

En este documento hay información sobre el manejo de **Eclipse** y de **Netbeans** para crear aplicaciones en Java (véanse los apéndices 1 y 2).

## (3.4) escritura de programas Java

### (3.4.1) codificación del texto

Todos el código fuente Java se escriben en documentos de texto con extensión **.java**. Por lo tanto para escribirlo basta cualquier editor de texto como el **bloc de notas** de Windows o los editores **vi** o **gedit** de Linux.

Al ser un lenguaje multiplataforma y especialmente pensado para su integración en redes, la codificación de texto utiliza el estándar **Unicode**; lo que implica que los programadores y programadoras hispanohablantes podemos utilizar sin problemas símbolos de nuestra lengua como la eñe o las vocales con tildes o diéresis a la hora de poner nombre a nuestras variables.

La codificación Unicode<sup>4</sup> usa normalmente 16 bits (2 bytes por carácter) e incluye la mayoría de los códigos alfabéticos del mundo.

### (3.4.2) notas previas

Los archivos con código fuente en Java deben guardarse con la extensión **.java**. Como se ha comentado cualquier editor de texto basta para crearle. Algunos detalles importantes son:

- ◆ En java (como en C) hay diferencia entre mayúsculas y minúsculas.
- ◆ Cada línea de código debe terminar con ;
- ◆ Una instrucción puede abarcar más de una línea. Además se pueden dejar espacios y tabuladores a la izquierda e incluso en el interior de la instrucción para separar elementos de la misma.
- ◆ Los comentarios; si son de una línea debe comenzar con // y si ocupan más de una línea deben comenzar con /\* y terminar con \*/

```
/* Comentario  
de varias líneas */  
//Comentario de una línea
```

También se pueden incluir comentarios **javadoc** (se explican en detalle más adelante)

<sup>4</sup> Para más información acudir a <http://www.unicode.org>

- ◆ `export PATH="$PATH:$JAVA_HOME/bin"`
- ◆ `export CLASSPATH="rutas a los directorios en los que se almacenarán programas en Java"`

En el caso de la variable `CLASSPATH`, es muy conveniente que su ruta incluya el directorio actual (el símbolo `.~.`). Un ejemplo de uso de `CLASSPATH` en el archivo `bashrc` sería:

```
export CLASSPATH="/usr/lib/jvm/java-6-sun:."
```

Gracias a ello podremos colocar programas Java y ejecutarlos sin problemas en la carpeta raíz del SDK y en la carpeta actual en la que nos encontremos.

### instalación del código fuente de las bibliotecas

Como otros lenguajes, Java se compone de una serie de bibliotecas. En el caso de Java estas bibliotecas son inmensas y están comprimidas en el archivo `src.jar`.

Es muy habitual en Java el uso de archivos **JAR**. En realidad son archivos que aglutan código fuente en java, comprimiéndoles en formato **ZIP**. Para poder acceder al código fuente en sí debemos descomprimir dicho archivo. Lo cual se hace (en cualquier sistema):

- (1) Yendo a la carpeta o directorio en el que se encuentra el SDK (es decir yendo a `JAVA_HOME`).
- (2) Allí se encuentra el archivo `src.jar`
- (3) Descomprimirlle con la orden `jar xvf src.jar`

Actualmente el archivo se llama `src.zip`, con lo que se puede descomprimir con cualquier programa capaz de manejar archivos ZIP.

### documentación

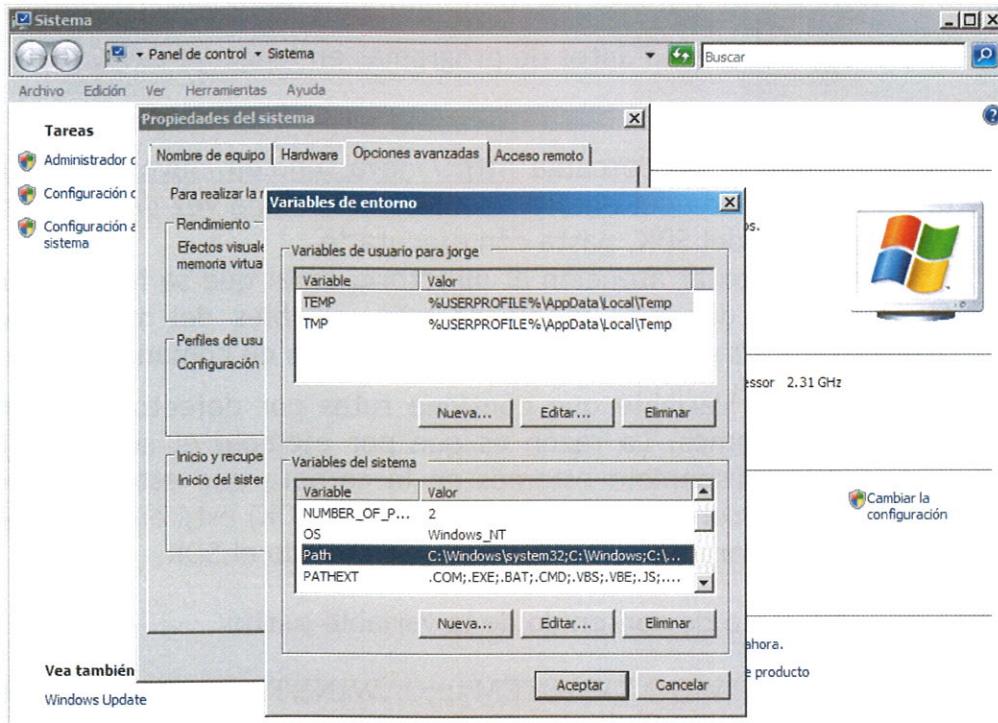
[docs.oracle.com/javase/8/](http://docs.oracle.com/javase/8/)

En <http://java.sun.com/docs> se encuentra la documentación oficial de Java. La más interesante es la documentación del API de Java (clases estándar de Java). Aunque inicialmente es conveniente pasar por **Get Started**. La única pega es que no hay documentación en castellano.

Otra posibilidad (más recomendable incluso), es descargarse la documentación en un archivo **zip**. De esta forma no dependeremos de la conexión a Internet para consultar la documentación oficial. Esto se puede realizar de la página de descarga <http://java.sun.com/javase/downloads/index.jsp>

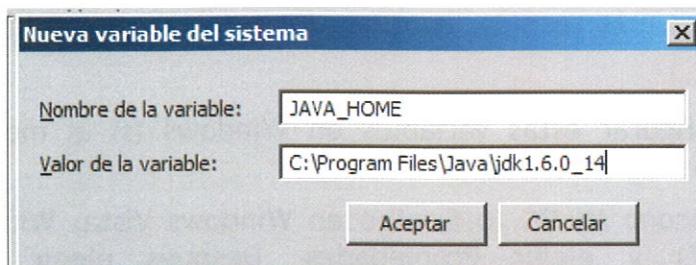
### (3.3.4) entornos de trabajo

El código en Java se puede escribir en cualquier editor de texto. Y para compilar el código en bytecodes, sólo hace falta descargar la versión del JDK deseada. Sin embargo, la escritura y compilación de programas hecha de esta forma es un poco incomoda. Por ello numerosas empresas fabrican sus propios



**Ilustración 3-3, El cuadro de las variables del Sistema en Windows Server 2008**

- (2)** Dentro de este cuadro, ya estará la variable PATH. Habrá que elegirla y pulsar en modificar. Sin borrar nada de lo que contiene, debemos añadir al final del texto el símbolo ; y después la ruta al directorio bin dentro de la carpeta del SDK (por ejemplo C:\Program Files\Java\jdk1.6.14\bin).
- (3)** Tras aceptar el cuadro anterior, podremos pulsar en Nueva para añadir la variable JAVA\_HOME indicando como valor la ruta al SDK.



**Ilustración 3-4, Ejemplo de configuración de la variable JAVA\_HOME en Windows**

- (4)** Hacer el mismo proceso para la variable CLASSPATH

Para comprobar la versión de Java basta con ir al símbolo del sistema Windows y escribir **java -version**

- ◆ **Java Servlets.** Herramienta para crear aplicaciones de servidor web (y también otros tipos de aplicaciones).
- ◆ **Java Cryptography.** Algoritmos para encriptar y desencriptar.
- ◆ **Java Help.** Creación de sistemas de ayuda.
- ◆ **Jini.** Permite la programación de electrodomésticos.
- ◆ **Java card.** Versión de Java dirigida a pequeños dispositivos electrónicos.
- ◆ **Java IDL.** Lenguaje de definición de interfaz. Permite crear aplicaciones tipo **CORBA** (plataforma de desarrollo de sistemas distribuidos)
- ◆ **Clases para la creación de colecciones**

#### **Java 1.3 (J2SE 1.3)**

- ◆ Se utiliza la máquina virtual de **Hotspot** (más rápida y segura).
- ◆ Se modifica RMI para que trabaje con CORBA
- ◆ **JPDA, Java Platform Debugger Architecture**

#### **Java 1.4 (J2SE 1.4)**

- ◆ Aparecen las aserciones (**assert**)
- ◆ Expresiones regulares estilo **Perl**.
- ◆ **NIO.** Nuevo interfaz de entrada y salida de datos.
- ◆ **JAXP.** API de desarrollo de documentos **XML**.

#### **Java 1.5 (J2SE 1.5)**

- ◆ Aparecen las plantillas
- ◆ Metadatos
- ◆ **Autoboxing**, conversión automática de tipos a tipos envolventes.
- ◆ **Enumeraciones**
- ◆ Argumentos variables (**varargs**)
- ◆ Mejora del bucle **for**

#### **Java 1.6 (Java SE 6)**

- ◆ Combinación con otros lenguajes (**PHP, Ruby, Perl,...**)
- ◆ Últimas especificaciones de **JAX-WS 2.0, JAXB 2.0, STAX** y **JAXP** para crear servicios web.

### Java EE

**Java Enterprise Edition.** Todavía conocida como **J2EE**. Pensada para la creación de aplicaciones Java empresariales y del lado del servidor. Su última versión es la 1.4

### Java ME

**Java Mobile Edition.** También conocida como **J2ME**. Pensada para la creación de aplicaciones Java para dispositivos móviles.

## (3.3) empezar a trabajar con Java

### (3.3.1) el kit de desarrollo Java (SDK)

Para escribir en Java hacen falta los programas que realizan el precompilado y la interpretación del código. Hay entornos que permiten la creación de los bytecodes y que incluyen herramientas con capacidad de ejecutar aplicaciones de todo tipo. El más famoso (que además es gratuito) es el **Java Developer Kit (JDK)** de Sun, que se encuentra disponible en la dirección <http://java.sun.com/j2se/downloads> <http://www.oracle.com>.

Actualmente ya no se le llama así sino que se le llama **SDK** y al descargarlo de Internet hay que elegir la plataforma deseada (SE, EE o ME).

### (3.3.2) versiones de Java

Como se ha comentado anteriormente, para poder crear los bytecodes de un programa Java, hace falta el SDK de Sun. Sin embargo, Sun va renovando este kit actualizando el lenguaje. De ahí que se hable de Java 1.1, Java 1.2, etc. Los nombres de los distintos SDK y del lenguaje correspondiente, están reflejados en esta tabla:

| Versión del SDK para la versión estándar de Java | Nombre que se le da al kit de desarrollo |
|--|--|
| 1.1  | JDK 1.1                                  |
| 1.2  | J2SE 1.2                                 |
| 1.3  | J2SE 1.3                                 |
| 1.4  | J2SE 1.4                                 |
| 1.5  | J2SE 1.5                                 |
| 1.6  | Java SE 6                                |
| 1.7  | Java SE 7                                |

Desde la versión 1.2 se habla de Java 2. Desde la versión 1.6 se ha abandonado la terminología **Java 2** y ahora se habla de **Java 6** y **Java 7** para las versiones 1.6 y 1.7 del kit de desarrollo.

Cada versión tiene varias revisiones, así la versión 1.6.7 del SDK indica versión 6 de Java, revisión 7.

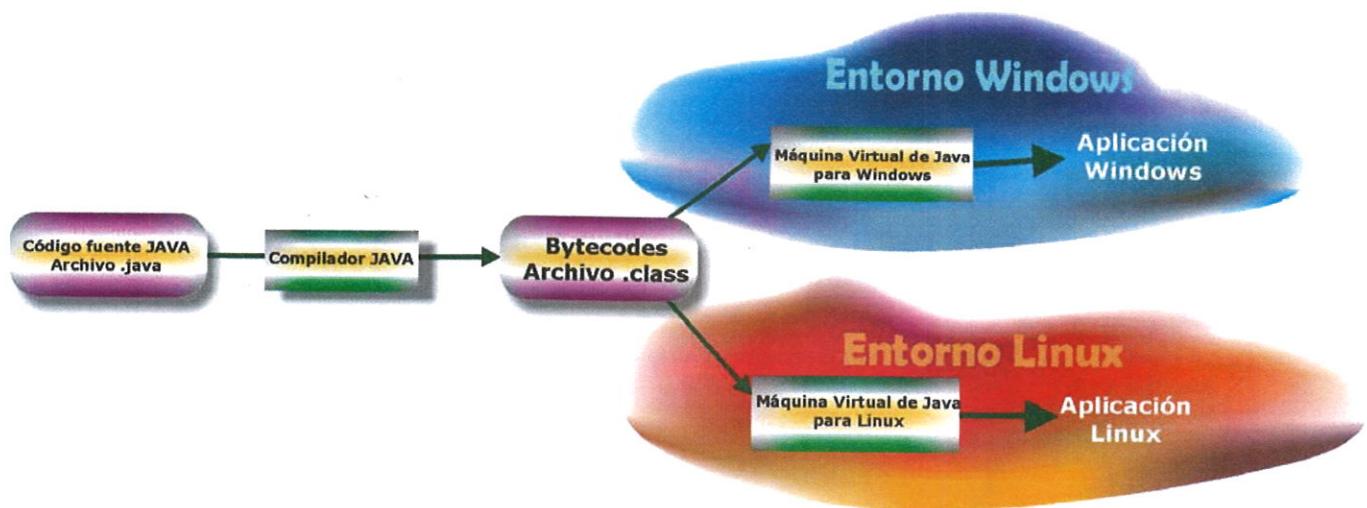


Ilustración 3-2, Proceso de compilación de un programa Java

### (3.2.2) seguridad

Al interpretar el código, el JRE puede delimitar las operaciones peligrosas, con lo cual la seguridad es fácilmente controlable. Además, Java elimina las instrucciones dependientes de la máquina y los **punteros** que generaban terribles errores en C y la posibilidad de generar programas para atacar sistemas. Tampoco se permite el acceso directo a la memoria y recursos del ordenador.

La primera línea de seguridad de Java es un **verificador del bytecode** que permite comprobar que el comportamiento del código es correcto y que sigue las reglas del lenguaje Java. Normalmente los compiladores de Java no pueden generar código que se salte las reglas de seguridad de Java. Pero un programador **malévolos** podría generar artificialmente bytecode que se salte las reglas. El verificador intenta eliminar esta posibilidad.

Hay un segundo paso que verifica la seguridad del código que es el **verificador de clase** que es el programa que proporciona las clases necesarias al código. Lo que hace es asegurarse que las clases que se cargan son realmente las del sistema original de Java y no clases creadas reemplazadas artificialmente.

Finalmente hay un **administrador de seguridad** que es un programa configurable que permite al usuario indicar niveles de seguridad a su sistema para todos los programas de Java.

Hay también una forma de seguridad relacionada con la confianza. Esto se basa en saber que el código Java procede de un sitio de confianza y no de una fuente no identificada. Se consigue gracias a que en Java se permite añadir **firmas digitales** al código para verificar la autoría del mismo.

### **(3.1.3) Java y JavaScript**

Una de las confusiones actuales la provoca el parecido nombre que tienen estos dos lenguajes. Sin embargo no tienen mucho que ver entre sí. Sun creó Java y la empresa **Netscape** creó JavaScript. Java es un lenguaje completo que permite realizar todo tipo de aplicaciones. JavaScript es un lenguaje que permite incrustar código dentro de las páginas web.

La finalidad de JavaScript es mejorar las páginas web, hacerlas más vistosas y dinámicas. La finalidad de Java es crear aplicaciones de todo tipo (aunque está muy preparado para crear sobre todo aplicaciones en red).

Aunque la sintaxis tiene elementos en común, desde luego no se parece tanto. De hecho Javascript es mucho más fácil que Java.

## **(3.2) características de Java**

### **(3.2.1) construcción de programas en Java. bytecodes**

#### **compilación tradicional**

En el mundo de la programación siempre se ha hablado de **lenguajes compilados** y de **lenguajes interpretados**. En el segundo caso, un programa intérprete se encarga de traducir cada línea al código máquina correspondiente. Los lenguajes interpretados a partir de los setenta se han dejado de usar porque no eran los apropiados para conseguir código eficiente.

Por el contrario, los lenguajes compilados producen código máquina analizando todas las líneas de código en conjunto. Los compiladores buscan el mejor código máquina posible. El resultado del proceso de compilación (en realidad de compilación y enlazado) es un **archivo ejecutable**.

Un archivo ejecutable es un programa que se puede lanzar directamente en el sistema operativo; en el caso de Windows o Linux simplemente con hacer doble clic sobre el archivo, se ejecutan sus instrucciones. La ventaja es que los programas ejecutables no necesitan compilarse de nuevo, son programas terminados. El problema es que los sistemas operativos utilizan diferentes tipos de archivos ejecutables: es decir, un archivo ejecutable en Linux no sería compatible con Windows.

C++ pasó a ser el lenguaje de programación más popular a principios de los 90 y sigue siendo un lenguaje muy utilizado. Muchas personas le consideran el lenguaje de programación más potente.

Otras adaptaciones famosas de lenguajes clásicos a lenguajes orientados a objetos, fueron:

- ◆ El paso de **Pascal** a **Turbo Pascal** y posteriormente a **Delphi**.
- ◆ El paso de **Basic** a **QuickBasic** y después a **Visual Basic**.

A pesar de las evidentes ventajas del lenguaje C++. Tiene sus serios inconvenientes.

- ◆ Su complejidad
- ◆ El hecho de ser un **lenguaje híbrido**, es decir que permite programar de forma no orientada a objetos, lo que provoca malas prácticas de programador.
- ◆ Los punteros, que requieren un especial cuidado por parte de la programadora o programador, ya que son los responsables de los errores más peligrosos y difíciles de detectar.
- ◆ El que sea un lenguaje apto para crear programas dañinos como virus y programas espías.
- ◆ No es un lenguaje apto para transmitirse en redes de ordenadores; especialmente en Internet (porque al ser compilado requiere cargar todo el código para ser compilado).

La llegada de Internet propició la creación de lenguajes más aptos para su uso en esta red de redes.

BREVE HISTORIA (PDF)  
JAMES GOSLING

### (3.1.2) La llegada de Java

En 1991, la empresa **Sun Microsystems** crea el lenguaje **Oak** (de la mano del llamado **proyecto Green**). Mediante este lenguaje se pretendía crear un sistema de televisión interactiva. Este lenguaje sólo se llegó a utilizar de forma interna en la empresa. Su propósito era crear un lenguaje independiente de la plataforma para uso en dispositivos electrónicos.

Se intentaba con este lenguaje paliar uno de los problemas fundamentales del C++; que consiste en que al compilar se produce un fichero ejecutable cuyo código sólo vale para la plataforma en la que se realizó la compilación. Sun deseaba un lenguaje para programar pequeños dispositivos electrónicos. La dificultad de estos dispositivos es que cambian continuamente y para que un programa funcione en el siguiente dispositivo aparecido, hay que rescribir el código. Por eso Sun quería crear un lenguaje **independiente del dispositivo**.

En 1995 Oak pasa a llamarse **Java**. Java es un importante exportador de café; por eso en EEUU se conoce como Java al café, tomarse una taza de Java

## (3.1) historia de Java

### (3.1.1) los antecedentes de Java. influencia de C y C++

Java es un lenguaje de programación que se desarrolló para satisfacer las nuevas necesidades que requería la creación de aplicaciones a finales de los 90.

Desde los primeros lenguajes aparecidos en los años cincuenta, hasta la aparición de Java, la ciencia de la creación de programas ha sufrido numerosas transformaciones. Todas ellas se basan en intentar que los programadores y programadoras consigan trabajar de la forma más eficiente posible.

La búsqueda del lenguaje perfecto es la búsqueda del lenguaje que sea más fácil de aprender y que otorgue más posibilidades a aquellos programadores y programadoras que lo utilicen.

En general ambos conceptos han dado lenguajes muy diversos. Por ejemplo, el lenguaje **Basic** es un lenguaje muy fácil de aprender, pero en cuanto se quieren resolver problemas complicados, resulta ineficaz. Por otro lado el lenguaje **C** es un lenguaje muy poderoso, capaz de crear todo tipo de aplicaciones; pero es bastante más difícil de aprender.

Java intenta cumplir ambas premisas, pero de forma equilibrada: ni es un lenguaje muy fácil de aprender, ni es un lenguaje capaz de realizar todo tipo de aplicaciones. En realidad Java es uno de los muchos lenguajes influenciados por el exitoso **lenguaje C**. Este lenguaje ha sido el favorito de los creadores de aplicaciones (especialmente de sistemas) en los años 60 y 70.

#### la influencia del lenguaje C

La aparición del lenguaje **Fortran**, supuso la creación del primer lenguaje de alto nivel. Por primera vez el programador podía programar un poco más alejado de la lógica de la máquina, es decir cada vez más lejos del lenguaje de unos y ceros que es el único que los computadores reconocen.

Poco a poco aparecieron cada vez más lenguajes con la pretensión de mejorar la forma de programar (**Lisp**, **Pascal**, **Fortran**, **Cobol**,...). Algunos de ellos siguen vigentes incluso hoy en día. La mayoría se especializaron en diferentes tipos de aplicaciones (Lisp para aplicaciones de ingeniería, Pascal para aprendizaje de la ciencia de la programación, Cobol para aplicaciones de gestión,...).

El caso es que para crear aplicaciones de alto rendimiento y de sistema, los programadores seguía utilizando el lenguaje **Ensamblador**. Por ello a finales de los 60 aparece el lenguaje C.

|   |           |
|---|-----------|
| <b>(3.9) tipos de datos primitivos</b>              | <b>32</b> |
| (3.9.1) enteros                                     | 32        |
| (3.9.2) números en coma flotante                    | 33        |
| (3.9.3) booleanos                                   | 34        |
| (3.9.4) caracteres                                  | 34        |
| (3.9.5) conversión entre tipos ( <i>casting</i> )   | 35        |
| (3.9.6) ámbito de las variables                     | 35        |
| <b>(3.10) operadores</b>                            | <b>36</b> |
| (3.10.1) introducción                               | 36        |
| (3.10.2) operadores aritméticos                     | 36        |
| (3.10.3) operadores condicionales                   | 37        |
| (3.10.4) operadores de BIT                          | 38        |
| (3.10.5) operadores de asignación                   | 38        |
| (3.10.6) operador ?                                 | 39        |
| (3.10.7) precedencia                                | 39        |
| <b>(3.11) constantes</b>                            | <b>40</b> |
| <b>(3.12) lectura y escritura por teclado</b>       | <b>41</b> |
| (3.12.1) escritura                                  | 41        |
| (3.12.2) lectura                                    | 41        |
| <b>(3.13) la clase Math</b>                         | <b>43</b> |
| (3.13.2) números aleatorios                         | 45        |
| <b>Apéndice (I) Eclipse</b>                         | <b>46</b> |
| (I.i) entornos de desarrollo integrado (IDE)        | 46        |
| (I.ii) descarga de Eclipse                          | 47        |
| (I.iii) aspecto de Eclipse                          | 50        |
| (I.iii.i) las perspectivas de Eclipse               | 50        |
| (I.iv) crear proyectos en Eclipse                   | 51        |
| (I.iv.i) crear proyectos básicos de Java            | 51        |
| (I.iv.ii) modificar recursos del proyecto           | 53        |
| (I.iv.iii) crear programas Java (clases)            | 53        |
| (I.iv.iv) cambiar el nombre a los elementos de Java | 54        |
| (I.v) ejecución de programas Java                   | 55        |
| (I.vi) ayudas al escribir código                    | 55        |
| (I.vi.i) esquema flotante ( <i>quick outline</i> )  | 55        |
| (I.vi.ii) asistente de contenido                    | 55        |
| (I.vi.iii) plantillas de código                     | 57        |
| (I.vi.iv) dar formato al código                     | 59        |
| (I.vi.v) errores                                    | 59        |
| (I.vii) modificar las preferencias del editor       | 60        |
| (I.vii.i) opciones generales                        | 60        |
| (I.vii.ii) realizar acciones al guardar             | 60        |
| (I.vii.iii) asistencia de contenido                 | 61        |
| (I.vii.iv) coloreado de la sintaxis                 | 61        |
| (I.vii.v) marcar apariciones                        | 61        |
| (I.vii.vi) apartado tecleo ( <i>typing</i> )        | 61        |
| (I.vii.vii) plantillas                              | 62        |
| (I.vii.viii) estilo del código                      | 65        |

---



