CONCEPTOS BÁSICOS

1. CONCEPTOS BÁSICOS

Los lenguajes de programación Orientados a Objetos, se caracterizan por tener su nivel de abstracción basado en el mundo real. Así, el énfasis está en la abstracción de datos, y los problemas del mundo real son representados por un conjunto de objetos de datos para los que se adjunta un conjunto correspondiente de operaciones.

Así, al igual que otros lenguajes de programación, introducen un nuevo conjunto de términos, o conceptos básicos que son esenciales comprender, para poder realizar cualquier análisis, diseño o desarrollo Orientado a Objetos:

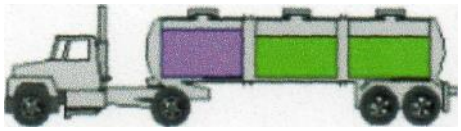
- **Objeto**
- **Atributo**
- **Método**
- **Interfaz**
- ***Clase***

A continuación veremos en más detalle cada uno de estos conceptos básicos.

2. OBJETO

Hay muchas definiciones que pueden darse de un objeto, entre las cuales se encuentran:

- Es cualquier cosa que vemos a nuestro alrededor, algo tangible y/o visible, animado o inanimado. **Por ejemplo, un camión, un perro, una cuenta bancaria …**

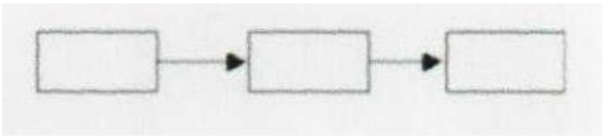


(camión)

- Algo que puede comprenderse intelectual mente. Por ejemplo, **un proceso de ordenación**

(proceso)

- Una entidad Software. **Por ejemplo una lista de cosas.**



(lista)

Definiciones dadas por creadores de metodologías Orientadas a Objetos como pueden ser:

- Un objeto se caracteriza por un número de **operaciones** y un **estado** que recuerda e! efecto de estas operaciones. **Ivar Jacobson**

- Un objeto tiene un **estado, comportamiento e identidad;** la estructura y comportamiento de objetos similares se definen en sus clases comunes. **Grady Booch**

- Un objeto es una entidad que tiene un **estado** (cuya represen tación está oculta) y un conjunto definido de **operaciones** que operan sobre ese estado, **lan Sommerville**

- Un objeto es una identidad con unos límites bien definidos que encapsulan **estado y comportamiento.** El estado se representa por atributos y relaciones, el comportamiento es representado por operaciones y métodos. **Object Management Group**

Los términos objeto e instancia son usados indistintamente.

Características de un objeto

Todos los objetos, tienen intrínsecos las siguientes características, como se han visto en alguna de definiciones del apartado anterior:

- **Identidad, Es un identificador unívoco para cada uno de los objetos.** En el caso de que los valores de los atributos fueran los mismos, es la única manera de poder determinar cada uno de los objetos. Así si tenemos dos cuentas corrientes con el mismo titular, y el mismo importe, la única forma de diferenciarlas es vía dicha identidad.
- **Comportamiento, Conjunto de operaciones o métodos** que proporcionan servicios a otros objetos que solicitan dichos servicios cuando necesitan que se realice una cierta operativa.
- **Estado. Conjunto de propiedades o atributos que recuerdan el efecto de las operaciones**.

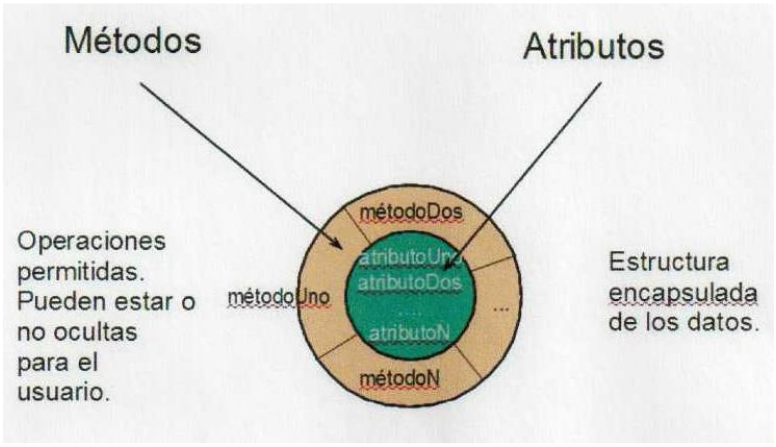
Ejemplo: Para un reloj determinado, la identidad o identificador podría ser unReloj, con los atributos hora (horas, min, seg), día (día, mes, año), modelo y numSerie y cuyos métodos u operaciones serían getHora, getDia, incrementarHora, incrementarDia, **l**impiarPantalla y traducírFrecuencia,

Estructura de un objeto

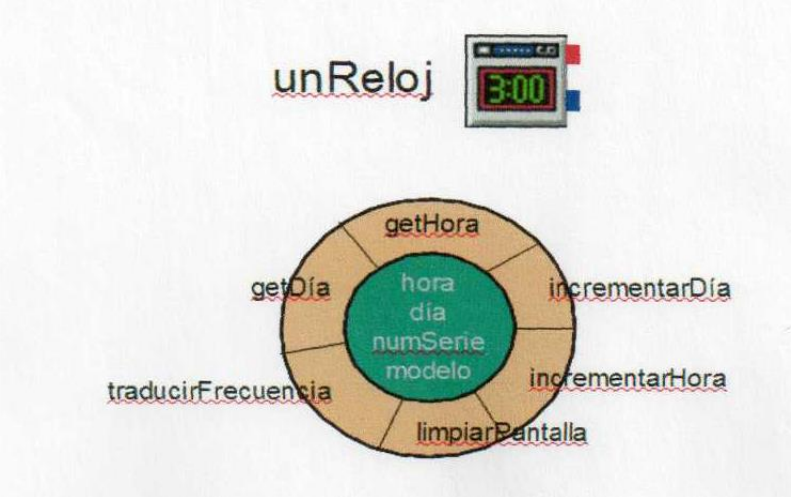
En base a las características del objeto mencionadas anteriormente, todo objeto está formado por atributos o estructura encapsulada de los datos y por los métodos u operaciones permitidas por dicho objeto, ya sean visibles para el usuario o no.
Los métodos pueden clasificarse de la siguiente manera:

- **Modificador (setter):** altera el estado de un objeto. Por ejemplo, setHora()
- **Selector (getter):** accede al estado de un objeto sin alterarlo.Por ejemplo getHora(X)
- **Iterador**: permite acceder a todas los elementos de un objeto. Solo disponible para colecciones de objetos.
- **Constructor**: crea un objeto e inicializa su estado. Por ejemplo RelojQ.
- **Destructor:** limpia el estado de un objeto y lo destruye. Por ejemplo ~Reloj(). No existe en Java.
- **Propósito general:** la lógica del programa. Por ejemplo, limpiarPantallaQ, incrementarDia().

Gráficamente, se puede visualizar la estructura de un objeto de la siguiente manera:



Ejemplo: Para el objeto unReloj mencionado anteriormente, la estructura sería la siguiente:

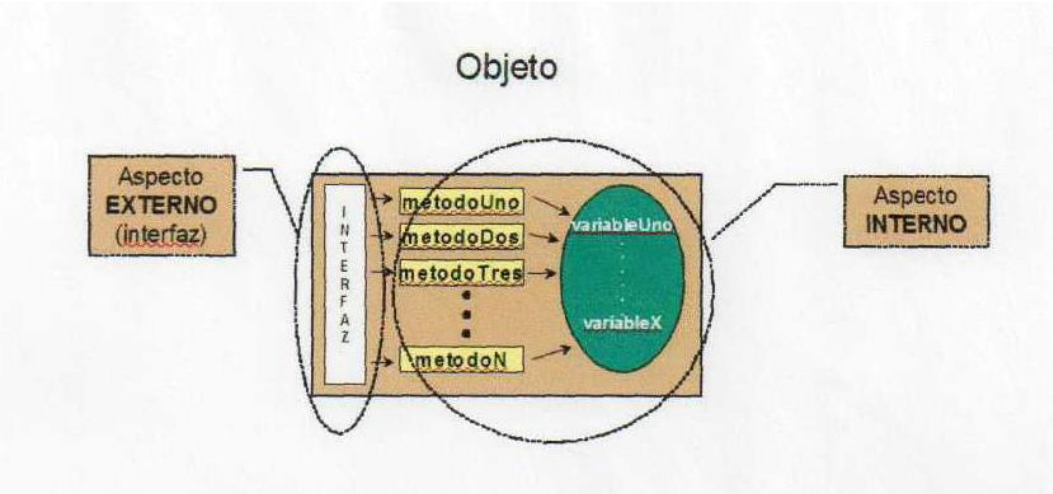


Atributo: Es una característica fundamental de cada objeto y por lo tanto como veremos posteriormente de una clase. Todos los atributos tienen algún valor, siendo este una cantidad, una relación con otro objeto... Si el valor del atributo es un valor fijo para todos los objetos, se dice que es un atributo estático

Método: Es una acción que se realiza sobre un objeto para consultar o modificar su estado.

El aspecto de los objetos.

Cuando se habla del aspecto de los objetos, no nos estamos refiriendo a los conceptos de buen o mal aspecto visual. Nos referimos a **como el objeto se ve internamente o aspecto interno y como ven al objeto desde otros objetos también llamado aspecto exterior.**



Este aspecto exterior, es llamado también **interfaz,** siendo la parte visible y accesible para el resto de objetos. Puede estar formado por uno o varios métodos. También se le define como el protocolo de comunicación de un objeto.

Es posible que exista algún método que solo pertenezca al aspecto interno pero no pertenezca al interfaz. En este caso, estos métodos no pueden ser llamados desde otros objetos, sino que solamente pueden ser llamados desde métodos del propio objeto.

Ejemplo: Para el objeto unReloj, el interfaz estaría formado por los métodos getHora, getDia, incrementarHora e incrementarDia. Los métodos limpiarPantalla y traducirFrecuencia solamente pertenecen (conjuntamente con los que forman el interfaz) al aspecto interno. Así el método LimpiarPantalla, es llamado por getHora y getDia antes de mostrar la información pedida en el método

Interfaz: Aspecto exterior que es visible al resto de objetos. Puede estar formado por uno o varios métodos.

3. CLASE

Hasta ahora hemos visto que define a un objeto. Una de las definiciones más sencillas es algo del mundo real, tangible o visible.

Hemos visto que los objetos están formados por atributos y métodos. La definición de estructura y comportamiento de un objeto es a lo que se denomina clase. Es por tanto un patrón para la definición de atributos y métodos para un tipo particular de objetos.

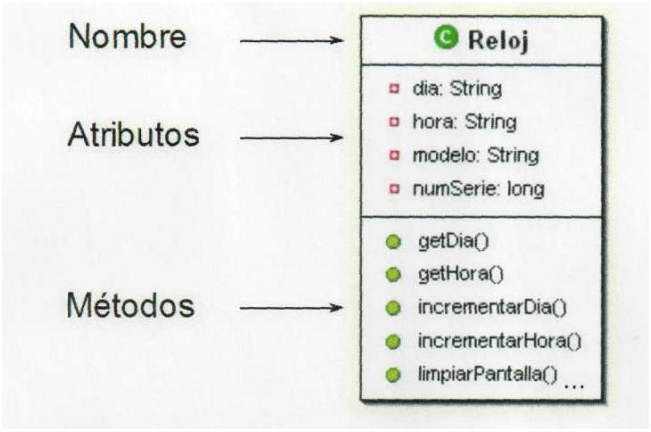
Todos los objetos de una clase dada son idénticos en estructura y comportamiento pero son únicos (aunque tengan los mismos valores en sus atributos).

Instancia es el término utilizado para referirse a un objeto que pertenece a una clase concreta.

Una clase, por tanto es solamente la definición. Mientras que un objeto o instancia es algo real con la estructura y comportamiento de la clase a la que pertenece.

La estructura de una clase, por tanto, viene determinada por un nombre, los atributos que contiene y los métodos que realiza.

Ejemplo: El objeto unReloj, pertenece a la clase Reloj, cuyo nombre es Reloj, cuyos atributos son dia, hora, modelo y numSerie y cuyos métodos son getHora, getDia, íncrementarHora, incrementarDia, limpiarPantalla y traducirFrecuencia.



Como se puede ver en la imagen anterior, en los diagramas UML, la definición de una clase, se realiza mediante un rectángulo, divido en tres partes y conteniendo en el siguiente orden: Nombre, Atributos y Métodos.

Clases versus Objetos.

Es importante saber diferenciar que es una clase y que es un objeto y en que consiste cada una de ellas. Por ello, recapitulamos toda la información mostrada hasta el momento.

- Una clase es un patrón para la definición del estado y el comportamiento de un tipo particular de objetos.
- Todos los objetos de una clase dada son idénticos en estructura y comportamiento, pero tienen identidad única.
- Un objeto pertenece a una clase en particular.
- Los objetos son creados y destruidos en tiempo de ejecución. Residen en el espacio de memoria.
- Así para la clase Reloj descrita anteriormente, tendremos los objetos unReloj (dia="01-03-2010", hora="13:01:03", modelo="Rolex", numSerie="123456") y otroReloj (dia="01-03-2010", hora="13:01:03", modelo="Swatch", numSe-rie="Sab748").

Pero si nos paramos a pensar, ¿qué surge antes? ¿La clase y por lo tanto creamos los objetos del mundo real? O partiendo de los objetos del mundo real ¿podemos definir su estructura y comportamiento?.
Para solventar esta problemática, aparece el **concepto de Clasificación**. La clasificación es el medio por el que ordenamos el conocimiento, ya que fundamentalmente es un problema de búsqueda de similitudes. Al clasificar buscamos grupos de cosas que tengan una misma estructura o exhiban un comportamiento común.
La clasificación dentro de la Orientación a Objetos, sobre todo en las fases de Análisis y Diseño, permite que los objetos con la misma estructura de datos y con el mismo comportamiento se agrupen para formar una clase.

4. CÓMO CREAR UNA CLASE DE OBJETOS EN JAVA

Según lo expuesto hasta ahora, un objeto contiene, por una parte, **atributos** que definen su estado, y por otra, **operaciones** que definen su comportamiento. También sabemos que un objeto es la representación concreta y específica de una clase. ¿Cómo se escribe una clase de objetos? Como ejemplo, podemos crear una clase ***COrdenador.***

```
class COrdenador
```

Observamos que para declarar una clase hay que utilizar la palabra reservada **class** seguida del nombre de la clase y del cuerpo de la misma. El cuerpo de la clase incluirá entre { y } los atributos y los métodos u operaciones que definen su comportamiento.

Los atributos son las características individuales que diferencian un objeto de otro. El color de una ventana Windows, la diferencia de otras; el D.N.I. de una persona la identifica entre otras; el modelo de un ordenador le distingue entre otros; etc.

La clase *COrdenador* puede incluir los siguientes atributos:

- Marca: Mitac, Toshiba, Ast
- Procesador: Intel, AMD
- Pantalla: TFT, DSTN, STN

Los atributos también pueden incluir información sobre el estado del objeto; por ejemplo, en el caso de un ordenador, si está encendido o apagado, si la presentación en pantalla está activa o inactiva, etc.

- Dispositivo: encendido, apagado
- Presentación: activa, inactiva

Todos los atributos son definidos en la clase por variables:

```
class  COrdenador
{
  String  Marca;
  String  Procesador;
  String  Pantalla;
  boolean OrdenadorEncendido;
  boolean Presentación;

  // …

}
```

Se han definido cinco atributos: tres de ellos, *Marca, Procesador* y *Pantalla*, pueden contener una cadena de caracteres (una cadena de caracteres es un objeto de la clase **String** perteneciente a la biblioteca estándar). Los otros dos atributos, *OrdenadorEncendido y Presentación,* son de tipo **boolean** (un atributo de tipo **boolean** puede contener un valor **true o false;** verdadero o falso). Se deben respetar las mayúsculas y las minúsculas.

No vamos a profundizar en los detalles de la sintaxis de este ejemplo ya que el único objetivo ahora es entender la definición de una clase con sus partes básicas. El resto de la sintaxis y demás detalles se irán exponiendo poco a poco más adelante.

El comportamiento define las acciones que el objeto puede emprender. Por ejemplo, pensando acerca de un objeto de la clase *COrdenador,* esto es, de un ordenador, algunas acciones que éste puede hacer son:

- Ponerse en marcha
- Apagarse
- Desactivar la presentación en la pantalla
- Activar la presentación en la pantalla
- Cargar una aplicación

Para definir este comportamiento hay que crear métodos. Los métodos son rutinas de código definidas dentro de la clase, que se ejecutan en respuesta a alguna acción tomada desde dentro de un objeto de esa clase o desde otro objeto de la misma o de otra clase. Recuerde que los objetos se comunican mediante mensajes.

Como ejemplo, vamos a agregar a la clase *COrdenador* un método que responda a la acción de ponerlo en marcha:

```
Void EncenderOrdenador()
{
  if  (OrdenadorEncendido == true) //si está encendido...
    System.out.println("El ordenador ya está en marcha.");
  else  //si no está encendido, encenderlo.
  {
    OrdenadorEncendido = true;
     System.out.println("El ordenador se ha encendido.");
  }
}
```

Como se puede observar un método consta de su nombre precedido por el del valor que devuelve cuando finalice su ejecución (la palabra reservada **void** indica que el método no devuelve ningún valor) y seguido por una lista de *parámetros* separados por comas y

encerrados entre paréntesis (en el ejemplo, no hay parámetros). Los paréntesis indican a Java que el identificador *(EncenderOrdenador)* se refiere a un método y no a un atributo. A continuación se escribe el cuerpo del método encerrado entre { y }. Ya conocemos algunos métodos, llamados en otros contextos funciones; por ejemplo la función logaritmo que devuelve un valor real correspondiente al logaritmo del valor pasado como argumento.

El método *EncenderOrdenador* comprueba si el ordenador está encendido: si lo está, simplemente visualiza un mensaje indicándolo; si no lo está, se enciende lo comunica mediante un mensaje.

Agregamos un método más para que el objeto nos muestre su estado:

```
void  Estado()

{
  System.out.println("\nEstado del   ordenador:"  +
                     "\nMarca   "  + Marca  +
                     "\nProcesador  " +  Procesador +
                     "\nPantalla   "  + Pantalla +
                     "\n");
   if  (OrdenadorEncendido == true)//si el ordenador está
   encendido
    System.out.println("El ordenador está  encendido.");
  else  //si no está encendido...
    System.out.println("El ordenador está apagado.");
 }
```

El método *Estado* visualiza los atributos específicos de un objeto. La secuencia de escape ***\n***, así se denomina, introduce un retorno de carro más un avance de línea (caracteres ASCII, CR LF).

En este instante, si nuestras pretensiones sólo son las expuestas hasta ahora ya tenemos creada la clase *COrdenador*. Para poder crear objetos de esta clase y trabajar con ellos, tendremos que escribir un programa, o bien añadir a esta el método **main.** Siempre que se trate de una aplicación (no de un *applet)* es obligatorio que la clase que define el comienzo de la misma incluya un método **main.** Cuando se ejecuta una clase Java compilada que incluye un método **main,** éste es lo primero que se ejecuta.

Hagamos lo más sencillo, añadir el método **main** a la clase *COrdenador*. El código completo, incluyendo el método **main,** se muestra a continuación:

```
class COrdenador
{
  String Marca;
  String Procesador;
  String Pantalla;
  boolean OrdenadorEncendido;
  boolean Presentación;

  void EncenderOrdenador()
  {
    if (OrdenadorEncendido == true) // si está encendido...
      System.out.println("El ordenador ya está encendido.");
    else // si no está encendido, encenderlo.
    {
      OrdenadorEncendido = true;
      System.out.println("El ordenador se ha encendido.");
    }
  }
```

```
  void Estado()
  {
    System.out.println("\nEstado del ordenador:" +
                       "\nMarca " + Marca +
                       "\nProcesador " + Procesador +
                       "\nPantalla " + Pantalla + "\n");
    if (OrdenadorEncendido == true) // si el ordenador está encendido...
      System.out.println("El ordenador está encendido.");
    else // si no está encendido...
      System.out.println("El ordenador está apagado.");
  }
```

```
  public static void main (String[] args)
  {
    COrdenador MiOrdenador = new COrdenador();
    MiOrdenador.Marca = "Ast";
    MiOrdenador.Procesador = "Intel Pentium";
    MiOrdenador.Pantalla = "TFT";
    MiOrdenador.EncenderOrdenador();
    MiOrdenador.Estado();
  }
}
```

El método **main** siempre se declara público y estático (no dependen de un objeto particular de la clase, se les puede invocar sin necesidad de crear objetos), no devuelve un resultado y tiene un parámetro ***args*** que es una matriz de una dimensión de cadenas de caracteres. Ya veremos para qué sirve. Analicemos el método **main** para tener una idea de lo que hace:

- La primera línea crea un objeto de la clase *COrdenador* y almacena una referencia al mismo en la variable *MiOrdenador*. Esta variable la utilizaremos para acceder al objeto en las siguientes líneas. Ahora quizás empiece a entender por qué anteriormente decíamos que un programa orientado a objetos se compone solamente de objetos.

- Las tres líneas siguientes establecen los atributos del objeto referenciado por *MiOrdenador*. Se puede observar que para acceder a los atributos o propiedades del objeto se utiliza el operador punto (**.**). De esta forma quedan eliminadas las ambigüedades que surgirían si hubiéramos creado más de un objeto.

- En las dos últimas líneas el objeto recibe los mensajes *EncenderOrdenador y Estado*. La respuesta a esos mensajes es la ejecución de los métodos respectivos, que fueron explicados anteriormente. Aquí también se puede observar que para acceder a los métodos del objeto se utiliza el operador punto.

En general, para acceder a un miembro de una clase (atributo o método) se utiliza la sintaxis siguiente:

nombre_objeto.nombre_miembro

La aplicación se guarda con el nombre *COrdenador.java*. Después se compila y ejecuta. Los resultados son los siguientes:

```
El ordenador se ha encendido.
```

```
Estado del ordenador: Marca Ast
Procesador Intel Pentium Pantalla TFT

El ordenador está encendido.
```

Otra forma de crear objetos de una clase y trabajar con ellos es incluir esa clase en el mismo fichero fuente de una clase aplicación, entendiendo por clase aplicación una que incluya el método **main** y cree objetos de otras clases. Por ejemplo, volvamos al instante justo antes de añadir el método **main** a la clase *COrdenador* y añadamos una nueva clase pública denominada *CMiOrdenador* que incluya el método **main.** El resultado tendrá el esqueleto que se observa a continuación:

```
public class CMi Ordenador
{
       public static void main (String[] args)
       {
           //…
       }
class COrdenador
{
     // …
}
```

Una aplicación está basada en una clase cuyo nombre debe coincidir con el del programa fuente que la contenga, respetando mayúsculas y minúsculas. Por lo tanto, guardemos el código escrito en un fichero fuente denominado *CMiOrdenador.java.* Finalmente, completamos el código como se observa a continuación, y compilamos y ejecutamos la aplicación. Ahora es la clase *CMiOrdenador* la que crea un objeto de la clase *COrdenador.* El resto del proceso se desarrolla como se explicó en la versión anterior. Lógicamente, los resultados que se obtengan serán los mismos que obtuvimos con la versión anterior.

```
public class CMiOrdenador
{
  public static void main (String[] args)
  {
    COrdenador MiOrdenador = new COrdenador();
    MiOrdenador.Marca = "Ast";
    MiOrdenador.Procesador = "Intel Pentium";
    MiOrdenador.Pantalla = "TFT";
    MiOrdenador.EncenderOrdenador();
    MiOrdenador.Estado();
  }
}

class COrdenador
{
  String Marca;
  String Procesador;
  String Pantalla;
  boolean OrdenadorEncendido;
  boolean Presentación;

  void EncenderOrdenador()
  {
    if (OrdenadorEncendido == true) // si está
```

```
encendido...
      System.out.println("El ordenador ya está
encendido.");
    else // si no está encendido, encenderlo.
    {
      OrdenadorEncendido = true;
      System.out.println("El ordenador se ha
encendido.");
    }
  }

  void Estado()
  {
    System.out.println("\nEstado del ordenador:" +
                       "\nMarca " + Marca +
                       "\nProcesador " + Procesador +
                       "\nPantalla " + Pantalla +
"\n");
    if (OrdenadorEncendido == true) // si el ordenador
está encendido...
      System.out.println("El ordenador está
encendido.");
    else // si no está encendido...
      System.out.println("El ordenador está apagado.");
  }
}
```

La aplicación *CMiOrdenador.java* que acabamos de completar tiene dos clases: la clase aplicación *CMiOrdenador* y la clase *COrdenador*. Observe que la clase aplicación es pública **(public)** y la otra no. **Cuando incluyamos varias clases en un fichero fuente, sólo una puede ser pública y su nombre debe coincidir con el del fichero donde se guardan.** Al compilar este fichero, Java creará tanto ficheros ***.class*** como clases separadas hay.

Según lo expuesto hasta ahora, en esta nueva versión también tenemos un fichero, el que almacena la aplicación, que tiene el mismo nombre que la clase que incluye el método **main,** que es por donde se empezará a ejecutar la aplicación.