

# ENTORNO DE DESARROLLO INTEGRADO

## SUN FORTE FOR JAVA 3.0

Un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) permite editar, compilar, ejecutar y depurar programas de una forma cómoda y ágil. En estas notas haremos mención a algunas de las posibilidades básicas de uso del IDE Forte, de Sun Microsystems.

### 1. EDICIÓN, COMPILACIÓN Y EJECUCIÓN DE PROGRAMAS

Su uso es como el de cualquier editor de textos convencional, aunque incluye alguna herramienta de utilidad para el programador.



#### 1.1 Autoformato de código

Resulta de gran utilidad el formato automático de código: a medida que se va escribiendo el programa, el editor reconoce las diferentes estructuras, resaltando las palabras clave de Java, realizando la tabulación automáticamente o señalando los símbolos de apertura y cierre correspondientes a {, }, (, ), [, ], .... Si en un momento dado se pierde el formato del código, puede recuperarse automáticamente mediante el menú contextual,

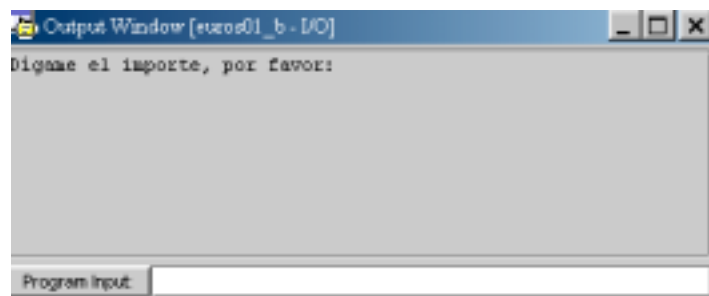


con la opción REFORMAT CODE.

## 1.2 Compilación y ejecución

Otras de las opciones de utilidad en dicho menú contextual son las de COMPILACIÓN (F9) o EJECUCIÓN (F6), a las que también se puede acceder desde la barra de herramientas a través de los símbolos correspondientes  y , respectivamente.

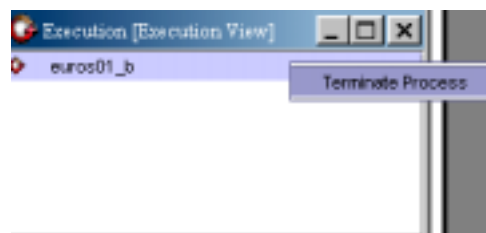
La ejecución de un programa cambia el entorno al escritorio RUNNING, en el cual están accesibles las ventanas de Entrada Salida (que actúa de terminal estándar de E/S)



y ejecución



Puede detenerse una ejecución en curso desplegando el menú contextual sobre el nombre del programa (proceso) en ejecución:

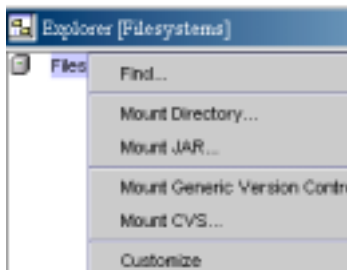


## 2. EXPLORADOR

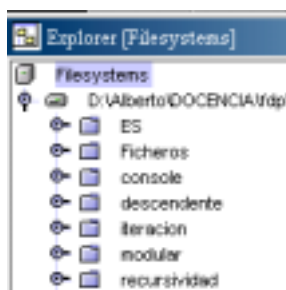
Otra herramienta de utilidad es el Explorador




que permite acceder de forma inmediata a los ficheros fuente Java. Para acceder a un programa será necesario "montarlo" en el explorador. Para ello se despliega el menú contextual

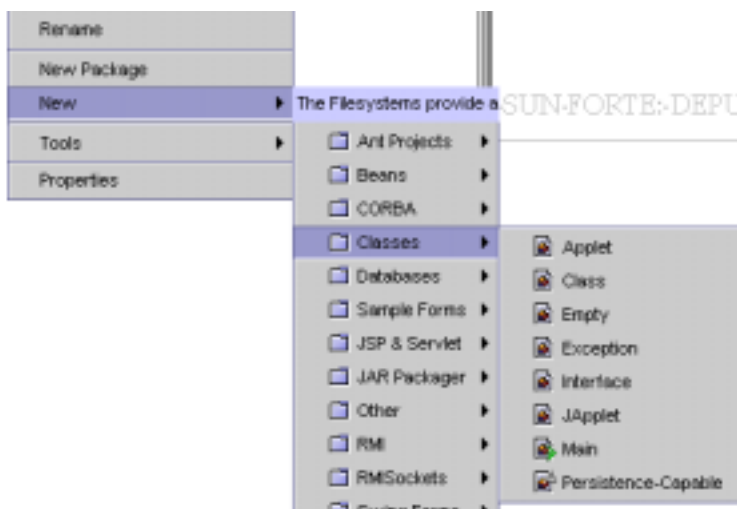


seleccionando la opción MOUNT DIRECTORY, que nos permitirá seleccionar la carpeta (paquete, en la terminología OO) que deseamos montar:

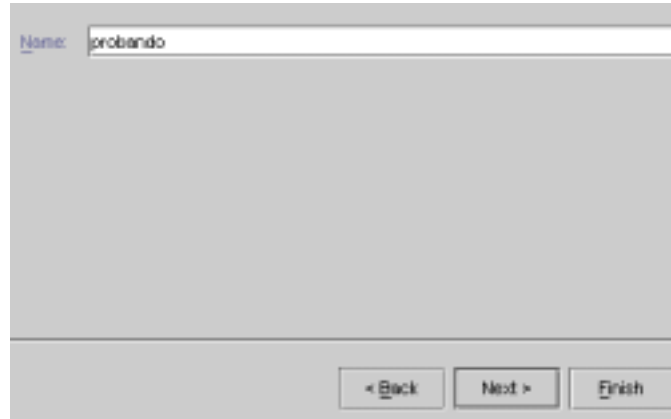


Esta operación es lo que Forte denomina "montar un FileSystem".

Pulsando en el símbolo  se pueden desplegar y replegar los árboles de carpetas, de forma similar al Explorador de Windows. Para construir un nuevo programa (una nueva clase, en terminología OO), basta con desplegar el menú contextual dentro de la carpeta correspondiente:



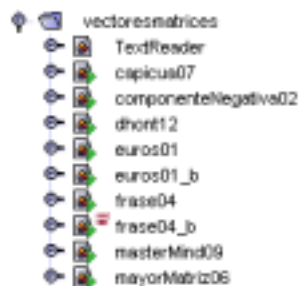
seleccionando la opción MAIN en la última de las ventanas. De este modo se abre el asistente de edición de clases donde, tras introducir el nombre de la misma (que será el nombre de nuestro programa y del fichero que lo contiene).



y pulsar el botón FINISH se abrirá el editor de programas con una clase vacía con su correspondiente método principal (función main):

```
Source Editor [probando *]
1 /*
2  * probando.java
3  *
4  * Created on 9 de enero de 2002, 16:45
5  */
6
7 package vectoresmatrices;
8
9 /**
10  *
11  * @author alberto
12  * @version
13  */
14 public class probando {
15
16     /** Creates new probando */
17     public probando() {
18     }
19
20     /**
21     * @param args the command line arguments
22     */
23     public static void main (String args[]) {
24
25         // AQUÍ ES DONDE SE VA A ESCRIBIR
26         // EL PROGRAMA PRINCIPAL
27     }
28
29 }
```

A medida que se vayan construyendo nuevos programas dentro de la carpeta el árbol en el Explorador irá adquiriendo un aspecto como el siguiente:



que nos proporciona una vía alternativa para editar los programas (Doble clic), compilarlos, ejecutarlos, borrarlos, cambiarles el nombre (menú contextual), ...

Una tarea de especial interés es la de COPIAR+PEGAR o CORTAR+PEGAR una clase/programa desde una carpeta/paquete de origen a otra carpeta/paquete de destino. Esto puede hacerse también a través del menú contextual o con los botones correspondientes



### 3. CONSIDERACIONES SOBRE LOS PROGRAMAS JAVA

Los detalles de mayor relevancia sobre la confección de programas Java tienen que ver con alguno de los siguientes aspectos:

#### 3.1 *Uso de bibliotecas de clases*

En los programas utilizaremos normalmente funciones o clases que están predefinidas en alguna de las bibliotecas que constituyen el lenguaje Java. Para poder utilizarlas es necesario importar la biblioteca correspondiente, de modo que el intérprete añada al código compilado de nuestro programa el código correspondiente a dichas funciones o clases. Esta importación se realiza con la sentencia **import**, que se utiliza ANTES de la sentencia de clase de nuestro programa (línea 14 del siguiente ejemplo)

```
14 import java.io.*;
15 public class nif11 {
16
17     /** Creates new Erase04 */
18     public nif11() {
19         }
20
21     /**
22     * @param args the command line arguments
23     */
24     public static void main (String args[]) throws IOException{
25
26         TextReader teclado = new TextReader(new InputStreamReader(System.in));
27         long dni;
28
29
30         System.out.println("Dígase su D.N.I.");
31         dni=Integer.parseInt(teclado.readString());
```

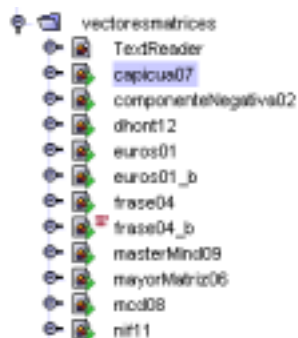
En ese caso estamos indicando que se importen todas las clases correspondientes al paquete `java.io`.

### 3.2 Excepciones

Algunas de estas funciones pueden generar excepciones, que nosotros no trataremos de forma individualizada, sino que propagaremos directamente hacia el exterior de nuestro programa. Para ello usaremos la sentencia `throws` (línea 24 del ejemplo anterior) indicando a continuación qué excepción o excepciones queremos propagar. En el caso del ejemplo, se trata de la excepción de E/S `IOException`, que puede venir generada por la sentencia de lectura de la línea 31. En Java es OBLIGATORIO tratar o propagar las excepciones, por lo que tendremos que incluir líneas semejantes cuando usemos métodos que puedan generar excepciones (esto está completamente documentado en la API correspondiente).

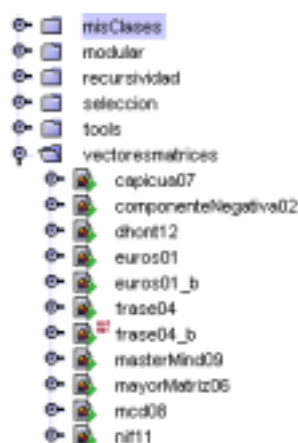
### 3.3 Uso de otras clases no pertenecientes a la API

En múltiples ocasiones utilizaremos alguna clase que no está incluida en la API (por ejemplo, la clase `TextReader`, para E/S por teclado o fichero) y que será necesario "importar" a nuestro programa. La forma más trivial de hacerlo es copiando dicha clase en la misma carpeta donde se encuentra nuestro programa. Así, en el siguiente ejemplo, cualquiera de los programas del paquete/carpeta `vectoresmatrices` puede hacer uso de la clase `TextReader`.



(ver el ejemplo anterior del programa `nif11`, donde se usa la clase `TextReader` en la línea 26, y no se incluye ninguna sentencia de importación específica).

Podemos tener todas las clases de uso frecuente en una carpeta separada y hacer uso de las mismas sin necesidad de tener copia en cada una de las carpetas. Siguiendo el ejemplo anterior, sería la situación en que no tuviésemos la clase `TextReader` en el paquete/carpeta `vectoresmatrices`, sino, por ejemplo en otro llamado `misClases` (siempre dentro del mismo FileSystem).



Para poder utilizar correctamente ahora la clase `TextReader` en el programa `nif11` deberá incluirse una sentencia `import` que indique la trayectoria dónde se encuentra guardada dicha clase. Esta trayectoria siempre será relativa al FileSystem, por lo que en este caso bastará con indicar una nueva línea 15

```

14 import java.io.*;
15 import misClases.*;
16 public class nif11 {
17
18     /** Creaes new frase04 */
19     public nif11() {
20     }
21
22     /**
23     * @param args the command line a
24     */

```

con dicha trayectoria (el símbolo \* indica que se importen todas las clases del paquete `misClases`, pero podría especificarse únicamente `misClases.TextReader`).

Esto mismo puede realizarse, como es lógico, con las clases que vayamos construyendo nosotros mismos. Podéis ver un ejemplo (un poco más complejo) de esta situación en el paquete `modular` de las soluciones de los ejercicios.


#### 4. HERRAMIENTAS DE DEPURACIÓN

Las herramientas de depuración se utilizan para detectar la/s sentencia/s en donde se han producido errores en el diseño de un algoritmo. La tarea de depuración consiste básicamente en explorar el algoritmo, ejecutándolo paso a paso y comparando en todo momento los valores que van tomando las distintas variables con los valores esperados. Es necesario, por tanto, haber analizado con anterioridad algunos casos de prueba que permitan anticipar en todo momento los valores que deben tomar las variables y saber si la progresión del algoritmo es correcta o no.

Normalmente la depuración se realizará ante la presencia de errores de ejecución o lógicos, ya que los de sintaxis suelen ir acompañados de algún tipo de mensaje que facilita su localización.

Los botones relacionados con la depuración son los siguientes (acercando el ratón a cada uno de ellos puedes ver la etiqueta con el nombre que le asocia Sun Forte):



El inicio de la depuración de un programa se realiza mediante el botón de INICIO (Step Into, F7) , que compilará el programa e iniciará su ejecución deteniéndose en la primera sentencia del programa.<sup>1</sup> La detención se indica mediante un sombreado de color azul en la ventana de edición:

```
public static void main (String args[]) throws IOException{
    double importe, euros[]={500, 200, 100, 50, 20, 10, 5, 2, 1, 0.5, 0.2, 0.1};
    int i, billetesmoneda[] = new int[15];
```

Conviene indicar que la sentencia sombreada todavía NO SE HA EJECUTADO, sin o que será la siguiente en ejecutarse.

Además, el entorno se habrá trasladado al escritorio de depuración (Debugging):

<sup>1</sup> En estas notas usaremos como ejemplo el programa `euros01_b`, del apartado de vectores y matrices



en el cual tendremos también acceso a las ventanas siguientes:

- Depuración: **Debugger Window [Variables]**: va a permitirnos ver los valores que toman las variables que se hayan definido en el programa, y también evaluar cualquier otra expresión que nos interese.
- E/S: idéntica a cuando se ejecuta un programa normalmente.

Para proseguir con la ejecución paso a paso disponemos de varias opciones, que describiremos seguidamente.

#### **4.1 PASO SIMPLE**

La forma habitual de ejecutar el programa sentencia a sentencia será pulsando el botón STEP OVER (F8) . A cada pulsación del botón, el flujo del programa algoritmo avanza una sentencia. El símbolo de la sentencia que se va a ejecutar en cada momento es destacado en color azul.

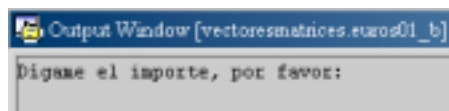
En cualquier momento puede pararse la ejecución paso a paso pulsando el botón FINISH (Mayúsculas+F5) :

#### Ejercicio

Ejecutar paso a paso el programa hasta la sentencia

```
importe=teclado.readFloat();  
inclusive.
```

En el momento que ejecutamos una sentencia de Entrada/Salida, la información correspondiente aparece en la ventana de Salida:



a lo cual se responderá de la manera habitual.

#### Ejercicio

Introducid el valor 123.45, y seguid ejecutando paso a paso el programa hasta la sentencia

```
importe-=billetemoneda[i]*euros[i];
```

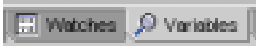
---

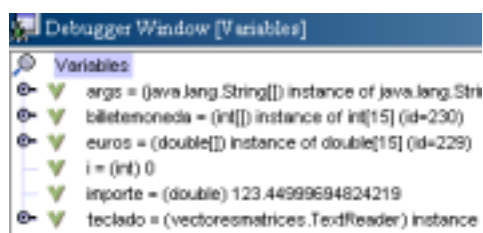
---


inclusive.

---

## 4.2 VENTANA DE DEPURACIÓN

Observaréis que las variables que han sido definidas en el programa, junto con sus valores aparecen en la Ventana de DEPURACIÓN, si hemos seleccionado en la parte inferior la solapa VARIABLES . En nuestro ejemplo, los valores deben ser los siguientes:




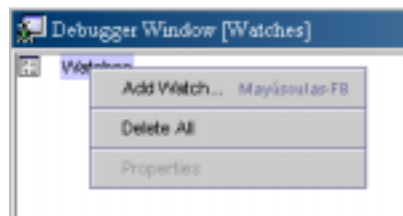
Podéis observar cómo los valores son correctos. Para el caso de vectores como **billeteMoneda** o **euros**, se pueden visualizar todas sus componentes pulsando en el símbolo , que permite desplegarlas y replugarlas a conveniencia.

### Ejercicio

Completad la ejecución paso a paso, fijándoos en los valores de las variables **i**, **importe** y **billeteMoneda[i]**, que determinan el número de veces que se ejecuta el lazo y el cumplimiento o no de la sentencia de selección.

---

Para visualizar sólo algunas de las variables, o bien expresiones se utiliza la solapa WATCHES . Para introducir una expresión se activa el siguiente menú contextual



sobre la palabra WATCHES, y seleccionamos la opción ADD WATCH.

En el cuadro de diálogo correspondiente se escribe la expresión que deseamos evaluar.

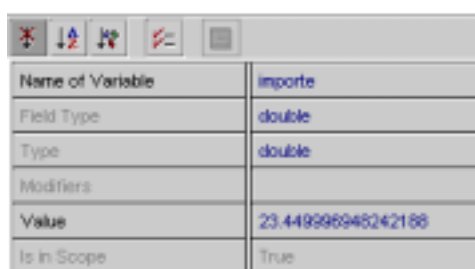
## Ejercicio

Ejecutad paso a paso nuevamente el programa evaluando las variables `i`, `importe` y `billetemoneda[i]` en el apartado WATCHES de la ventana de depuración.

---

Para eliminar una expresión que no nos interesa ya evaluar basta con seleccionarla y pulsar el botón SUPRIMIR, o hacerlo con el menú contextual de dicha expresión.

También pueden modificarse las expresiones sin más que seleccionarlasy cambiarlas en la parte derecha de la ventana, en el botón NAME OF VARIABLE



Name of Variable	importe
Field Type	double
Type	double
Modifiers	
Value	23.449996948242188
Is in Scope	True

Si deseamos modificar su valor, pulsaremos el botón VALUE.

### **4.3 OTRAS FORMAS DE VISUALIZACIÓN DE VARIABLES**

Existe otra alternativa para ver rápidamente el valor de una variable: basta con acercar (SIN PULSAR) el cursor al nombre de dicha variable, y automáticamente aparecerá una etiqueta que indica el tipo de variable y su valor (es decir, la misma información que aparece en la ventana de depuración VARIABLES).


### **4.4 EJECUCIÓN NORMAL DE CONJUNTOS DE SENTENCIAS**

Esta posibilidad es útil cuando se sabe con certeza que una parte del algoritmo está correctamente diseñada y por tanto, la ejecución paso a paso de dicha parte no es necesaria. Consiste en ejecutar las sentencias que se suponen correctas de la forma habitual, para detener la ejecución del programa en las sentencias donde suponemos que se encuentra el error.




#### **4.4.1 Puntos de ruptura**

Pueden establecerse PUNTOS DE RUPTURA (breakpoints) en las sentencias del algoritmo, de modo que el programa se ejecutará con normalidad hasta dicho punto, y a partir de ahí puede realizarse alguna de las siguientes posibilidades:

- retomarse la ejecución normal
- seguir paso a paso
- establecer un punto de ruptura en una sentencia posterior del algoritmo y continuar hasta él

El punto de ruptura se fija antes del inicio de la ejecución mediante el botón TOGGLE BREAKPOINT , situando previamente el cursor en la línea donde queremos fijar el punto de ruptura, que quedará resaltada en rojo:

```
for (i=0; i<euros.length-1; i++) {
    billetemoneda[i]= (int)(importe/euros[i]);
    importe--billetemoneda[i]*euros[i];
    if (billetemoneda[i]!=0)
        System.out.println(billetemoneda[i]+" de "+euros[i]);
}
```

Para realizar la ejecución hasta el punto de ruptura se utiliza el botón INICIO DE DEPURACIÓN (F5)  en lugar del habitual de ejecución F6 . Una vez detenido el programa en el punto correspondiente puede reanudarse la ejecución hasta un punto de ruptura posterior o hasta el final del algoritmo pulsando CONTINUE (F5) .

---


### Ejercicio

Establecer un punto de ruptura en la sentencia indicada anteriormente y ejecutar el programa desde ese punto. A continuación fijar otro punto de ruptura en el último if, ejecutar hasta dicho , paso a paso, siguiendo un par hasta eble del algoritmo de resta de dos matrices, y continuar paso a paso a partir de ahí.

---

Cabe señalar que los puntos de ruptura pueden añadirse o quitarse a voluntad durante el proceso de depuración.

#### 4.4.2 Ejecución hasta el cursor

Otra alternativa, que en ocasiones puede resultar más cómoda que los puntos de ruptura, consiste en situar el cursor en un punto del programa y ejecutarlo con normalidad el programa hasta dicho punto. Para ello disponemos del botón EJECUTAR HASTA EL CURSOR (F4) . Con ello la ejecución del programa se detiene en dicho punto, pudiéndose proseguir a partir de ahí de cualquiera de las

formas conocidas (una nueva ejecución hasta otra posición del cursor, paso a paso, fijando puntos de ruptura, ...).