cantidad de dinero a sacar es superior a la que hay en la cuenta se mostrará un mensaje advirtiendo que no se puede realizar la operación. El programa irá contando el número de cuentas creadas y lo mostrará al final. El programa dispondrá de un método que imprima la cantidad de dinero que queda en la cuenta. Crea un método main que implemente la solución.

# UT4 Ejercicios de clases 2

**1.-** Crea una clase llamada Libro que guarde la información de cada uno de los libros de una biblioteca. La clase debe guardar el título del libro, autor, número de ejemplares del libro y número de ejemplares prestados. La clase contendrá los siguientes métodos:

Constructor por defecto.

Constructor con parámetros.

Métodos Setters/getters

Método préstamo que incremente el atributo correspondiente cada vez que se realice un préstamo del libro. No se podrán prestar libros de los que no queden ejemplares disponibles para prestar. Devuelve true si se ha podido realizar la operación y false en caso contrario.

Método devolución que decremente el atributo correspondiente cuando se produzca la devolución de un libro. No se podrán devolver libros que no se hayan prestado. Devuelve true si se ha podido realizar la operación y false en caso contrario.

Método toString para mostrar los datos de los libros. Este método se heredada de Object y lo debemos modificar (override) para adaptarlo a la clase Libro.

Escribe un programa para probar el funcionamiento de la clase Libro.

**2.-** Creamos una clase llamada empleado con los atributos nombre, apellido, edad, salario de forma que no se puedan acceder de ninguna clase externa. Crea un constructor vacío y otro que reciba cuatro parámetros con los valores de sus cuatro atributos. Crea un método para cambiar cada uno de los atributos y otro para devolver cada uno de los atributos, todos deben poder ser usados desde cualquier clase externa.

Crea un método que permita que a un empleado se le sume un plus a su salario siempre y cuando su edad sea de 40 años o más.

Para probar la clase Empleado que has desarrollado crea una clase que cree varios empleados (pide el número al usuario) y los referencie en un array que también crearás. Sube a los empleados que tengan derecho 50 euros a su sueldo, indicando en el programa a qué empleados se les ha podido realizar la subida y a cuáles no (imprime los datos de cada empleado, a continuación si se les ha subido o no el sueldo y el nuevo sueldo que tendrán).

Si para hacer lo que te piden necesitas a algún método más de los que se te indican, añádelo.

# UT4 Ejercicios de clases 3

**1.1.-** Escribe el código en Java de una clase para representar un empleado de una empresa. Del empleado vamos a querer tener los datos de su nombre, apellido, edad y salario.

Realiza dos métodos constructores, uno con los valores de los cuatro atributos y otro sin atributos.

Crea también los métodos que devuelvan los valores de los atributos y otros que permitan modificarlos, así como un método que si el empleado tiene más de 40 años se le aumente el sueldo una cantidad que se pasa por parámetro. A este último método le llamaremos comprobarPlus y devuelve true si se aumenta el sueldo y falso si no se aumenta.

Se debe crear también un método toString() para imprimir un objeto Empleado.

Utiliza la clase JOptionPane para recibir los valores del usuario para ir creando los empleados. Para ello, crearás un método denominado **leerEmpleado()** en la misma clase donde tengamos el método main(), este método pide los datos de un empleado y devuelve un objeto de la clase Empleado.

Crea tres empleados para probar los métodos descritos.

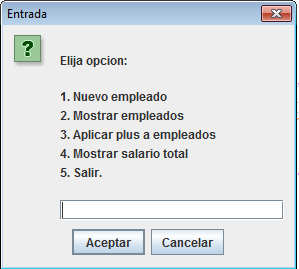
**1.2.-** Haz el ejercicio creando un array de empleados, donde se puedan acumular hasta 100 empleados (usa menos para probar el programa). Crea los empleados guardándolos en el array, para posteriormente comprobar si hay que aumentar el plus a los empleados.

Muestra los datos de cada empleado del array.

Muestra la suma del salario de todos los empleados antes y después de realizar el aumento que corresponda.

Haz el ejercicio con JOptionPane.

**1.3.-** Haz el ejercicio ahora con un menú como el siguiente:



Para ir creando los empleados usarás la **1ª opción** y un **atributo estático** que va contando los empleados según se van creando en el/los constructor/es.

En la **2ª opción** mostrarás todos los empleados creados hasta el momento en que se selecciona esa opción

En la **3ª** **opción**, se recorre el array y se sube la cantidad indicada como plus a los mayores de 40 años. Fíjate en que si se elige varias veces esta opción, habrá empleados a los que se les aplique la subida más de una vez. Busca una solución a este problema.

En la **4ª opción**, se muestra la suma de todos los empleados creados hasta el momento en que se selecciona la opción.

En esta versión del programa, deberás crear métodos estáticos en el fichero fuente del main, que reciban el array de empleados y lleven a cabo lo que se pide en cada opción. Es decir, **en el menú aparecerán solo las llamadas a los métodos**.

Si el usuario selecciona alguna opción y no hay empleados, el programa deberá notificarlo.

**2.-**  Implementa una clase en Java que permita realizar promedios. La clase debe tener dos métodos, uno para ingresar un nuevo número, llamado *agregarNumero(int numero)* y otro para obtener el promedio hasta el momento, llamado *obtenerPromedio()*. Determina qué atributos son necesarios para implementarla.

Implementa luego una clase de prueba que permita introducir algunos valores y que muestre el promedio.

**3.-** Implementa una clase en Java que represente una fracción de números enteros. Implementa asimismo los siguientes métodos en la clase Fracción:

* Suma de dos fracciones, que será una nueva fracción. Resta de dos fracciones, similar a la suma.
* División de dos fracciones.
* Multiplicación de dos fracciones.
* Calculo de la inversa de una fracción (cambiar numerador por denominador y viceversa).

Todos los métodos se realizan sobre un objeto Fracción pasándole por parámetro la segunda fracción cuando sea necesario.

Intenta simplificar los resultados usando un nuevo método.

**4.-** Realiza un programa en Java que, mediante un menú con varias opciones haga las siguientes tareas (hazlo con Scanner y con la clase JOptionPane):

**Opción 1:** Pide los datos de un alumno (nombre, apellido, curso, nota) y da de alta al alumno en nuestro programa.

**Opción 2:** Muestra los datos de los alumnos cuya nota media es mayor o igual a 5 y el número de ellos que hay que cumplan esa media.

**Opción 3:** Muestra los datos de los alumnos cuya nota media es menor a 5 y el número de ellos que hay que cumplan esa media.

Utiliza funciones (métodos estáticos) para llevar a cabo las tareas pedidas.

**5.-** Crea una clase en Java llamada Cubo con dos atributos:

int capacidad; // capacidad máxima en litros

int contenido; // contenido actual en litros

La clase tendrá un constructor que recibe la capacidad inicial del cubo. Una vez establecida la capacidad, ya no será posible modificarla.

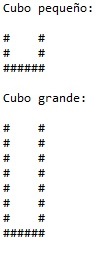
Habrá un método llamado **llena()**, que llenará el cubo hasta su capacidad máxima y otro **vacia()** que pone el contenido a 0 litros.

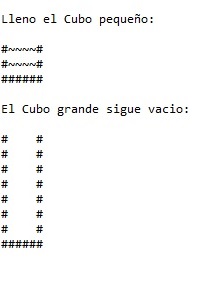
Haz un método llamado **vuelcaEn()** que vuelca el contenido de un cubo sobre otro. Antes de echar el agua se comprueba cuánto le cabe al cubo destino (será el que se manda por parámetro (void vuelcaEn(Cubo destino)).

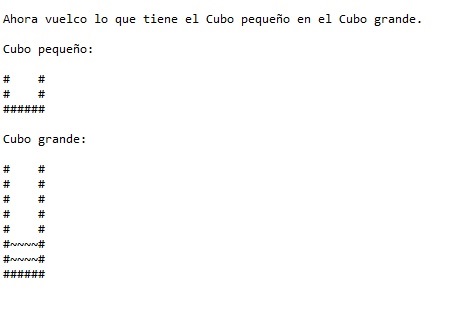
Se pide también un método llamado **pinta()** que pinta un cubo en la pantalla. Se muestran los bordes del cubo con el carácter # y el agua que contiene con el carácter ~.

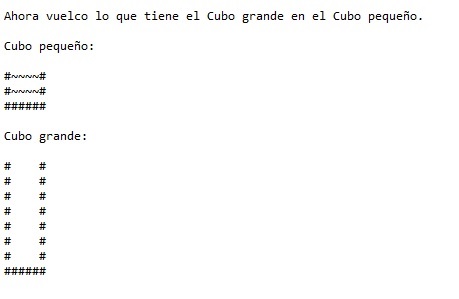
Cada litro se representa con una línea.

Para probar el ejercicio, ve haciendo las sentencias correspondientes a las siguientes salidas:









# UT4 Ejercicios de clases 4

**1.-** Desarrolla una clase Cafetera con atributos:

**capacidadMaxima** (la cantidad máxima de café que puede contener la cafetera)

**cantidadActual** (la cantidad actual de café que hay en la cafetera)

Implementa, al menos, los siguientes métodos:

* **Constructor** predeterminado: establece la capacidad máxima en 1000 (c.c.) y la actual en cero (cafetera vacía).
* **Constructor** con la capacidad máxima de la cafetera; inicializa la cantidad actual de café igual a la capacidad máxima.
* **Constructor** con la capacidad máxima y la cantidad actual. Si la cantidad actual es mayor que la capacidad máxima de la cafetera, la ajustará al máximo.
* **Accedentes y mutadores (getter y setter)**.
* **llenarCafetera()**: hace que la cantidad actual sea igual a la capacidad máxima.
* **servirTaza(int)**: simula la acción de servir una taza con la capacidad indicada por parámetro. Si la cantidad actual de café en la cafetera “no llega” para llenar la taza, se sirve lo que quede.
* **vaciarCafetera()**: pone la cantidad de café actual en cero.
* **agregarCafetera(int)**: añade a la cafetera la cantidad de café indicada en el parámetro. Si esa cantidad excede el máximo, ajusta al máximo.

Escribe el código en el programa principal para probar lo siguiente:

**Capacidad máxima = 1000 Cantidad actual = 0**

**Agregamos 20 c.c. de cafe...**

**Capacidad máxima = 1000 Cantidad actual = 20**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Llenamos la cafetera...**

**Capacidad máxima = 1000 Cantidad actual = 1000**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Servimos una taza de 500 c.c....**

**Todavia quedan 500 c.c.**

**Capacidad máxima = 1000 Cantidad actual = 500**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Servimos una taza de 600 c.c....**

**Se sirve todo lo que quedaba (500 c.c.)**

**Capacidad máxima = 1000 Cantidad actual = 0**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Servimos una taza de 200 c.c....**

**Lo siento, pero no queda nada de cafe.**

**Capacidad máxima = 1000 Cantidad actual = 0**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Llenamos la cafetera con 700 c.c....**

**Capacidad máxima = 1000 Cantidad actual = 700**

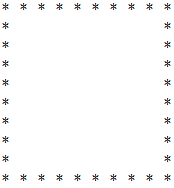
**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Llenamos la cafetera con 400 c.c....**

**Capacidad máxima = 1000 Cantidad actual = 1000**

**2.-** Implementa una clase en Java que permita representar cuadrados. Cada objeto Cuadrado vendrá representado por sus cuatro vértices, que serán los atributos de la clase. Además de los métodos para modificar y devolver los valores de los vértices, se piden los siguientes métodos:

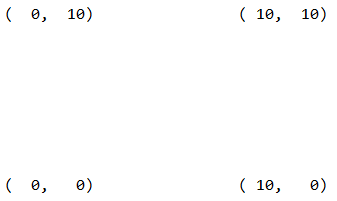
Cada cuadrado tendrá un método (dibujar) para representarse por medio de asteriscos. Por ejemplo, el cuadrado con vértices (0, 0) (0, 10) (10, 10) y (10, 0) se representaría:



También queremos un **método (dibRellenando)** que represente el cuadrado relleno de asteriscos.



Otro **método (dibujarVertices)** para escribir los 4 vértices del cuadrado de la forma siguiente para el ejemplo anterior:



Un **método (esCuadrado)** que devuelva true si se trata de los vértices de un cuadrado y false si no lo es.

La *distancia entre dos puntos* P1: (x1, y1) y P2: (x2, y2) viene dada por la fórmula:

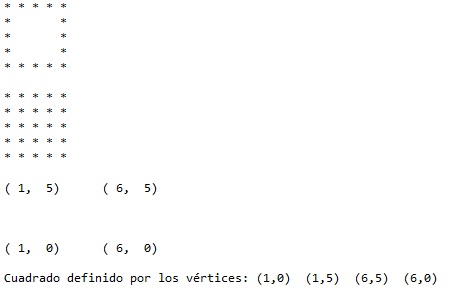
**D=**

Cada cuadrado tendrá un **método (lado)** que devuelve el valor del lado del cuadrado creado.

Escribe un **método (compara**) en la clase Cuadrado para poder comparar el área de dos cuadrados. Este método recibe un cuadrado y lo compara con el cuadrado que invoca el método, actuando igual que el método compare de la clase String.

Implementa luego una clase de prueba que permita crear objetos de la clase Cuadrado y probar todos sus métodos.

*Otro ejemplo:*



Observa que para dibujar los cuadrados vamos a ignorar la distancia al eje de las x y de las y, los pintamos siempre desde el punto P1(x1, y1), teniendo únicamente en cuenta la longitud del lado.

**3.-** Vamos a escribir un programa para representar el consumo de energía de una instalación eléctrica. Para ello, se hará una clase que representa los aparatos conectados en la instalación.

Cada aparato tiene un consumo eléctrico determinado. Al encender un aparato eléctrico, el consumo de energía se incrementa en la potencia de dicho aparato. Al apagarlo, se disminuye el consumo en dicha potencia.

Inicialmente, los aparatos están todos apagados.

Además, se desea consultar el consumo total de la instalación.

Haz un programa que declare dos aparatos eléctricos, una bombilla de 150 watios y una plancha de 2000 watios. El programa deberá imprimir el consumo nada más crear los objetos. Después, se enciende la bombilla y la plancha, y el programa imprime el consumo. Luego se apaga la bombilla, y se vuelve a imprimir el consumo.

**Ejemplo de salida:**

Inicialmente el consumo eléctrico es 0.0

Encendemos la bombilla Potencia 150.0

Encendemos la plancha Potencia 2000.0

El consumo eléctrico es 2150.0

Apagamos la plancha Potencia 2000.0

El consumo eléctrico es 150.0

# UT4 Ejercicios de clases: enunciado clase 1 de dic Fraccion

Implementa una clase en Java que represente una fracción de números enteros. Implementa asimismo los siguientes métodos en la clase Fracción:

* Suma de dos fracciones, que será una nueva fracción. Resta de dos fracciones, similar a la suma.
* División de dos fracciones.
* Multiplicación de dos fracciones.
* Calculo de la inversa de una fracción (cambiar numerador por denominador y viceversa).

Todos los métodos se realizan sobre un objeto Fracción pasándole por parámetro la segunda fracción cuando sea necesario.

Intenta simplificar los resultados usando un nuevo método.

<http://www.objetos.unam.mx/matematicas/leccionesMatematicas/01/1_005/index.html>