Capítulo 1

Breve recorrido por Java

1.1. Características de Java

Antes de comenzar a programar en un lenguaje de programación nuevo conviene conocer sus características fundamentales. Dichas características determinan en gran medida el modo en que se debe emplear el lenguaje. En Java son destacables las siguientes características:

- **Java está orientado a objetos**: Java fue diseñado como un lenguaje nuevo que no debía ser compatible con otros lenguajes. Esto permitió un diseño del lenguaje basado en una aproximación limpia y pragmática a la programación orientada a objetos. En Java *todo es un objeto* salvo los tipos simples que pueden ser *no objetos* de alto rendimiento.
- **Java es simple**: El lenguaje fue diseñado para que los programadores profesionales pudiesen aprenderlo de modo fácil. Java puede verse como una evolución de C++ que evita los conceptos más confusos de C++. Al ser una evolución de C++ muchas partes de su sintaxis son conocidas para los programadores que hayan usado este lenguaje o incluso C.
- **Java es robusto**: En el sentido de que Java restringe opciones que se sabe pueden dar lugar a errores con facilidad. Esto conlleva que Java haga diversas comprobaciones en tiempo de ejecución. Es tarea del programador construir excepciones que gestionen todos los posibles errores de ejecución.
- **Java es interpretado**: Una aplicación Java tiene que compilarse, pero el compilador de Java no da lugar a código ejecutable. Genera los denominados bytecodes. Los bytecodes de Java pueden posteriormente interpretarse en cualquier sistema que tenga un interprete de Java. Este funcionamiento de Java combina las ventajas de los lenguajes compilados (la representación en bytecodes se pensó de modo que el interprete no sea lento como en otros lenguajes interpretados) con las de los interpretados (cualquier evolución del hardware no obliga a una recompilación de todos los programas, basta con una actualización del interprete de Java). Eso sí, siempre es necesario cargar el interprete de Java para ejecutar una aplicación Java.
- Java presenta otras características que utilizará en otros cursos: es multihilo, es distribuido, es dinámico, etc.

1.2. Obtención e instalación de recursos

Con lo dicho anteriormente queda claro que para poder ejecutar aplicaciones java es necesario que su sistema tenga instalada la máquina virtual Java (a la que la gente se suele referir con el acrónimo JVM). En la actualidad la mejor forma de saberlo es acceder a la página http://www.java.com/ y pulsar sobre la pregunta '¿Tengo Java?' que aparece justo debajo del botón de descarga gratuita de Java. A continución pulse el botón que comprueba nuestra versión de java. Si la prueba nos indica que no tenemos Java o que nuestra versión no es la correcta basta con pulsar 'Descargar' (en la parte superior y seguir las instrucciones).

También puede abrir una ventana de comandos y ejecutar el comando:

```
[user@foo userHome] $ java -version
java version ''1.8.0_31''
Java (TM) SE Runtime Environment (build 1.8.0_31-b13)
Java HotSpot (TM) 64-Bit Server VM (build 25.31-b13, mixed mode)
```

cuya salida le indicará que versión de java tiene instalada. Obviamente también es necesario tener el compilador de java. Para comprobar que lo tiene puede ejecutar en una ventana de comandos:

```
[user@foo userHome] $ javac
Usage: javac <options><source files>
where possible options include:
    -g              Generate all debugging info
    -g:none         Generate no debugging info
    ...
```

Cabe destacar que para ejecutar aplicaciones Java es necesario disponer, además de la máquina virtual, del *Java Application Programming Interface* (API).

Para programar en Java es necesario el *kit* de desarrollo (JDK) que permite crear y probar aplicaciones (y que incluye además la máquina virtual). En el momento de escritura de esta práctica puede obtener el JDK accediendo a la dirección de Oracle http://www.oracle.com/technetwork/java/javase/downloads/.

En esta última página también tiene acceso a NetBeans, un entorno de desarrollo (IDE) que se empleará durante las prácticas.

Por último, pero muy importante: Antes de comenzar a implementar una aplicación Java se tiene que preparar el entorno de usuario del sistema donde se va a trabajar para que esté en perfectas condiciones para su ejecución. El sistema operativo Linux (que es el que va a ser utilizado para la realización de las prácticas) dispone de una serie de variables de entorno referentes a las distintas aplicaciones del sistema ya sea Java, nedit, Emacs, etc. donde están definidos, por ejemplo, los perfiles de usuario. En el caso de Java se define una variable de entorno denominada `CLASSPATH` que determina el directorio en el que se va a encontrar las clases (y packages) implicadas en la ejecucin de una clase Java.

Esta variable está definida por defecto cuando se instala Java, pero nosotros desde nuestro perfil de usuario podemos modificarla. Para que la modificación sea permanente (es decir, para que se active automáticamente al entrar en el sistema) podemos modificar el fichero `.bashrc`, o crearlo si no existe, situado en la raíz de nuestra cuenta:

```
[user@foo userHome] $ echo $HOME
/opt/home/user

[user@foo userHome] $ more .bashrc
#--------------------------------------
# Source global definitions (if any)
#--------------------------------------
if [ -f etc/bashrc ]; then
/etc/bashrc # Read /etc/bashrc if present.
fi

[user@foo userHome] $ nedit .bashrc

[user@foo userHome] $ more .bashrc
#--------------------------------------
# Source global definitions (if any)
#--------------------------------------
if [ -f etc/bashrc ]; then
/etc/bashrc # Read /etc/bashrc if present.
fi
export CLASSPATH = $CLASSPATH:.
```

En este ejemplo hemos añadido el directorio actual `.` a `CLASSPATH` de modo que permitimos que la máquina virtual localice siempre las clases del directorio actual.

1.3. Conceptos de Programación Orientada a Objetos

Antes de centrarnos en el lenguaje conviene recordar ciertos conceptos sobre POO pues la sintaxis de Java depende mucho de ellos.

La POO surge para evitar ciertos problemas que aparecían en programas estructurados. Por ejemplo, en el Centro de Cómputo Noruego en Oslo se desarrollaban simuladores de vuelo. Se comprobó que cuando se ejecutaban los simuladores las naves chocaban. Un análisis reveló que las aplicaciones confundían las características de diferentes objetos simulados. Es decir, la posición en el espacio o la velocidad de una nave eran atribuidas a otra.

La POO es una forma diferente de diseño de software que trata de imitar la realidad. En el fondo el mundo no es más que una colección de objetos que colaboran entre sí. Cada objeto tiene rasgos como su identidad (los objetos son diferenciables aunque no sea más que por su nombre), su estado (características que lo describen) y su comportamiento (qué puede hacer).

Normalmente nos referimos a objetos abstrayéndolos en clases. En este aula hay dos puertas. Obviamente son diferentes, pero con ambas se pueden hacer las mismas cosas. Para representar esto en un programa implementaríamos la clase `Puerta`. Para representar el estado (abierta o cerrada) usaríamos una propiedad `abiertaOCerrada`. Para su comportamiento implementaríamos dos métodos `abrir()` y `cerrar()`. Posteriormente

crearíamos dos instancias de la clse (dos objetos) con identificadores `puertaDeAlante` y `puertaDeAtras`.

La abstración es fundamental en la POO. Otra característica no menos importante es la encapsulación: un objeto puede hacer cosas propias, pero no cosas de otros. Por ejemplo no tiene sentido que la clase puerta tenga un método `entrarAlumno`.

1.4. Elementos básicos del lenguaje

Vamos a crear una primera clase de Java. Se trata de la versión estructurada del ya famoso `hello world`. Escriba en un editor de textos cualquiera el archivo `HolaMundo.java` con el siguiente código:

```
/**
 * Esta clase implementa una aplicacion que saca por pantalla
 * la frase Hello World
 */
/*  la linea de importaciones se incluye por motivos didacticos.
 *  En este ejemplo no es realmente necesaria
 */
import java.lang.*;
class HolaMundo {
    public static void main(String[] args){
        System.out.println(''Hola Mundo!!!!!!!!'');
                //System se importa de java.lang
        }
}
```

Para compilar esta clase tiene que llamar al compilador de java `javac` y pasarle el archivo como parámetro. Es decir, debe ejecutar el comando:

```
[user@foo userHome] $ javac HolaMundo.java
```

Obtendrá como resultado el archivo `HolaMundo.class` que contiene bytecodes de java. Para ejecutar este programa debe llamar a la máquina virtal y decirle que ejecute el archivo de bytecodes. Para ello ejecute:

```
[user@foo userHome] $ java HolaMundo
Hola Mundo!!!!!!!!
```

Nótese que a java se le indica que ejecute `HolaMundo.class` indicándole que ejecute `HolaMundo`. No debe ponerse la extensión. En ese caso se obtiene una excepción pues se está tratando de ejecutar otra cosa, como se explicará posteriormente.

El archivo que acaba de compilar y ejecutar contiene diferentes elementos del lenguaje como: identificadores, palabras reservadas, comentarios, sentencias, bloques de código, literales y metacaracteres. Vamos a estudiar un poco cada uno de ellos.

Identificadores y palabras clave: Se denominan identificadores a los nombres que les damos a las clases, los métodos y las variables. Un identificador de java es una secuencia de letras mayúsculas o minúsculas, dígitos y los caracteres _ y `$`. La secuencia no puede comenzar con un dígito. Cabe destacar que Java diferencia entre mayúsculas y minúsculas. Es decir, `Pepe` no es lo mismo que `pepe`.

Algunos identificadores deben cumplir ciertas normas. Por ejemplo, el nombre de la clase, `HolaMundo` en el ejemplo, debe coincidir, salvo la extensión, con el nombre del archivo que la contiene. Esto debe ser cierto para todas las clases públicas.

Aparte, hay ciertas convenciones que se deben respetar: Los nombres de las clases comienzan por mayúscula y el resto de nombres por minúscula. Si un nombre contiene más de una palabra, las iniciales de todas las palabras a partir de la segunda deben estar en mayúsculas (observe el nombre de la clase `HolaMundo`). Se recomienda que los nombres de clases sean sustantivos, los de métodos verbos y que las variables dejen claro su contenido.

Aunque los identificadores pueden contener los caracteres _ y `$`, lo habitual es que se reserven para situaciones concretas.

Java utiliza un ámplio abanico de palabras clave. Estas palabras son reservadas. Es decir, no se pueden utilizar como identificadores. Es el caso de las palabras `import`, `public`, `stataic`, `class` y `void` en el ejemplo.

Comentarios: En java existen tres tipos de comentarios. El ejemplo refleja dos de ellos. Por un lado tenemos los comentarios multilinea. Comienzan por `/*` y finalizan por `*/`. En medio puede aparecer cualquier cosa salvo, obviamente, la cadena `*/`. Para iniciar un comentario corto se emplea `//`; el comentario finaliza en el siguiente salto de línea.

El tercer tipo de comentario se denomina comentario de documentación. Comienzan por `/**` y finalizan por `*/`. Permite documentar el programa y utilizar posteriormente una herramienta de Java, denominada `javadoc` para generar archivos de documentación.

Sentencias: Son las órdenes que se deben ejecutar para que el programa se ejecute. Finalizan siempre con un `;`. Por ejemplo `System.out.println(''Hola Mundo!!!!!!!!'');`.

Bloque de código: Un bloque de código comienza por el metacaracter { y finaliza con el metacaracter }. En medio de estos metacaracteres nos encontraremos grupos de sentencias y otros bloques de código. Un bloque de código es una unidad lógica que puede utilizarse como una sentencia simple. Son uno de los mecanismos de encapsulamiento de Java.

Literales: Los literales son valores constantes dentro de la clase. Algunos ejemplos serían el literal entero `100`, el literal de coma flotante `98.6`, el literal caracter `'x'` o el literal cadena que aparece en el ejemplo `''Hola Mundo!!!!!!!!''`. Un literal puede aparecer en cualquier punto en que esté permitido un valor de su mismo tipo.

Metacaracteres: Existen otros caracteres que sirven para diferentes cometidos. Por ejemplo, los paréntesis se emplean para contener listas de parámetros, modificar precedencia de expresiones, contener comparaciones en ciertas sentencias, o en conversiones de tipo. Las llaves pueden usarse, además de para marcar bloques, para contener los valores iniciales de matrices o para marcar ámbitos. Los corchetes contienen matrices, tanto en su declaración como en su acceso, etc..

El metodo `main`: Toda aplicación java debe contener un método `main` cuya signatura es la que aparece en el ejemplo (puede alternarse, pero no está bien visto, el orden de `public` y `static`). Esta es la primera función que se ejecuta para correr el programa. La función tiene argumentos, una tabla de elementos de tipo `String`. El nombre de este argumento pueden ser el que usted quiera, pero es habitual que sea `args` o `argv`. Estos argumentos permiten escribir programas que admitan recibir información en linea de comando.

System.out.println: En Java es difícil programar algo que no use la API. En este caso estamos utilizando la clase `System` contenida en el paquete `java.lang`. Se explicará más adelante.

1.5. Pensando en objetos

El programa que hemos creado tiene varios defectos desde el punto de vista de orientación a objetos. Lo habitual es que cuando programe en Java piense en objetos, clases y las relaciones entre diferentes clases. Cuando creamos una clase nos fijamos en la funcionalidad que ofrece, no en su ejecución. Desde el punto de vista de la POO la clase que imprime el mensaje `Hola Mundo` en la pantalla no debe pensar en la ejecución del programa. Es decir, la versión orientada a objetos de este programa debería ser diferente. Escriba en un editor de textos cualquiera el archivo `HolaMundoObj.java` con el siguiente código:

```
/**
 * Esta clase que saca por pantalla la frase Hello World
 */
/*  la linea de importaciones se incluye por motivos didacticos.
 *  En este ejemplo no es realmente necesaria
 */
import java.lang.*;
public class HolaMundoObj {
    String saludo; //La clase String se importa de java.lang
    public HolaMundoObj() {
        saludo = ''Hola mundo''; //El constructor inicia las propiedades
    }
    public void mostrarSaludo(){
        System.out.println(saludo) //se importa de java.lang
    }
}
```

Para compilar esta clase tiene que llamar al compilador de java `javac` y pasarle el archivo como parámetro. Es decir, debe ejecutar el comando:

```
javac HolaMundoObj.java
```

Comprobará que obtiene un error. He olvidado el caracter ';' después de `(saludo)`. Fíjese en lo concreto que es el compilador indicando errores. Corrija el ejemplo y vuelva a compilar. Obtendrá como resultado el archivo `HolaMundoObj.class` que contiene bytecodes de java.

Si trata de ejecutar esta clase como en el ejemplo anterior obtendrá una excepción (compruébelo). El problema es que, como ya hemos dicho, cualqueir aplicación java necesita una función `main` (fíjese que es lo que le dice interprete de java). Antes de arreglar este problema vamos a analizar el código: El archivo que acaba de compilar contiene más elementos del lenguaje como expresiones y operadores.

Expresiones y operadores: Las expresiones son entidades formadas por diferentes miembros separados por operadores que los evalúan y relacionan. Por ejemplo `saludo = ''Hola Mundo''`.

Los operadores son signos especiales para ejecutar acciones específicas sobre valores que construyen nuevos valores. Java define operadores muy diversos que trabajan sobre un (unarios), dos (binarios) o tres operandos (ternarios). Se clasifican en operadores aritméticos, de comparación, lógicos, de desplazamiento y de asignación. Se explicarán más adelante

1.5.1. Estructura de un archivo Java

En Java siempre se programa en base a clases. Crearemos clases y utilizaremos otras clases externas que estén ya creadas. Para poder usar una clase debemos importarla. Para ello emplearemos `import` seguido del nombre de la clase que queramos importar. Hay una excepción a lo dicho. Los recursos mínimos necesarios para crear un programa se encuentran dentro del paquete de clases `lang` de la librería de java. El compilador importa siempre ese paquete aunque no se indique (como ya decía el comentario del programa de ejemplo).

Para entender bien un archivo java debe considerarse siempre el concepto de ámbito. Cualquier cosa que se declare/defina queda queda definida dentro de un ámbito concreto que suele coincidir con un bloque de código. Cuando importamos una clase es porque queremos utilizarla dentro del código de la clase que se está creando. Por ello, la importación debe hacerse fuera del bloque que define la clase, para que sea visible en todo el archivo que se está creando.

Después de importar las clases externas necesarios aparece el bloque de la clase. Para ello indicamos `class` seguido del nombre de la clase (que debe seguir las normas anteriormente explicadas) y de un bloque de código. Delante de la palabra `class` pueden aparecer diversos modificadores cuya función se explicará más adelante.

Dentro del bloque de la clase aparecen las propiedades (o atributos) de la clase seguidas de los métodos. El alcance de las propiedades es toda la clase. En nuestro caso se ha declarado la propiedad `saludo` de tipo `String`. Las propiedades se declaran como el resto de variables. Lo único que las diferencia es el lugar donde se declaran. Lo habitual es que las propiedades se inicialicen, aunque suele hacerse dentro de un método concreto como se verá más adelante. En la declaración de una propiedad también pueden aparecer modificadores que afectan a su visibilidad.

Los métodos se declaran estableciendo en primer lugar su nivel de acceso (`public` en el ejemplo) y luego el tipo de valor que devuelven (`void` si no devuelven nada), su nombre y una lista de parámetros, quizá vacía, entre paréntesis. A continuación va el bloque de código del método.

Cualquier clase debe presentar un método que permita construir objetos (denominados normalmente instancias) de la clase. Se denomina el método *constructor* de la clase. El

constructor es un método muy particular. Se limita a construir una instancia de la clase, por lo que no precisa indicar el tipo que devuelve. Por otro lado su nombre es siempre el mismo que el nombre de la clase (`HolaMundoObj` en nuestro caso).

1.6. Paquetes

Una aplicación Java se construye utilizando un conjunto de clases. Algunas externas o predefinidas y otras creadas *ex-proceso* para la aplicación. Para distribuir el programa se emplea el concepto de *paquete*, que es otro mecanismo de encapsulamiento de Java. Un paquete contiene un conjunto de clases que tienen una alta relación entre ellas, bien por constituir una aplicación o bien por ser clases que relaciona la lógica programada.

Vamos a crear un paquete `holamundo` para nuestra aplicación. El paquete contendrá la clase anteriormente definida.

Para construir un paquete desde un conjunto de clases basta con escribir al principio de cada uno de los archivos que lo componen `package xxx` donde `xxx` es el nombre del paquete. En nuestro caso el paquete holamundo contendrá la clase que se obtiene al compilar el archivo HolaMundoObj.java que contiene:

```
/**
 * Esta clase que saca por pantalla la frase Hello World
 */
/*  la linea de importaciones se incluye por motivos didacticos.
 *  En este ejemplo no es realmente necesaria
 */
package holamundo;
import java.lang.String;
import java.lang.System;
public class HolaMundoObj {
    String saludo;
    //Creamos el constructor de la clase
    public HolaMundoObj() {
        saludo = ''Hola mundo''; //El constructor inicia las propiedades
    }
    public void mostrarSaludo(){
        System.out.println(saludo)
        }
}
```

Cree a continuación un directorio llamado igual que el paquete y mueva dentro el archivo .class que resulta de la compilación.

Vamos ahora a crear un nuevo paquete que contenga una clase para ejecutar un objeto de la clase `HolaMundoObj`. Para ello cree el archivo `EjecutorHolaMundoObj` con el siguiente contenido:

```
/**
 * Paquete para ejecutar el Hola Mundo
```

```
 */
package util;
import holamundo.HolaMundoObj;
public class EjecutorHolaMundoObj {
    HolaMundoObj hola;
    public EjecutorHolaMundoObj(){
        hola=new HolaMundoObj();
        hola.mostrarSaludo();
    }
    public static void main(String[] args) {
        EjecutorHolaMundoObj ejecutor=new EjecutorHolaMundoObj();
    }
}
```

Si compila este archivo y trata de ejecutarlo descubrirá que aún no es posible. La máquina virtual le dice que hay un problema con el nombre `util/EjecutorHolaMundoObj`. El problema viene de que su archivo `.java` indicaba que esta clase formaba parte del paquete `util`. El interprete de java necesita que cada archivo esté donde se supone que debe estar y con el nombre que se espera. Un paquete es, para java, un directorio que contiene archivos que son clases. Por lo tanto si ejecutamos desde el directorio actual una clase que forma parte de un paquete se espera que tengamos un subdirectorio que se llame como el paquete con un archivo `.class` que se llame como la clase a ejecutar. Es decir, para ejecutar el programa tiene dos opciones:

- Comente la línea `package util;` y vuelva a compilar. Ejecute el archivo resultante.
- Cree un directorio cenominado `util` y mueva a dicho directorio la clase que habíamos creado (`EjecutorHolaMundoObj.class`). Ahora podrá ejecutar con el comando `java util/EjecutorHolaMundoObj`

Vamos a analizar la ejecución en ese segundo caso. Al ejecutarse el comando la máquina virtual busca en el subdirectorio `util` del directorio actual el archivo de bytecodes `EjecutorHolaMundoObj.class`. Como lo encuentra busca el principio de la función `main`. Esta función comienza declarando que existe un objeto llamado `ejecutor` que es de la clase `EjecutorHolaMundoObj`. La declaración del nombre sólo indica que a través de dicho nombre se puede acceder a objetos de la clase `EjecutorHolaMundoObj.class`, no que existan. Para que se cree (instancie en el lenguaje de objetos) un objeto de la clase debe ejecutarse la sentencia `new`. Esta sentencia se emplea para instanciar cualquier objeto y va seguida de la función constructora de la clase que se desea instanciar. Por tanto `EjecutorHolaMundoObj ejecutor=new EjecutorHolaMundoObj();` está asociando a `ejecutar` una nueva instancia de la clase `EjecutorHolaMundoObj`. Mediante new no sólo se instancia el objeto, también se ejecuta su método constructor. En nuestro caso dicho método declara la existencia de un objeto `hola` (nótese que no ha hecho falta decir de que tipo, ya se explicará porqué) al que se le asocia mediante new una instancia de la clase `HolaMundoObj`. Nótese que dicha clase se ve desde la que se está ejecutando pues se ha importado. A partir de aquí el interprete debe localizar la nueva clase. Para ello trata de localizar el archivo `HolaMundoObj.class` dentro del subdirectorio del directorio actual llamado `holamundo`. Como lo encuentra, comienza a interpretar el método constructor

de dicha clase. Dicho método se limita a definir la propiedad `saludo` del objeto creado. Posteriormente, el interprete debe seguir ejecutando el método desde donde se hizo la llamada. Es decir, la instrucción `hola.mostrarSaludo();` . Esta sentencia significa que se está llamando (invocando en el mundo de objetos) al método `mostrarSaludo` del objeto `hola`. Dicho método existe, así que se ejecuta, lo que, finalmente, imprime el mensaje.

1.7. Niveles de acceso

Habrá observado que en los ejemplos, delante de los métodos, puede ir la palabra reservada `public`. Esta palabra reservada, junto con otras, indica desde donde es visible (y por tanto se puede invocar) el método. Se emplea el mismo conjunto de palabras para indicar niveles de acceso a clases y propiedades. Las posibilidades son las siguientes:

- El modificador `public`: Puede usarse para clases o para propiedades/metodos de la clase. En todos los casos significa que el elemento sobre el que se aplica es accesible desde cualquier otra clase.
- El modificador `protected`: Se emplea sólo para propiedades y métodos. Los elementos `protected` son sólo visibles/accesibles para clases que formen parte del mismo paquete de la clase donde ellos están. Es decir, si hubiésemos indicado que el método `mostrarSaludo` de `HolaMundo` era `protected`, el ejecutor no hubiese podido funcionar (a no ser, claro está) que lo hubiésemos metido en el mismo paquete que la clase `HolaMundo`. Compruébelo.
- El modificador `private`: Se emplea para métodos y propiedades. Los elementos `private` sólo pueden ser accedidos por métodos de la clase donde se declaran.
- Podemos no tener ningún modificador. En este caso estamos jugando con la visibilidad por defecto (denominada *package-privative*). Los elementos *package-privative* son parecidos a los protected. Son accesibles por elementos de la clase donde se declaren o por otras clases del mismo paquete. La única diferencia es en el caso de que la clase que los contenga tenga subclases (se explicarán). En tal caso (y a diferencia de `protected`) el elemento no es accesible desde las subclases.

Es su deber como programador de java decidir con cordura los niveles de acceso que permite. Lo habitual es que las propiedades sean `private` (salvo las constantes). Si es necesario que alguien externo a la clase pueda ver la propiedad se crea un método público que facilite el trabajo. En cuanto a los métodos lo habitual es hacer público sólo lo estrictamente necesario. un caso particular es el método `main`. Obviamente debe ser visible desde la máquina virtual, por lo que debe ser público.

Compruebe como afectan modificaciones de visibilidad a los ejemplos desarrollados.

Obviamente, para acceder a una propiedad o un método de un objeto previamente ese objeto debe existir. Es decir, debe haberse instanciado. Como hemos visto invocamos un método de un objeto que existe poniendo `nombre_del_objeto.metodo()`. De igual modo accedemos a una propiedad poniendo `nombre_del_objeto.propiedad`.

Podemos tener métodos que son accesibles sin necesidad de instanciar un objeto. Esto sucede cuando tenemos métodos de clase. Existen aunque no haya objetos de la clase. Para hacer que un método sea de clase ponemos `static`. El ejemplo más típico es el

del método `main`. Este método debe ser invocado por la máquina virtual antes de que se instancie ningún objeto de la aplicación. Es por ello que siempre debe ser `static`.

También podemos tener propiedades `static`. Se denominan variables de clase.

1.8. En resumen, organización

Una conclusión evidente de todo lo dicho es que una aplicación java debe construirse siguiendo férreos criterios de organización. Los nombres de los ficheros que contienen clases no son libres. Y mucho menos el lugar dentro del sistema de archivos donde pueden estar las clases. En Java la organización del programa forma parte de la programación en sí misma. Debe respetar al pié de la letra todo lo que se le indique de organización y nombrado. Si no, el programa siplemente puede no funcionar a pesar de estar bien escrito.

Los IDE que permiten programar en Java, como NetBeans, tienen en cuenta estas necesidades de organización. En muchos casos incluso se encargan ellos de crear los directorios necesarios y llevar a ellos las clases compiladas.

1.9. Ejercicios

Para poder comenzar a manejar Java como un lenguaje de programación más, se procederá a la implementación de una serie de clases que van a permitir una primera toma de contacto con los dispositivos de entrada y salida estándar del sistema así como tomar datos introducidos desde la línea de comandos.

La siguiente clase calcula si un año es bisiesto o no. Un año se considera bisiesto si es divisible por 4 y no lo es por 100 o si lo es, también es divisible por 400.

```
public class AnhoBisiesto{
    public static void main(String[] args){
        // Lee el primer argumento pasado por la linea de comandos
        int anho = Integer.parseInt(args[0]);
        boolean bisiesto;
        bisiesto = ((anho % 4 == 0) && (anho % 100 != 0)) || (anho % 400 == 0);
        if(bisiesto){
            System.out.println(''El año '' + anho + '' es bisiesto'');
        }
        else {
            System.out.println(.El año ''+ anho + '' no es bisiesto'');
        }
    }
}
```

Esta clase presenta como novedad que lee el parámetro que se pasa como argumento. Es muy parecido a C, salvo que en este caso los argumentos se cuentan a partir del nombre de la clase que se ejecuta. Como por ejemplo:

```
[user@foo prac1] java AnhoBisiesto 1998
El año 1998 no es bisiesto
```

Otra forma de pedir la introduccin de datos es a través de la consola del sistema. En Java, para leer de una fuente de datos, bien sea de un fichero, o el teclado o de otro periférico es necesario construir tres objetos que denominaremos:

- Canal: identifica la fuente de datos, en este caso de la consola del sistema se identifica con el objeto `System.in`.
- Flujo: transfiere la informacin a través de un array de bytes indiferenciados, es decir, no asociados a ningún tipo simple ni a ninguna clase.
- Filtro: hace posible la interpretacin de los bytes y los convierte en tipos primitivos o en cadenas de caracteres.

Aplicado a un ejemplo que lee el nombre introducido por teclado tendría la siguiente implementación, es importante notar que hay que importar el paquete `java.io`:

```
import java.io.*;
public class DameTuNombre{
    public static void main(String[] args) throws IOException {
        String nombre;
        //InputStreamReader recoge bytes de la consola al programa
        //BufferedReader interpreta los bytes que llegan de flujo
        InputStreamReader flujo;
        BufferedReader teclado;
        //Inicializacion de las variables
        flujo = new InputStreamReader(System.in);
        teclado = new BufferedReader(flujo);
        System.out.println(''Introduce tu nombre = '');
        nombre = teclado.readLine();
        System.out.println(''Tu nombre es '' + nombre);
    }
}
```

Este programa ejecutado en la consola del sistema produciría la siguiente salida por la misma:

```
[user@foo prac1] java DameTuNombre
Introduce tu nombre =
Ignacio
Tu nombre es Ignacio
```

El problema es más complejo, aunque no demasiado, si el dato de entrada no se trata como una string (Java proporciona un modo muy sencillo de interpretar los datos que se leen desde un flujo de bytes, como se verá en siguientes prácticas). Podemos también modificar la clase `AnhoBisiesto` para que pida al usuario la introducción del año correspondiente; en Java la lectura es fácil ya que permite la conversión de String a int de manera casi inmediata.

```
public class AnhoBisiesto2{
    public static void main(String[] args) throws IOException {
        //Inicializ. de las variables
        int anho;
        boolean bisiesto;
        String textoAnho;

        BufferedReader teclado = new BufferedReader(new
                            InputStreamReader(System.in));
        System.out.print(''Introduce el año en estudio: '');
        textoAnho = teclado.readLine();
        Integer intAnho = Integer.valueOf(textoAnho);
        anho = intAnho.intValue();
        bisiesto = ((anho% 4 == 0) && (anho% 100 != 0)) || (anho% 400 == 0);
        if(bisiesto){
            System.out.println(''El año '' + anho + '' es bisiesto'');
        }
        else{
            System.out.println(''El año '' + anho + '' no es bisiesto'');
        }
    }
}
```

La ejecución de la clase produce la salida por pantalla siguiente:

```
[user@foo prac1] java AnhoBisiesto2
Introduce el año en estudio
1998
El año 1998 no es bisiesto
```

Ejercicio 1: Realizar un programa en Java que solicite al usuario un número entero positivo, calcule su cubo y lo muestre al usuario de la siguiente manera:

```
[user@foo prac1] java Cubo
Escribe un número entero positivo: 9
El cubo de 9 es 729
```

Ejercicio 2: Modifique la clase anterior de modo que el valor se pase por línea de comando.

Ejercicio 3: Escriba un programa que simule el lanzamiento de un par de dados. Se puede simular el lanzamiento de un dado mediante la elección de un entero comprendido en el conjunto {1, 2, 3, 4, 5, 6} aleatoriamente. En Java se puede simular mediante la invocacin del método de clase `random()` de la clase `Math`:

`(int)(Math.random()*6) + 1`

Puedes asignar este valor a una variable para representar el valor del dado lanzado. Se repite el proceso para el segundo dado. A continuación, se sumarán ambos valores para mostrar por pantalla el resultado total. Por ejemplo:

```
El lanzamiento del primer dado es 3
El lanzamiento del segundo dado es 5
El total de los dos lanzamientos es 8
```

Hay una jugada denominada *ojos de tigre* que consiste enobtener 2 unos. En caso de que sea la jugada obtenida el programa debe felicitar al jugador.