FACULTAD DE INGENIERÍA UNAM
DIVISIÓN DE EDUCACIÓN CONTINUA

CURSOS INSTITUCIONALES

JAVA BÁSICO

Del 07 al 18 de Octubre del 2002

APUNTES GENERALES

CI-390

Instructor: Ing. Rodolfo González Maldonado

CONSEJERÍA JURIDICA
OCTUBRE DEL 2002

Introducción

Java es un lenguaje de programación orientado a objetos. Esto hace que tenga una serie de particularidades que comparte con otros lenguajes orientados a objetos y que afectan no sólo a su sintaxis, sino sobre todo al enfoque que el programador puede y debe dar al trabajo para desarrollar aplicaciones.

En los próximos apartados se intentan resumir las características de la programación orientada a objetos. No se trata de realizar una explicación extensiva de la programación orientada a objetos, sino de proporcionar un breve resumen de sus rasgos mas importantes. Los lectores con conocimientos de programación orientada a objetos pueden pasar por alto estos apartados y concentrarse en los siguientes, que ya entran de lleno en los aspectos clave de la programación en Java.

La programación orientada a objetos

La clave de la programación orientada a objetos se encuentra en la palabra *objeto*. Objeto, dentro del mundo de la programación, designa a un conjunto de variables y métodos relativos a esas variables.

Los objetos pretenden ser una abstracción de lo que sucede en el mundo real. En el mundo real todo lo que percibimos son objetos. La silla sobre la que nos sentamos es un objeto, el bolígrafo con el que se escribe también es un objeto, la PC .... todos son objetos. Cada uno de esos objetos tiene una serie de propiedades (La mesa esta definida por un tamaño, composición, etc.) y tienen unos comportamientos ante determinadas acciones.

A la hora de programar, las propiedades de un determinado objeto se recogen en *variables o* campos de información. Así mismo, si modelamos con software un objeto mesa, podemos recoger sus propiedades (color, antigüedad, etc.) en unas variables determinadas. Asimismo, el comportamiento del objeto se implementa en *métodos*.

Nota

Al proceso formal de reunir en una misma entidad -el objeto- variables y funciones es a lo que se denomina encapsulación, y constituye una de las características fundamentales de la programación orientada a objetos.

NOTAS:

Las ventajas de la encapsulación son dos, tal y como se muestra a continuación:

- ***Modularidad*** del software. Al ser los objetos las entidades claves de programación, cada objeto puede ser modificado y mantenido por separado.

- ***Ocultamiento*** de la información. Con la programación orientada a objetos podemos mantener métodos y variables que no son accesibles desde fuera del objeto, con lo que eliminamos posibilidades de error a la hora de programan

Las clases

La mesa en la que me encuentro trabajando en este preciso momento es un objeto, mientras que la mesa en la que estas trabajando tu es distinta. Sin embargo, ambos objetos comparten una serie de propiedades y de comportamientos. A pesar de ser objetos distintos que pueden tener unas apariencias muy diferentes, todo el mundo es capaz de reconocer ambos objetos como mesas, saber para que sirven y que características son relevantes en ambos. Si tratásemos de expresar esto en términos de programación orientada a objetos, tendríamos que decir que lo que sucede es que ambos objetos son *instancias* de una *clase* de objetos denominada mesa.

Una clase no es masque la idealización o prototipación de todas las propiedades y comportamientos comunes a ciertos objetos. Otra forma de definir una clase es decir que es la abstracción de las variables y métodos comunes a ciertos objetos.

Un clase vendría a ser el conjunto de variables y de acciones que nos sirven para definir un conjunto de objetos. Así, la mesa de mi despacho sobre la que escribo es una instancia de la clase mesa.

Nota

La palabra instancia proviene del inglés instance. Tradicionalmente se ha venido traduciendo como instancia, pero quizás hubiese sido más correcto y más intuitivo traducirlo como ejemplo o muestra.

Otro punto que es conveniente ilustrar es la diferencia entre variables y métodos de la clase y variables y métodos de una instancia de esa clase.

NOTAS:

Es fácil comprenderlo con un ejemplo: existen propiedades que la mesa de mi despacho comparte con todas las mesas del mundo: tiene unos elementos sustentadores y una plataforma superior. Sin embargo, mi mesa tiene propiedades que no son compartidas por todas las mesas: no todas tienen cajones, como mi mesa. En la terminología orientada a objetos, los cajones son variables del objeto mesa de mi despacho, mientras que las patas son variables de cualquier objeto mesa, y por lo tanto se dice que son variables de la clase mesa.

Los mensajes

Los objetos en programación, al igual que en el mundo real, disponen de propiedades y comportamientos públicos o accesibles desde fuera del objeto y de propiedades y comportamientos internos o privados. A menudo al conjunto de variables y métodos públicos se le llama *interfaces públicas*. Cuando un objeto se quiere comunicar con otro, debe hacerlo a través de su interface pública enviándole un *mensaje*. Como todo mensaje, éste contiene el nombre del objeto con el que se quiere comunicar, así como la operación o método que desea que el objeto realice y aquellos posibles *argumentos o* datos que el objeto necesita para responder a la comunicación.

Supongamos el objeto Jose, de la clase Persona, que quiere comunicares con el objeto Paco, de la clase ordenador. Como todo objeto de la clase Ordenador, Paco tiene una serie de variables, como son la variable estado (si esta encendido, apagado o -como sucede alguna vez- simplemente se ha quedado colgado), y una serie de métodos, entre los que se encuentra el método leeDisco, que sirve para leer información de una unidad de disco. Si el objeto José quiere examinar el contenido de un disco flexible, deberá lanzar un mensaje a Paco (el objeto con el que quiere comunicarse), el método que ese objeto debe efectuar (en este caso leeDisco) y los argumentos necesarios para que el objeto ejecute correctamente el método (en nuestro caso, especificarle que se trata de la unidad de disco flexible). Avanzando conceptos, este mensaje se expresaría en Java de la siguiente manera:

```
Paco.leeDisco( discoFlexible );
```

Naturalmente, para que Jose pueda enviar ese mensaje a Paco, el método leeDisco debe pertenecer a la interfaz pública de la clase Ordenador.

NOTAS:

La herencia

Otra propiedad que permite implementar la programación orientada a objetos es la *herencia*. Las clases del mundo real pueden quedar descompuestas en clases mas simples, o pueden ser agrupadas formando una clase mas compleja. La clase MesaDeDibujo esta, sin duda, relacionada con la clase Mesa, mas general. Podríamos decir que la clase MesaDeDibujo es una subclase de la clase Mesa. Tiene todas las propiedades y comportamientos de la clase Mesa y también tiene otras propiedades específicas que no comparten todos los objetos de la clase mesa.

La programación orientada a objetos permite que una clase pueda heredar propiedades y comportamientos de otra clase.

A menudo, la clase padre es conocida como superclase de la clase hija, mientras que la clase hija se denomina subclase de la clase padre.

Nota

Algunos lenguajes de programación orientada a objetos, como C ++ , permiten que una clase proceda de varias superclases. A esta propiedad se la denomina herencia múltiple. Este mecanismo, aunque algunas veces es de utilidad, puede introducir una gran complejidad en el código. Por esta razón Java no implementa la herencia múltiple. Cada subclase proviene solamente de una superclase.

La herencia también permite modelar unas clases genéricas que se denominan *clases abstractas.* Esta técnica se conoce con el nombre de *abstracción* y es otra característica fundamental de la programación orientada a objetos.'

Supongamos, por ejemplo, que queremos modelar mediante un lenguaje de programación orientado a objetos el mundo de las moscas y de las abejas. Una forma de poder hacerlo sería la siguiente: declaramos una clase general que denominaremos insecto, en la que se definirán los comportamientos y propiedades comunes tanto a moscas como a abejas, y posteriormente crearemos las clases Abeja y Mosca como subclases de la clase Insecto.

NOTAS:

Entre estas tres clases hay una sutil diferencia: en el mundo real sí existen instancias de la clase Abeja y de la clase Mosca, pero en cambio no nos encontraremos jamas instancias de la clase Insecto. La clase Insecto no es mas que una clasificación abstracta, realizada en nuestra mente, pero que no se corresponde a objetos concretos del mundo real.
En términos de programación orientada a objetos, la clase Insecto seria lo que se denomina una clase abstracta. Se trata de una clase que no tiene traducción en ningún objeto concreto.

Junto con la encapsulación y la herencia, existe una tercera característica que constituye la base fundamental de la programación orientada a objetos. Esta propiedad se conoce como *polimorfismo.*

Los constructores de clases

Hemos comentado anteriormente que una clase se caracteriza por unas variables de clase y por unos métodos de clase. Existen unos métodos especiales cuya misión es exclusivamente la de crear una instancia de la clase. A estos métodos se les denomina *constructores.* Por convención, tienen el mismo nombre de la clase y son una especie de inicializadores de dicha clase.

NOTAS:

Características generales de los programas en Java

E1 ejemplo de applet en Java que hemos visto en el capitulo anterior puede servirnos para ilustrar las características generales de las aplicaciones en Java:

```
import java.applet.Applet;

import java.awt.*;

//Primer applet en Java
//Fecha inicio: 1/1/2001
//Definición de la clase Bienvenida
public class Bienvenida extends Applet {


    public void paint( Graphics g ) {

    //mensaje de bienvenida

    g.setColor( Color.red );

    g.drawString("Bienvenido al mundo del JAVA",50,25);

    g.setColor( Color.black );

     g.drawString ("Esto es el principio",50,40)
    }
}
```

Al igual que sucede con el lenguaje C, una línea de programación viene determinada por lo que hay escrito entre el signo ;

Cuando una línea comienza por dos barras inclinadas hacia la derecha, el compilador de Java ignora lo que hay escrito a continuación hasta el próximo retorno de carro. Es, por tanto, una forma de introducir comentarios en la aplicación Java. Así se han introducido en el programa anterior las fechas de inicio y de final del applet en Java.

Nota *Otra alternativa para introducir comentarios es comenzar la línea* ***o*** *líneas de comentarios con la barra inclinada hacia la derecha y el asterisco. El compilador ignorará todo lo escrito a continuación hasta que se encuentre los caracteres asterisco y barra inclinada a la derecha. De esta forma se ha incluido el comentario <<CURSO BASICO DE JAVA>>.*

NOTAS:

Variables en Java

A continuación nos ocuparemos de las variables en Java. Como cualquier otro lenguaje de programación, Java se sirve de variables para almacenar información.. Las variables no son mas que contenedores de datos, datos que pueden cambiar o no a lo largo de la aplicación.
Las variables en Java se definen por tres rasgos: el *nombre*, el *tipo* y el *alcance* de la variable.

- El nombre de la variable es el identificador de la variable a lo largo de la aplicación.
- El tipo de una variable hace referencia a los valores que se pueden almacenar en la variable y a su tamaño.
- El alcance de una variable es la parte de código donde la variable tiene significado. Fuera de su alcance, la variable no tiene significado y por lo tanto no es accesible: el alcance también determina cuando una variable se crea y se destruye.

De esta forma, a modo de resumen, cuando escribimos la línea public int numeroDeVeces;, lo que estamos haciendo es declarar una variable de nombre numeroDeVeces de tipo entero (que puede almacenar únicamente valores enteros) y que es creada al comienzo de la clase donde es declarada.

Nombres de variables

No hay reglas a la hora de ponerle nombre a una variable. Las únicas limitaciones son las siguientes:

- No se pueden incluir espacios en blanco.

- Existen una serie de palabras reservadas en Java: se trata de palabras que forman parte del lenguaje Java y que ya disponen de un significado determinado, por lo que no se pueden utilizar para denominar a una variable. Por ejemplo, hemos visto en un ejemplo anterior que int identifica a una variable de tipo entero. Pues bien, como int es una palabra que forma parte de Java, esta reservada y no puede utilizarse para denominar a una variable.

NOTAS:

- No se pueden declarar dos variables con el mismo nombre y el mismo alcance. La razón parece obvia: el compilador no podría saber a cual de las dos variables nos referimos y por lo tanto no lo traduciría a bytecodes.

Junto alas limitaciones anteriores, existen una serie de convenciones que se han adoptado en la practica en aras de la legibilidad del código. La mayoría de ellas provienen de lo que se ha denominado notación húngara, y son las siguientes:

- Las variables deben comenzar por una letra minúscula. Las clases, por el contrario, deberían comenzar por una letra mayúscula.

- Cuando el nombre de una variable sea el resultado de la combinación de varias palabras (el ejemplo de la variable anterior numerodeVeces), todas esas palabras deben quedar agrupadas en una sola, que será el *producto* de la unión de todas y en la que cada palabra posterior a la primera comience por mayúscula.

Consejo

Los nombres de las variables deben ser descriptivos de la variable a la que representan. Aunque la natural tendencia al mínimo esfuerzo nos sugiera nombres cortos para nuestras variables, el utilizar nombres no suficientemente descriptivos hace que se reduzca la legibilidad del código y dificulta su depuración.

NOTAS:

Tipos de variables

Los tipos de variables que se utilizan en Java son los que se relacionan en la siguiente tabla

Tipo	Valores que puede tomar
Byte	Octeto de bits.
Short	Número entero de 16 bits (de 0 a 216).
Int	Número entero de 32 bits (de 0 a 232).
Long	Número entero de 64 bits (de 0 a 264).
Float	Número de coma flotante de 32 bits.
Double	Número de coma flotante de 64 bits.
Char	Carácter ASCII.
boolean	Valor true o false.

Nota

A los programadores en C/C++ los tipos de variable utilizados en Java les recordarán mucho a aquellos que utiliza C o C++. Es oportuno recordar, sin embargo, que hay tres tipos de datos típicos de C y C++ que no se implementan en Java, como son los punteros, las uniones (union) y las estructuras de datos (struct).

Alcance de las variables

El alcance de una variable queda determinado según el lugar concreto donde esa variable es declarada. Según el alcance de las variables, en Java existen cuatro clases de variables:

NOTAS:

- ® ***Variables de clase***: pertenecen a la clase o al objeto instancia de la clase y por lo tanto perviven lo que la clase y el objeto.
- ® ***Variables locales***: se declaran dentro de los métodos de una clase y su ámbito de significación es ese método. Fuera de é1 no existen.
- ® ***Argumentos***: como ya se ha dicho, se trata de los datos que requiere un método para llevar a cabo su tarea. También es el método su ámbito de pervivencia.
- ® ***Parámetros*** de manejo de excepciones: son muy similares a los argumentos. De hecho, son los argumentos que los maneadores de excepciones usan. Sobre este tipo de variables se volverá más adelante, al explicar el uso de excepciones en Java.

NOTAS:

Operadores en Java

Los operadores son elementos de programación que, como su nombre indica, realizan una operación determinada y devuelven un resultado. Por ejemplo, el símbolo + en la siguiente expresión:

variableAlfa + variableBeta

es un operador que, como veremos, agrega los contenidos de las variables alfa y beta y devuelve el resultado de la suma. Tradicionalmente, los operadores se clasifican en cinco grupos:

- Aritméticos.
- Relacionales.
- Lógicos.
- De bit.
- De asignación.

Nota

Los operadores pueden requerir de uno o dos operados para realizar la operación. En el primer caso se dice que son operadores unarios, mientras que en el segundo caso se les denomina operadores binarios.

OPERADORES ARITMÉTICOS

Estos operadores realizan las funciones aritméticas mas sencillas sobre operados enteros o flotantes.

NOTAS:

+	a+b Devuelve el resultado de sumar a y b.
-	a-b Devuelve el resultado de restarle b a a.
*	a*b Devuelve el resultado de multiplicar a por b.
/	a/b Devuelve el resultado de dividir a entre b.
%	a%b Devuelve el resto de dividir a entre b

Junto a estos operadores, todos ellos binarios, existen el operador unario ++, que incrementa el operando en una unidad, y el operador unario --, que decrementa el operando en una unidad. Cada uno de estos operadores tiene dos versiones, delante o detrás del operando:

contado ++ ; ++contador;

Existe una importante diferencia entre ambas sentencias. Si contador vale en este momento 3 y se ejecuta la sentencia:

numeroDeVeces = contador ++ ;

la variable numeroDeVeces toma el valor 3. Sin embargo, si la sentencia que se ejecuta es la siguiente:

numeroDeVeces = ++ contador;

la variable numeroDeVeces toma el valor 4. Naturalmente, al final de ambas sentencias el valor de la variable contador será 4.

Nota

Cuando actúa sobre dos cadenas de caracteres, el operador + implementa la concatenacı6n de las cadenas.

NOTAS:

OPERADORES RELACIONALES

Los operadores relacionales determinan la relaci6n existente entre dos operandos. E1 valor que devuelven es true en el caso en que la relación sea la especificada y false si no lo es.

La expresión 3 > 4 devolverá un valor false.

Los operadores relacionales se muestran en la tabla:

>	a>b
>=	a >= b
<	a<b
<=	a <= b
==	a==b
!=	a!=b

NOTAS:

Operadores lógicos

La función de estos operadores es realizar las operaciones lógicas and, or y not. Cuando a y b son operadores booleanos, los operadores && y || pueden ser sustituidos por & y I, respectivamente.

&&	a && b	Devuelve true si son ciertos a y b
II	allb	Devuelve true si son ciertos a o b
I	!a	Devuelve true si a es false.

Operadores de bit

Los operadores de bit permiten manipular los bits de los datos directamente. Los operadores que proporciona Java para la manipulación de bits se muestran en la siguiente tabla :

>>	a >> b Desplaza los bits de a hada la derecha una distancia a.
<<	a << b Desplaza los bits de a a la izquierda una distancia b.
<<<	a <<< b Igual que el anterior, pero sin signo.
&	a&b Suma lógica (operación and) de a y b.
I	alb Operación or de a y b
^	a ^ b 0peración xor (or exclusivo) de a y b.
~	~a Devuelve el complementario de a.

NOTAS:

Operadores de asignación

Estos operadores se utilizan para que podamos cambiar el valor de una variable. El operador de asignación básico es el =. Este operador asigna el valor que encuentra a su derecha al operador que esta a la izquierda.

Nota

Cuando hemos declarado una variable pero todavía no le hemos asignado Ningún valor, el valor que adopta por defecto es la palabra reservada null.

Basados en el operador básico, Java ofrece otros operadores para poder acortar operaciones de asignación comunes. Estos se pueden observar en la siguiente tabla:

+=	a+=b	a=a+b
-=	a-=b	a=a-b
*=	a*=b	a=a*b
/=	a/=b	a=a/b
%=	a%=b	a=a%b
&=	a&=b	a=a&b
|=	a|=b	a=a|b

Estos operadores constan del operador de asignación y de un operador aritmético o lógico. Primero evalúan el operador aritmético o lógico y posteriormente el operador de asignación.

NOTAS:

Precedencia de operadores

Cuando en una expresión aparece mas de un operador hay que discernir cual de ellos se ejecutara primero, ya que en la mayor parte de los casos el orden en el que se ejecuten va a alterar el resultado. En la expresión 7 + 3 * 2 si se ejecuta primero el operador + el resultado es 20, mientras que si se ejecuta el operador * el resultado será 13.

Para solventar este problema, existe lo que se llama la *precedencia de operadores*. La precedencia de operadores no es mas que un conjunto de reglas que expresan el orden en el que las distintas operaciones de una misma sentencia se ejecutan. La precedencia de operadores en Java se lista en orden descendente:

Operadores	Descripción
[ ]	Posición de un elemento en un array Referencia a un método o variable de una clase
++ ___ ~ !	Operadores que preceden al operando
New	Creación de instancia
* / %	Multiplicación, división y resto de división entera
+ -	Suma y resta
<< >> >>>	Desplazamiento de bits
< <= > >=	Mayor, mayor o igual, menor, menor o igual
== !=	Igualdad o desigualdad
&	Operador de bit and
^	Operador de bit xor
|	Operador de bit or
&&	Operador lógico and
||	Operador lógico or
= += . .	Operadores de asignación

NOTAS:

Independientemente del orden que le imponga la precedencia de operadores, esta precedencia puede alterarse mediante el uso de paréntesis.

Cuando se utilizan paréntesis, la expresión comienza a ejecutarse por los paréntesis mas internos a la expresión.

NOTAS:

Implementación de objetos en Java

Si quiero implementar un objeto en Java, hay que tener en cuenta dos pasos:

- Declarar el objeto.
- Crear el objeto.

La declaración de objetos

La declaración de un objeto se realiza igual de si de otra variable se tratase. La forma general es:

ClaseDelObjeto nombreDelObjeto;

La declaración del objeto es una operación que sirve para indicar al compilador que en lo sucesivo el nombre nombreDelObjeto sirve para hacer referencia a una instancia -la cual, por cierto, no ha sido aún creada- de una clase GlaseDelObjeto. La expresión Date fechaNacimiento; declara un objeto de la clase Date al que en lo sucesivo se pasara a denominarse: fechaNacimiento. La clase Date es una clase que nos posibilita el manejo de fechas y sobre la que volveremos a tratar para conocer mas detalles en los capítulos siguientes.

Creación de objetos en Java

La declaración de un objeto no supone, como ya se ha indicado anteriormente, la creación del propio objeto. Dicho de otra manera mas simple, la expresión anterior no nos indica que exista ningún objeto Date denominado fechaNacimiento, simplemente nos esta indicando que existe una variable fechaNacimiento que esta preparada para recibir un objeto de la clase Date.

NOTAS:

La creación del objeto Date correspondiente se llevaría a cabo con la siguiente expresión:

fechaNacimiento = new Date();

De forma mas general, la creación de un objeto en Java sigue la siguiente pauta:

nombreObjeto = new *ClaseDelObjeto () ;*

La palabra reservada new es un operador que reserva la memoria correspondiente al objeto. Para realizar esto, requiere como operando el constructor de la clase. Como es bien sabido, este constructor va a inicializar el objeto, para lo cual puede necesitar algún argumento. En la realidad nos vamos a encontrar a veces con una determinada clase que va a tener varios constructores y cada uno de ellos inicializará el objeto de una forma diferente, aunque todos ellos tienen el mismo nombre. Por ejemplo, otro constructor de la clase Date seria el que se muestra a continuación:

fechaNacimiento - new Date(1969, 6, 13);

Este constructor inicializa el objeto fechaNacimiento a unos valores determinados.

Nota

Es muy habitual que la declaración del objeto y su creación se produzcan dentro de la misma sentencia:

ClaseDelObjeto nombreObjeto = *new* GlaseDelObjeto;

NOTAS:

Acceso a las variables y métodos de los objetos

El acceso alas variables y métodos de un objeto se realiza a través del operador punto. La forma general es:

nombreDelObjeto. variableAAcceder

Si, per ejemplo, queremos acceder a la variable height del objeto miRectangulo que pertenece a la clase Rectangle deberemos programar:

miRectangulo. height

El acceso a los métodos del objeto se hace igual. Si pretendemos acceder al método move() de miRectangulo, haremos lo siguiente:

miRectangulo. move (12,12)

Nota

La clase Rectangle es una clase que pertenece al paquete java.awt y sirve para definir rectángulos. La variable height indica la altura en pixeles del rectángulo, mientras que el método move() desplaza la esquina superior izquierda del rectángulo a las coordenadas especificadas como argumento.

NOTAS:

Creación de clases en Java

La definición de una clase en Java consta de varias partes, tal y como vemos a continuación:

- La declaración de la clase y de la clase padre correspondiente.
- La definición de las variables de la clase.
- La definición de los métodos relativos a la clase.

Para ilustrar el proceso de creación de clases, vamos a implementar una clase llamada VectorEspacial, que nos sirve para identificar cualquier vector de tres dimensiones. Como todos los vectores, pueden sumarse, restarse y calcular el producto escalar. De momento, excluiremos otras operaciones entre vectores mas complejas, como el producto vectorial.

Declaración de la clase y de la clase padre

La declaración de una clase en Java es simplemente la descripción del nombre de la clase, de que otra clase es hereditaria y si es una clase pública, final o abstracta.

En general, una clase en Java se declara:

[tipoClase] class NombreDeLaClase [extends ClasePadre]

Una declaración de clase en Java al menos debe contener la palabra reservada class y a continuación el nombre de la clase. En lo referente al nombre que se tiene para la clase, es valido lo que ya se ha comentado anteriormente para nombres de variables. Hacer notar, eso si, que la convención indica que el nombre de la clase debe comenzar con una letra mayúscula.

NOTAS:

En Java, todas las clases deben tener una clase padre, también llamada normalmente superclase. Si nosotros no le especificamos de qué superclase provienen, Java asume por defecto que proviene de la clase Object. La clase Object es la clase raíz, padre de todas las clases en Java y se verá sobre ella mas adelante.

Para especificar la superclase, Java dispone de la palabra reservada extends, tras la cual se debe indicar el nombre de la superclase.

Asi, la declaración:

```
class MesaDeOficina extends Mesa
{

    .....

}
```

nos indica que la clase MesaDeOficina es hija de la clase Mesa. Como ya es sabido, esto quiere decir que la clase MesaDeOficina hereda todas las variables y métodos de la clase Mesa.

Como ya se ha comentado antes, Java no admite herencia múltiple, por lo que una clase sólo tiene una superclase.

Las clases abstractas se declaran en Java anteponiendo la palabra abstract a la palabra reservada class de la manera que se indica a continuación:

```
abstract class Insecto {

    //variables de clase

int numeroDePatas; float velocidadDeVuelo;

    //métodos de la clase

public void vuela();
```

NOTAS:

Esta seria la implementación en Java de las clases Insecto, Mosca y Abeja propuestas en un apartado anterior.

Al ser la clase Insecto una clase abstracta, no podemos crear instancias de ella, por lo que la línea:

```
Insecto mosquito = new Insecto();
```

provocaría un error de compilación, cosa que no sucedería con la línea siguiente:

```
Abeja abejaReina = new Abeja();
```

A veces, por motivos de diseño, puede resultar conveniente que una clase no pueda ser superclase de otras. Esta clase es, por así decirlo, la última en el árbol de clases: no puede tener descendientes. A este tipo de clases se le denomina *clases finales*

La forma de implementar clases finales en Java es simplemente anteponiendo final a la palabra class, tal como se puede observar en el siguiente ejemplo:

```
final class Abeja extends Insecto {
```

Al haber sido declarada como final, la siguiente declaración de clase originaria un error de compilación.

```
class AbejaReina extends Abeja {
```

En lo que respecta a la implementación de nuestra clase Vector Espacial, de momento tendríamos:

```
class VectorEspacial {

    // variables asociadas

    . . . . .

    //métodos asociados
}
```

NOTAS:

Definición de las variables de una clase

Las variables de una clase se declaran, de forma general, de la siguiente manera:

[Tipo de Acceso] [static] [final] [transient] [volatile] *tipodeVariable nombredeVariable;*

E1 tipoDeAcceso designa que otras clases pueden tener acceso a la variante. Puedes tomar alguno de los va]ores que se relacionan a continuación:

Tabla 3.8. Tipos de acceso en Java

Tipo de acceso	Descripción
private	Solo es accesible para los miembros de la clase.
package	Accesible para los miembros de la clase y las clases del mismo paquete.
Protected	Es accesible para los miembros de la clase, las subclases y todas las clases del mismo paquete.
public	Accesible para todas las clases, sean o no del mismo paquete.

El concepto que nosotros podríamos llegar a tener de paquete en Java es totalmente equivalente al que se tiene de librería en C++: se trata de un conjunto de clases que están agrupadas de una determinada manera.

NOTAS:

Nota

Tal y como han sido definidos, los tipos de acceso también tienen su importancia a la hora de implementar la herencia. Una subclase hereda únicamente las variables y los métodos accesibles para ella. Con esto queremos decir que no hereda las variables declaradas como private. Tampoco heredará aquellas variables declaradas como package si la subclase no se encuentra en el mismo paquete que la clase padre.

Static es una palabra clave que indica que la variable es una variable de la clase en lugar de una variable del objeto. Para ilustrar una vez más la diferencia entre variables de clase y variables de objeto, pensemos en las clases Math y Rectangle. La clase Math define una serie de elementos para operaciones matemáticas. En cuanto a la clase Rectangle, ya sabemos que sirve para definir rectángulos. Una de las variables de los objetos de tal clase Rectangle es height, que es la variable donde se almacena la altura del rectángulo. Esta variable tendrá un valor diferente para cada objeto de la clase Rectangle concreto. Se trata de una variable del objeto. En la clase Math, sin embargo, se define la variable pi, que representa al famoso número 3.141592 .... Esta variable debe valer lo mismo para todos los objetos de la clase. Se trata de una variable de la clase, y como tales declarada como static.

La palabra final indica que la variable se trata de una constante, es decir, que no se puede cambiar su valor a lo largo de la ejecución del programa. Volviendo a la variable pi de la clase Math esta declarada de la siguiente forma:

```
class Math {

public static final pi = 3. 1415.. .;
```

Transient indica que la variable no pertenece al estado del objeto. Sin embargo, Volatile nos esta indicando que la variable es modificada de forma asíncrona. Esto reviste importancia única y exclusivamente a la hora de ponernos a programar las hebras, y no haremos mas distinciones sobre este tema.

El tipo de la variable puede ser uno de los tipos de referencia básicos (int, float, short ...) o bien ser una clase.

NOTAS:

En nuestro caso, las variables a definir de la clase VectorEspacial serian, por ejemplo, las siguientes:

```
class VectorEspacial {

// variables de la clase

public float x;              //coordenada del eje x

public float y;              // coordenada del eje y

public float z;              //coordenada del ele z

//m&todos de la clase

}
```

Las variables x, y, z al haber sido declaradas como public, pueden accederse desde cualquier otra clase. Así, para modificar el valor de las coordenadas de un objeto vectorA de la clase VectorEspacial simplemente sería necesario hacer·

```
vectorA.x = 1;

vectorA .y = 1;

vectorA. z = 1;
```

lo que no podría hacerse si las tres variables hubiesen sido declaradas como private·

NOTAS:

Definición de los métodos de una clase

Los modos de la clase tienen las partes que podemos ver a continuación:

- declaración del método.
- Cuerpo del método.

La declaración del método debe contener, al menos, las siguientes cláusulas:

TipoDeRetorno nombreMétodo (Tipo Arg1, Arg2, *Arg3 .....TipoArgN* argN)

{

//cuerpo del método

}

o

TipoDeRetono especifica el tipo de la variable que devuelven. Si el método no devuelve ningún valor, esto se especifica con la palabra reservada void.

NombreMetodo es el nombre con el que el método se va a referenciar. Respecto al nombre del método, son validas todas las limitaciones y recomendaciones aplicables a los nombres de variables.

Como ya se ha explicado, es posible darle aun método el mismo nombre de otro método de la clase padre. A esta operación se la llama *sobrescrita de métodos,* y sirve para que distintos objetos se comporten de forma diferente ante la misma acción.

NOTAS:

Para ilustrar esta técnica que tiene mucha importancia en Java, supongamos que hemos definido una clase Rectangle. Esta clase tiene un método denominado dibuja, al que se le pasa como argumento los dos lados desiguales del rectángulo. Supongamos que queremos crear la clase Cuadrado como subclase de la clase Rectangle. El cuadrado en realidad no es mas que un tipo particular de rectángulo en el cual ambos lados son iguales, por lo que nosotros podríamos implementar un método dibuja () diferente al de la clase padre, en el que ahora únicamente se le pasa como argumento el lado del cuadrado. Hemos realizado una sobrescritura del método dibuja () de la clase padre. Si programamos la sentencia:

```
figuraGeometrica.dibuja(12, 12);
```

y el objeto figuraGeometrica pertenece a la clase Cuadrado, originaremos un error de compilación. Si, por el contrario, figuraGeometrica pertenece a la clase Rectangulo, dibujara un cuadrado de lado 12.

Volviendo al ejemplo de la clase VectorEspacial para ilustrar la escritura de métodos, se podría declarar un constructor de la clase de la siguiente manera:

```
void VectorEspacial ( float x, float y, float z)
{
this.x = x;
this.y = y;
this.z = z;
}
```

Este constructor crea un objeto de la clase VectorEspacial y le asigna valores a los tres componentes del vector. Es interesante notar cómo se realiza el mecanismo de asignación de los vectores. La palabra reservada this sirve para diferenciar el argumento del método de nombre x de la variable de la clase de nombre x. Mas específicamente, si los argumentos del método los hubiésemos denominado, por ejemplo, coordenadaX, coordenadaY y coordenada Z, el método se rescribiría como:

NOTAS:

```
void VectorEspacial (  float coordenadaX, float coordenadaY, float coordenadaZ)
    {

//carga los valores en las variables de la clase x = coordenadaX; y =

    coordenadaY; z = coordenadaZ;
}
```

Este constructor no devuelve ningún valor, por lo que el tipo de retorno del método es void.

A continuación vamos a implementar un método sumaVector() algo mas complicado que implemente la suma de vectores espaciales. Una manera de construir este método sería la siguiente:

```
VectorEspacial sumaVector(VectorEspacial vectorASumar){

    VectorEspacial vectorResultado = new

    VectorEspacial(x    + vectorASumar.x , y + vectorASumar.y ,z

    +vectorASumar.z) ;

    return vectorResultado;

}
```

Este método tiene como argumento el vector espacial a sumar. Implementa un objeto de la clase VectorEspacial en el que va a guardar el resultado de la suma, y precisamente lo inicializa a la suma de los dos vectores. La forma de devolver como resultado ese vector suma es utilizando la sentencia return vectorResultado. La forma general de devolución de valores de los métodos es la siguiente:

```
return variableADevolver;
```

Obsérvese que como en este método no se produce ninguna indefinición de variables, no es necesario el uso de la palabra reservada this.

Además de las cláusulas obligatorias del método, que ya hemos visto anteriormente, la declaración de un método puede incluir de manera opcional otras. La forma general de declaración de un método es la siguiente:

NOTAS:

[TipoDeAcceso] ..: [static] [final] [abstract] [native]

[synchronized] T*ipoDeRetorno nombreMetodo(tipoArgl* argl,

TipoArg2 arg2, ...) {

//cuerpo del método

}

E1 tipoDeAcceso designa que otras clases pueden tener acceso al método. Los valores que puede tomar son los mismos que se aplican al tipo de acceso de las variables de la clase, descritos anteriormente.

Static es una palabra clave que indica que el método es un método de la clase en lugar de un método del objeto.

Final indica que el método no puede ser sobrescrito por las posibles subclases de la clase.

Abstract indica que el método es un método abstracto. Al igual que existen clases abstractas, Java permite la declaración de métodos abstractos, es decir, que no tienen implementación. Las razones para ello son, nuevamente, razones de diseño.

Nota

Sobre el modificador synchronized, únicamente decir que están relacionados con una parte de la programación multitarea que no trataremos en este curso. Y el modificador native se encuentra relacionado con la importación de código nativo en C.

NOTAS:

Estructuras de control en Java

Las estructuras de control son líneas del programa que especifican la forma o el orden en la que otras líneas del programa deben ser ejecutadas. Las estructuras de control de Java se aplican a los apartados que puedes ver a continuación:

El bucle *while()*

La sintaxis general del bucle while podría ser parecida a la que mostramos a continuación:

```
while (expresion) {

lineaDeCodigo1;

lineaDeCodigo2;
lineaDeCodigoN;

}
```

Esta estructura de control nos esta indicando que mientras la expresión contenida en expresión sea cierta se van a ejecutar todas las líneas de código dentro de las llaves, Cuando solamente sea necesario ejecutar una línea de código, la expresión anterior puede quedar simplificada y reducida a:

```
while (expresion) lineaDeCodigo;
```

El bucle *do-while()*

Esta estructura de control es bastante parecida a la anterior. Su forma es:

```
do {

    lineaDeCodigo1;

    lineaDeCodigo2;

} while(expresion)
```

NOTAS:

El bucle *for()*

La forma general de este bucle será similar a la que representamos a continuación:

```
For (inicialización; terminación; incremento) {

lineaDeCodigol;

lineaDeCodigo2;

}
```

Inicialización contiene lo que queremos que haga el bucle la primera vez que entremos en él. Terminación contiene una expresión que es evaluada en cada iteración: mientras ésta sea cierta, se ejecutara el bucle. Incremento es una sentencia que se ejecuta al principio de cada iteración en el bucle.

Un ejemplo de este bucle puede ser el programa que listamos a continuación, que va a representar en la pantalla los números del 1 al 10:

```
int contador;

for (contador = 1; contador <= 10; contador++ ) {

System.out.println ( << Vuelta número << + contador);

}
```

Existe la posibilidad de que nosotros declaremos la propia variable contador dentro de for. Con esta declaración de variables, el código visto anteriormente vendría a ser el equivalente al que se muestra a continuación:

```
for int contador = 1 ; contador < 10 ; contador++ ) {
    System.out.println(<<Vuelta número << + contador);

}
```

NOTAS:

La sentencia if-else

Con esta estructura de control podemos ejecutar de forma selectiva partes de código de la aplicación. Su forma mas generalizada es la siguiente:

```
if( expresión ) {

lineaDeCodigo1;

 lineaDeCodigo2;
}

else {

      lineaDeCodigoA;

      lineaDeCodigoB;


   }
```

Cuando la aplicación llega a if, evalúa el contenido de Expresión. Si éste es cierto, entonces procede a ejecutar las líneas de código 1, 2 y sucesivas. Por el contrario, si el contenido de Expresión es falso, ejecuta las líneas A, B y siguientes.

Esta sentencia admire diversas variantes. Por ejemplo, si no se desea ejecutar ninguna acción en el caso de que expresión sea falsa, se puede eliminar la proposición else, con lo que el bloque quedaría:

```
If ( expresion ) {

lineaDeCodigo1; lineaDeCodigo2;

}
```

NOTAS:

También, cuando la línea de código que se va a ejecutar en alguno de los casos -o en ambos- es sólo una, es posible eliminar las llaves, con lo que el bloque de código quedaría como sigue:

```
if ( expresion ) lineaDeCodigol;

else lineaDeCodigoA;
```

Una tercera posibilidad es la de anidar bloques if-else de forma consecutiva. Supongamos que deseamos realizar acciones distintas en función de que la variable tipoDeAccion valga 1, 2, 3 ó 4. Esto se puede hacer de dos maneras. Una de ellas consiste en mantener el formato original:

```
if( tipoDeAccion == 1 ) {
    . . .
}

else {
    if{ tipoDeAccion == 2 ) {
        . . .
    }
    else {
        if( tipoSeAccion == 3 ) {
            . . .
        }


        //esto serviría para 4
        else {
            . . .

        }

    }

    }
```

NOTAS:

La otra consiste en utilizar de forma encadenada if-else, como se muestra a continuación:

```
if( tipoDeAccion == 1 ) {

    . . .

}

else if( tipoDeAccion == 2 ) {

    . . .

}

else if{ tipoDeAccion -- 3 ) {

    . . .

}

else {

    .... .

}
```

NOTAS:

La sentencia case-switch

La sentencia switch sirve para ejecutar determinados bloques de código en función del valor que tome una variable. La forma general es la siguiente:

```
switch (variable) {

case valor1:
        . . .

case valor2:

        . . .


case valorn:
        . . .
}
```

Este bloque de código se ejecutará de la manera siguiente: supongamos que variable toma el valor valorn. Entonces, cuando se llegue al bloque switch se ejecutarán las líneas de código correspondientes a valorn, posteriormente las correspondientes a valorn+1, y así sucesivamente. Si queremos que solamente se ejecute el bloque de código correspondiente al valor que toma variable, deberemos asegurarnos de incluir al final de cada bloque de código la expresión break, de la siguiente manera:

```
switch (variable) {

case valor1:
        . .
        break;
```

NOTAS:

```
case valor2:
        . . .
        break;

case valorn:
        . . .
        break;
}
```

Cuando se llega a la línea que contiene la palabra break se interrumpe la continuidad en la ejecución y se pasa a ejecutar la línea de código posterior a la llave cerrada de switch. Esto mismo sucederá si variable no toma ninguno de los valores expresados en alguno de los case. Si queremos ejecutar alguna acción en el case de que variable no se ajuste a ninguna de las opciones, deberemos incluir la opción default· Volviendo al ejemplo anterior, en el que queríamos realizar una acción distinta en función de que la variable tipoDoAccion tomase los valores 1, 2, 3 ó 4, ahora podríamos implementarlo con switch-case:

```
switch (variable) {
case valor1:
        . . .
case valor2:
        . . .
case valorn:
        . . .
}
```

NOTAS:

Aunque son muy parecidos, los bloques switch-case y if-else no son completamente equivalentes:

En switch-case la decisión se realiza sobre una variable, mientras que per el contrario, en if-else es una expresión. En el sentido anteriormente expuesto, if-else resulta mucho mas general que switch-case·

If-else siempre toma una acción per defecto. Esto no tiene per qué suceder con switch-case·

Etiquetas break-continue

En la estructura switch-case habíamos visto el uso de break para interrumpir la ejecución de case y salir del bloque switch. Un segundo uso de break es la de llevar la ejecución del programa a una etiqueta. Las etiquetas en Java se ponen con un identificador o nombre de la etiqueta seguido de dos puntos. Una etiqueta sería:

```
etiqueta:    LineaDeCodigol;
             LineaDeCodigo2;
             LineaDeCodigoN;
```

Pues bien, si en algún punto de la ejecución del programa incluimos la siguiente línea:

```
break etiquta;
```

cuando el programa llegue a esta línea saltará a la zona etiquetada y ejecutará las LifneDeCódigo1, LíneaDeCodigo2, etc.

Nota

La inclusión de etiquetas en una aplicación disminuye sustancialmente su legibilidad, al mismo tiempo que introduce elementos no estructurados en la programación. Por todo ello, siempre deben considerarse todas las alternativas posibles antes de incluir etiquetas en una aplicación.

NOTAS:

Las etiquetas en Java sustituyen al goto de C/C++. Aunque goto es una palabra reservada en Java, carece de un significado concreto.

Nota

La palabra reservada continue cumple la misma misión que break. La diferencia es que únicamente puede ser llamada desde dentro de un bucle.

Otras estructuras de control

Además de las ya vistas en los apartados anteriores, Java dispone de otras estructuras de control. Quizás la más importante de ellas sea el bloque try-catch. Este bloque resulta básico para el manejo de errores mediante excepciones y se verá con mas profundidad en los siguientes apartados.

NOTAS:

Los paquetes de Java

Es una práctica habitual en la programación agrupar objetos en librerías de clase. En Java, esas librerías de clase se denominan paquetes.

Java nos permite la incorporación de las clases que nosotros escribamos en paquetes a los que siempre se puede recurrir para no volver a escribir las mismas clases. El desarrollo de paquetes simplifica la tarea de programar.

Además de poder crear nuestros propios paquetes de clases, Java incorpora en su entorno de desarrollo toda una serie de paquetes estándar. Estos paquetes cubren una inmensa cantidad de aspectos que pueden variar desde clases para facilitar el dibujo en pantalla hasta comunicaciones o el manejo de cadenas de caracteres, etc. Antes de escribir clases nuevas, es conveniente echar un vistazo alas clases predefinidas de Java.

Nota

Un conocimiento profundo de las clases estándar provistas por el lenguaje es fundamental en la programación orientada a objetos. El dominio de las clases estándar ahorra horas de programación y estandariza los procesos, robusteciendo las aplicaciones. Sólo cuando se está completamente seguro de que las clases existentes no cubren las necesidades concretas de programación se debe recurrir al desarrollo de nuevas clases.

Importación de paquetes

Para que nuestra aplicación o applet entienda las clases de un paquete determinado, es necesario importarlas. La importación de clases en Java se lleva a cabo mediante la sentencia import. La forma general de la sentencia es la siguiente:

import paquete. ClaseAimportar ;

Esta sentencia se debe incluir al principio del archivo .java. Así, para poder utilizar la clase Rectangle, que pertenece al paquete java.awt, será necesario incluir lo siguiente al principio del archivo:

import java.awt.Rectangle;

NOTAS:

Si se utiliza una clase sin importarla, se origina un error de compilación. Si en nuestra aplicación o applet hacemos un uso continuado de muchas clases de un mismo paquete, este mecanismo de importación de todas las clases se haría muy tedioso. Para incluir todas las clases de un paquete se puede utilizar el símbolo ' * '. Así, la siguiente línea:

```
import java.awt.*;
```

importaría todas las clases del paquete java.awt

NOTAS:

Paquetes predefinidos en JDK

A continuación se van a introducir los paquetes incluidos por Sun en el entorno de desarrollo de Java. Algunas de estas clases se usaran constantemente en los siguientes apartados.

Son ocho paquetes los que Java incluye en JDK 1.0.2:

- Java.lang, que es el paquete principal.

- Java.io, para controlar las operaciones de entrada/salida.
- Java.util, un paquete con clases y métodos de propósitos variados.
- Java.net, para dar soporte a las comunicaciones TCP/IP.
- Java.applet, centrado en Applet, fundamental en Internet.
- Java.awt, que contiene las clases y métodos de Abstract Windows Toolkit para el manejo de la interfaz de usuario.
- Java.awt.image, con clases y métodos independientes del sistema.
- Java.awt.peer, con clases y métodos específicos de cada plataforma.

En los próximos apartados veremos estos paquetes con algo mas de profundidad.

java.lang

Este paquete contiene las clases que constituyen el núcleo de la programación en Java. Este paquete es incorporado por javac de forma automática, por lo que no hace falta especificarlo mediante la sentencia import. Este es el único paquete de Java que se importa de manera automática. Una clasificación de las clases de este paquete podía ser:

- La clase Object. Esta clase es la raíz de todas las clases en Java. Cualquier elemento de Java es descendiente, directo o indirecto, de la clase Object.

- Clases para el tratamiento de cadenas de caracteres. En particular, las clases String y StringBuffer.

- Clases para el manejo de números y de funciones matemáticas. Entre éstas destaca la clase Math, que va a contener todos los métodos que nos serán necesarios para llevar a cabo todo tipo de funciones.

NOTAS:

- Las clases System y Runtime. Mediante el uso de estas paginas se puede acceder a los recursos del sistema. E1 uso de estas clases presupone unos conocimientos avanzados de Java y de los sistemas hardware, por lo que en este caso sólo las vamos a tratar de forma básica.

- Clases para programación multitarea. Se trata de las clases Thread, ThreadDeath y ThreadGroup. La primera de ellas se verá con más profundidad.

- Clases para el manejo de errores y de excepciones. Estas clases le dan robustez a la programación en Java, recogiendo las posibles salidas a los errores y excepciones que puedan ocurrir.

ja.io

En este paquete se recogen las clases que hacen referencia a la salida y entrada de datos. Este paquete proporciona clases para el manejo de canales de entrada/salida a bajo nivel y tratamiento de archivos.

java.util

Este paquete contiene un conjunto de clases para facilitar el manejo de ciertas estructuras de datos. Incluye la definición de clases como las clases Date y Time, para poder acceder a la fecha y hora del sistema, clases para la manipulación de cadenas de caracteres, o métodos para la generación de números pseudoaleatorios.

Se trata de una especie de cajón de sastre donde se recogen una variedad de clases para facilitar propósitos muy distintos.

java.net

En este paquete se agrupan todas aquellas clases que hacen referencia a las comunicaciones en red. Estas clases se utilizan para el desarrollo de aplicaciones cliente-servidor y aplicaciones de comunicación entre programas.

java.applet

Este paquete contiene únicamente a la clase Applet. La clase applet es sumamente importante en Java, puesto que es la que permite el desarrollo de aplicaciones incrustadas en paginas de HTML y la que confiere a Java todas las capacidades en Internet.

NOTAS:

java.awt

Este paquete define las clases necesarias para desarrollar una interfaz gráfica de usuario, de manera que el usuario de la aplicación pueda interactuar con ella. Incluye clases para el desarrollo de botones, etiquetas de texto, etc. Estas clases son independientes de la plataforma en la que se ejecute la aplicación.

java.awt.image

En este paquete se hallan las clases necesarias para el manejo de grafitos. También son clases independientes del sistema operativo sobre el que se ejecuten.

java.awt.peer

Este paquete contiene las clases para conectar todos los componentes definidos en la clase awt con los componentes específicos de cada sistema operativo

Otros tipos de datos en Java

Además de los tipos de datos básicos que hemos tenido la ocasión de ver hasta ahora, en Java aparecen también otros tipos que son derivados de los anteriores. Entre estos tipos, dos de los que utilizaremos con una mayor frecuencia son las cadenas de caracteres y los arrays.

Ambos tipos de datos son modelados en Java como objetos, lo que nos dará la oportunidad de utilizar algunas variables y métodos útiles.

Cadenas de caracteres

Java tiene a su disposición una clase para poder implementar las diversas cadenas de caracteres, la clase String, perteneciente al paquete java.lang.

Nota

Una cadena de caracteres no es masque un conjunto de datos de tipo carácter que ocupa posiciones de memoria contiguas.

NOTAS:

La declaración de una variable de tipo String denominada frase se llevará a cabo con la sentencia:

String frase;

Como ya se había comentado en el capitulo anterior, el operador + nos va a servir para poder concatenar diversas cadenas de caracteres. La sentencia:

frase = "En un lugar " + " de la Mancha..";

almacena en la variable frase la cadena de caracteres "En un lugar de la Mancha...".

En realidad, el operador + no solamente concatena las diversas cadenas de caracteres, sino que también lleva a cabo una conversión de tipos. Si nosotros tenemos la variable entera diaDelMes, de valor 21, la sentencia:

frase = "México a " + diaDelMes + " de Julio del 2001 ";

almacena en la variable frase la cadena de caracteres <<México a, 21 de Julio del 2001>>, habiendo llevado a cabo una transformación del tipo entero diaDelMes en una cadena de caracteres.

Aviso

Es importante tener en cuenta la diferencia entre cadena de caracteres y carácter:

En Java 'a' es el carácter a y será tratado por Java como una variable de tipo char. Sin embargo, "a" es la cadena de caracteres a y será tratada por Java como una variable del tipo String.

Un método de la clase String que nos puede ser de alguna utilidad es el método lenght(), que tiene la siguiente forma:

public int lenght ();

y que devuelve la longitud del objeto de la clase String.

NOTAS:

Arrays de datos

Como es sabido, los arrays son vectores de datos de cualquier tipo recogidos en una variable determinada.

Nota

Java implementa el uso de arrays de una manera simple y segura, evitando la referencia a punteros existente en C/C++.

Como si se tratase de una variable más, lo primero que hay que hacer para utilizar un array es declararlo. Vimos en capítulos anteriores que al declarar una variable, debíamos especificar el nombre, el tipo y el alcance. Si, por ejemplo, queremos crear un array de enteros de alcance público llamado vectorDePrueba, la declaración sería:

```
public int [] vectorDePrueba;
```

Esta sentencia simplemente declara un objeto de la clase array de enteros llamado vectorDePrueba. Puesto que los arrays son objetos en Java, es necesario distinguir la declaración del objeto de la creación de una instancia de ese objeto. Más concretamente, la sentencia anterior no ha reservado ninguna zona de memoria en la que alojar vectorDePrueba, entre otras cosas porque el tamaño de vectorDePrueba es desconocido en este punto y no se sabe cuanta memoria reservar.

Entonces, para crear el objeto vectorDePrueba es necesario añadir la siguiente línea:

```
vectorDePrueba    = new int  [10];
```

Las dos sentencias anteriores se pueden resumir en:

```
public int [ ] vectorDePrueba = new int[10];
```

que declara y crea el objeto simultáneamente. La forma general de creación de arrays se puede expresar como:

```
tipoElemento [ ] nombreArray= new tipoElemento [longitud];
```

NOTAS:

Aviso

En Java el primer elemento del array es el que ocupa la posición cero. Así, la sentencia vector[1] no hace referencia al primer elemento del array vector, sino al segundo. El primer elemento se reverenciaría con la sentencia vector[0].

Los arrays pueden contener cualquier elemento. La forrna más general posible de array podría venir dada por la sentencia que mostramos a continuación, que declara un vector de cinco elementos cualesquiera en Java:

```
Object[] vectorGenerico = new Object[5];
```

Para obtener el tamaño de un array se utiliza la variable lenght, característica de todos los arrays. Para ilustrarla con un ejemplo, escribimos la parte de código correspondiente al calculo del elemento mayor de un vector de enteros vector:

```
..... +

int i;

int mayor = 0;

for (i=0 ; i <= vectyor.lenght; i ++) {
        if (mayor < vector [ i ] ) mayor = vector [ i ];
}
. . . .
```

NOTAS:

Construyendo una aplicación en Java

Ya le hemos echado un vistazo a las piezas fundamentales que constituyen una aplicación en Java. Ahora solamente nos falta ponerlas juntas.

El método main()

Una vez que hemos conseguido compilar con javac un archivo .java, si ejecutamos el comando java nombre del archivo, éste busca automáticamente un método denominado main() y que tiene la sintaxis mostrada a continuación:

```
public static void main( String [ ] argumentos ) {

. . .
. . .

}
```

Dentro de este método tendremos que incluir todo lo que queramos que realice nuestra aplicación. Este método conlleva como argumento el array de cadenas de caracteres que denominamos argumentos. Como norma general, en la mayoría de las ocasiones no necesitaremos esos argumentos.

En otras ocasiones, el paso de argumentos a través de las líneas de comandos puede sernos de una gran utilidad. Un ejemplo muy sencillo del paso de argumentos viene dado por esta pequeña aplicación en Java que responden todo aquello que el usuario le introduce a través del teclado, y que hemos llamaremos : eco. El código es el mostrado a continuación:

```
Public class Eco {

  public static void main( String[ ] argumentos ) {

     if ( argumentos[0] != null ) {
          System.out.println( argumentos[0] );

     }

  }
```

NOTAS:

}

Consejo

Si queremos asegurarnos de mantener la máxima portabilidad de nuestras aplicaciones en Java, debemos evitar el paso de argumentos a través de la Línea de comandos. La razón para todo esto es que existen sistemas operativos, como por ejemplo Macintosh, en los que la utilización de argumentos en la línea de comandos no tiene prácticamente ningún sentido y no funcionaran correctamente. Además, siempre existen mecanismos para poder sustituir esta operación.

Nociones básicas de entrada y salida

Sea cual sea el objeto de nuestra aplicación o applet, de lo que estamos seguros es de que necesitará un mecanismo de introducción de información, y otro de presentación de esa información ya elaborada. En este apartado nos iniciaremos en los métodos básicos para que nuestras aplicaciones sean capaces de recibir y mostrar información.

Nota

Las aplicaciones más avanzadas suelen utilizar la interfaz de usuario de Java como mecanismo de paso de información

NOTAS:

Los canales estándar de entrada y salida

El ordenador, cuando se comunica, dispone de unos mecanismos determinados de comunicación en función precisamente de ese tipo de información. Así, si la comunicación consiste en la entrada de datos, ésta puede hacerse, por ejemplo, a través de teclado. El usuario tiene que introducir un número. Entonces teclea ese número y éste pasa a la aplicación a través de lo que se denomina un *canal de entrada* (en ingles, *input stream)* de datos. El canal de entrada de información en nuestra PC suele ser el teclado o un archivo.

Análogamente, si lo que se pretende es que se muestren unos datos elaborados, el medio a través del cual la aplicación envía esos datos se denomina *canal de salida (output stream)* de datos. El canal de salida puede ser la pantalla, la impresora, un archivo o cualquier otro dispositivo físico de salida de datos.

Un tipo particular de datos que puede mostrar la aplicación son los mensajes de error, que suelen indicarnos que algo ha fallado en nuestra aplicación. Este tipo de datos dispone en Java - además de en otros muchos lenguajes- de un canal propio que se denomina *canal de error (error stream).*

La idea es separar en canales las vías por las que se transmite la salida de información del ordenador, la entrada o los mensajes de error.

Con el concepto de canales de entrada y salida simplificamos el enlace de nuestra aplicación con el medio físico de nuestro ordenador. Supongamos que queremos que el ordenador muestre por pantalla unos determinados datos. Tendremos que escribir el código que elabore esos datos y luego lanzar esos datos por el canal de salida, que habitualmente es la pantalla. Supongamos ahora que deseamos que esos datos se muestren por impresora. No necesitamos modificar el código: basta asignar como canal de salida nuestro propio dispositivo de impresión. Lo mismo sucedería si ahora queremos que el informe sea almacenado dentro de un archivo del disco duro.

> Si no existiesen esos canales, tendríamos que escribir un código especifico para que la información apareciese en pantalla y otro totalmente distinto para que el informe saliese impreso.

NOTAS:

Aviso

El canal de error no tiene por qué coincidir con el canal de salida. Resultaría conveniente recordar que nuestro canal de salida en un momento determinado podría ser un archivo o la impresora, mientras que, casi con toda seguridad, desearemos que el canal de error siga siendo nuestra pantalla.

La clase System

La clase System es una clase que controla el funcionamiento de la Maquina Virtual Java. Entre otras cosas, sirve para controlar los canales de entrada, salida y error.

Aviso

La clase System es una clase de la que no pueden crearse instancias. Es creada por Java cuando se ejecuta un applet o aplicación, y por lo tanto, se la puede invocar directamente.

La corriente de entrada se controla a través de System.in. La de salida a través de System.out, mientras que la de errores se controla mediante System.err.

Nota

Un método útil de la clase System es el método exit (), que provoca la finalización de una aplicación en Java cuando éste es invocado. Como argumento tiene el estatus de salida. Nosotros le vamos a dar siempre el valor 1.

NOTAS:

Acceso a los canales estándar de salida y de error

La manera más sencilla para escribir dates en el canal de salida es utilizar alguna de las sentencias siguientes:

System.out.print(cadena);

System.out.println(cadena);

Ambos métodos van a provocar la escritura en el canal de salida estándar del dato cadena.

La diferencia entre ambos métodos radica en el echo de que el segundo de los método introduce un salto de línea al final de la cadena, con lo que la siguiente cadena continuaria en la línea posterior.

Nota

Para que nosotros podamos introducir un salto de línea utilizando el método print (), basta con poner al final del date el carácter de control '\n'. Este carácter es un carácter estándar que cuando es lanzado a través del canal de salida, es convenientemente interpretado come un salto de línea.

También podríamos escribir en el canal de errores con los métodos siguientes:

System.err .print (cadena);

System.err.println(cadena);

que tienen la misma funcionalidad que los anteriores.

En cuanto al tipo de datos de la variable cadena hay que decir que estos métodos admiten varios tipos de datos. Los tipos de datos admitidos son:

NOTAS:

Tipo de dato	*Devuelve*
Object	El nombre de la clase
String	El contenido de la cadena de caracteres
char[]	El conjunto de caracteres uno a continuación del otro
int	El contenido de la variable
long	El contenido de la variable
float	El contenido de la variable
double	El contenido de la variable
boolean	true o false

Nota

Existe un tercer método write() que se va a utilizar para escribir datos binarios (información que no es de tipo carácter). Se utiliza menos frecuentemente que los anteriores y nosotros no haremos uso de él.

Acceso al canal estándar de entrada

Para leer datos del canal estándar de entrada tenemos el método siguiente:

System. in. read ( )

Este método devuelve el carácter que se ha introducido mediante el teclado, y el valor -1 si no se ha introducido ningún carácter. Cada vez que este método lee un carácter del teclado, lo borra de la memoria intermedia, de manera que la siguiente vez que se le llame leerá automáticamente el siguiente carácter.

NOTAS:

Uso de canales de entrada con DataInputStream

Java tiene a disposición una clase, la DataInputStream, del paquete java.io para la utilización de un canal de entrada. Con DataInputStream podemos asignar cualquiera de los canales de entrada de datos aun objeto de esta clase. Las vías a través de las que una aplicación suele recibir datos normalmente son tres:

- A través de teclado.
- A través de un archivo.
- A través de la red a la que esta conectado.

Para realizar la declaración de la clase DataInputStream podemos hacerlo si invocamos a su constructor de la forma siguiente:

```
DataInputStream canal = new DataInputStream( canalAsociado );
```

Con este método se consigue la asociación del canal canalAsociado al objeto canal.

Si el canal que nosotros estamos interesados en asociar es el teclado, en la posición de la variable canalAsociado tendremos que colocar System.in

Si se trata de un archivo denominado "archivo.txt", la declaración del objeto canal revestiría la siguiente forma:

```
DataInputStream canal = new

    DataInputStream ( new

        FileInputStream ("archivo.txt" ));
```

Nota

El proceso de asociación de un canal de entrada a los datos transmitidos a través de Internet lo veremos mas adelante..

NOTAS:

Una vez que hayamos asociado el canal de entrada al objeto de la clase DataInputStream, esta clase tiene a su disposición una serie de métodos para poder leer los diversos tipos de datos de entrada. Una lista de aquellos que son los métodos mas útiles:

Método	**Función**
readByte()	Lee un byte
readInt ()	Lee un dato de tipo entero
readChar()	Lee un dato de tipo carácter
readLine()	Lee una linea (hasta '\n')

Aviso

Java utiliza un puntero para controlar lo que lleva leído del canal de entrada. La utilización de los métodos de la tabla desplaza ese puntero. Así, si en un archivo tenemos la cadena "abc" e invocamos al método *readChar () este método devolverá* 'a'y el puntero se *moverá* apuntando a 'b'.
Si invocamos otra vez el *método readChar (), éste devolverá* 'b', *desplazando* el puntero a 'c'.

NOTAS:

A modo de ejemplo, podemos pensar en una variante de la aplicación eco que vimos atrás. En este caso, utilizaremos el canal de entrada para leer lo que el usuario introduzca por medio del teclado. El código es:

```
import java.io. * ;

public class Ejemplo {

    public static void main (String[] args)

        throws java.io.IOException {

    DataInputStream canal =  new

        DataInputStream ( System.in);
    System.out.println(canal.readLine ());

    }
}
```

Aviso

Para que funcione correctamente es necesario pulsar el retorno de carro al final de la cadena de caracteres introducida. También es fundamental la sentencia throws java.io.IOException, cuyo significado entenderemos más adelante.

Uso de canales de salida con DataOutputStream

Igual que existe la clase DataInputStream para asociar canales de entrada, Java dispone de la clase DataOuDutStream para asociar canales de salida. Desafortunadamente, el tratamiento de los canales de salida, aunque es bastante análogo a los de entrada, tiene algunas complicaciones adicionales.

NOTAS:

La clase Object

La clase Object se encuentra en la raíz de la jerarquía de todas las clases en Java. Todas las clases en Java descienden de la clase Object.

La clase Object es una clase abstracta que define las características y comportamientos comunes a cualquier objeto, como puede ser la capacidad para comparar dos objetos entre si.

La clase Object define una serie de métodos de utilidad. Algunos de ellos serán explicados a continuación.

El método equals()

Este método sirve para comprobar la igualdad entre dos objetos. La palabra igualdad sirve aquí para designar que dos objetos contienen el mismo valor, no el que ambos objetos pertenezcan a la misma clase. El método devuelve true si los objetos son iguales y false en caso contrario.

El método getClass()

Este método sirve para obtener la clase de la que procede un determinado objeto. Se trata de un método final.

El método toString()

Este método proporciona una cadena de caracteres en la que se representa el objeto en cuestión.

El método finalize ()

Con este método se fuerza a la destrucción del objeto al que se hace referencia.

Nota

Una de las ventajas que tenia Java era el garbage-collector, que iba liberando los objetos de la memoria a medida que ya no eran necesarios, con lo que un método de este tipo no es estrictamente necesario. Sin embargo, este método nos permite controlar a nuestra voluntad la destrucción del objeto.

NOTAS:

La palabra clave super

Habíamos comentado antes que una de las características más útiles de Java es la sobrescrita de métodos. Cuando queremos que un objeto se comporte de forma similar a otro, pero que difiere en unas cuantas características, lo que tenemos que hacer es sobrescribir ese método o métodos que queremos que sean distintos.

Java proporciona la palabra reservada super para hacer referencia a un método de la clase padre. El use general de super es el siguiente:

super. *metodoDeLaClasePadre (* argumentosDelHetodo .....);

Si queremos invocar al constructor de la clase padre, entonces utilizaremos únicamente la palabra super, de la siguiente forma:

super (argumentosDelConstructorDeLaClasepadre .....);

Para ver el use de la palabra super vamos a utilizar un ejemplo que nos resultara muy sencillo: supongamos la clase Rectángulo, cuyo constructor es:

```
Public  Rectangulo(int altura, int anchura);
```

El constructor tiene dos argumentos, la anchura y la altura del rectángulo en cuestión. Supongamos que queremos definir una clase Cuadrado, hija de la clase Rectángulo. Una forma de hacerlo seria declarar la clase y después sobrescribir uno a uno los métodos, empezando por sobrescribir el constructor de la clase.

Una forma sencilla y elegante de hacer todo esto es utilizar la palabra clave super, de la siguiente forma:

```
class Cuadrado extends Rectangulo {

//re-escribimos el constructor

public Cuadrado( int lado ) {
        super (lado, lado);

}
//rescribimos los otros métodos que sean necesarios
```

NOTAS:

Supongamos que queremos definir un elemento genérico Figura como cualquier elemento que tenga una cierta dimensión y se pueda representar en pantalla. Con estos condicionantes ni siquiera podemos crear una clase abstracta Figura, ya que en esa clase abstracta ya deberíamos indicar que hacen los métodos. Una posibilidad es declararla como interface:

```
interface Figura {

    //métodos de la interface

    public pinta(int x, int y);

}
```

No hay que definir cómo se lleva a cabo lo que hace el método pinta().

Cuando queremos definir una figura concreta, por ejemplo, la clase Circulo, ya conocemos que es lo que tenemos que hacer para que esa figura aparezca en pantalla, por lo que podemos especificar el contenido del método pinta(). Entonces se dice que Circulo implementa la interfaz Figura, y se haría en Java usando la palabra clave implements de la forma:

```
class Circulo implements Figura { //método (ya definido) pinta()

public pinta ( int x, int y ) {
        //definición de lo que hace el método pinta ( )

.  .  .  .  .  .

  }
}
```

NOTAS: