Apuntes

de

Java

Tema 5:
API de Java

Uploaded by

Ingteleco

http://ingteleco.webcindario.com
ingtelecoweb@hotmail.com

La dirección URL puede sufrir modificaciones en el futuro. Si no funciona contacta por email

TEMA 5: EL API DE JAVA

5.1.- TIPOS DE API

SUN desarrolla dos tipos de APIs:

- El API núcleo (core API) es el conjunto mínimo de operaciones que los desarrolladores en Java podemos asumir presentes en todas las plataformas Java.
- Las extensiones estándar son las que JavaSoft define y publica pero fuera del API núcleo. Algunas extensiones estándar eventualmente se integrarán en el núcleo.

5.2.- LAS NUEVE LIBRERÍAS

SUN divide los componentes de las distintas APIs en nueve librerías (en las que se mezclan núcleo y extensiones estándar):

JDK API. Es la más fundamental de las librerías. Consta de los ocho paquetes básicos Java: java.applet, java.awt, java.awt.peer, java.awt.image, java.lang, java.net, java.io, java.util.

Java Security API. Seguridad, obviamente.

Java Media API. Todavía en desarrollo. Incorpora posibilidades de gestionar todo tipo de medios (gráficos, vídeo, sonido...) definidos en seis áreas: Java 2D, Java 3D, Java Media Framework, Java Telephony, Java Share, Java Animation.

Java Enterprise API. Incorpora un conjunto de librerías que soportan conectividad a bases de datos y aplicaciones distribuidas: JDBC, IDL, RMI.

Java Commerce API. Se orienta a compras seguras y manipulación de finanzas. Permite el uso de tarjetas de crédito y transacciones electrónicas.

Java Embedded API. Especifica cómo crear un subconjunto del API total de Java. Su intención es usarse en dispositivos incapaces de soportar el API completo. Contiene la funcionalidad mínima (basada en java.lang, java.util y partes de java.io), y define una serie de extensiones para áreas particulares como comunicaciones y un interfaz gráfico de usuario.

Java Server API. Es una extensión estándar que permite el desarrollo de servidores internet e intranet.

Java Management API. Contiene objetos y métodos Java extensibles para construir applets sobre una red corporativa en internet o intranet.

Java Beans API. Definen un conjunto de APIs portables e independientes de plataforma para componentes de software, incluibles en otras arquitecturas de componentes como OLE/COM/Active-X (Microsoft), OpenDoc, o LiveConnect (Netscape).

5.3.- PAQUETE DE LA JDK API

Paquete de lenguaje (java.lang)

Es el paquete central de Java. Contiene las definiciones de las clases:

- **Object**: La clase central de Java.
- **Math**: Clase no instanciable que tiene las constantes E y PI y funciones matemáticas típicas com sin, cos, max, min, random, etc.
- **Wrappers**: Number, Boolean, Character, Integer, Long, Double
- **String** y **StringBuffer**
- **Throwable**: Es la superclase de la jerarquía de las excepciones y errores de Java. Lo veremos.
- **Thread**: La base de la ejecución multi-hilo. También lo veremos.
- **Class**: representa las clases en la aplicación que se está ejecutando: hay una instancia de Class para cada clase cargada.
- **System**

Clase Object

Ya hemos comentado que la clase estándar Object es superclase de todas las demás.

Una variable de tipo Object puede contener una referencia a cualquier objeto, sea instancia de una clase o de un array. Todas las clases y arrays heredan los métodos de la clase Object, que en resumen son:

- *public String toString()* - devuelve una representación en forma de cadena de caracteres del objeto. Es el método al que llama el operador sobrecargado + sobre strings.
- *protected Object clone()* - devuelve una copia del objeto. Es la contrapartida profunda de la asignación = (superficial).
- *public boolean equals(Object obj)* - Compara dos objetos en función de su valor (no referencia). Es, por su parte, la contrapartida profunda de la comparación
- *public final Class getClass()* - Devuelve el objeto Class que representa la clase del objeto. Toda clase tiene un objeto del tipo Class equivalente, que puede utilizarse para recoger todo tipo de información de la clase.
- *public int hashCode()* - devuelve un código hash para cada objeto. Permite la utilización de tablas hash como se especifica en java.util.Hashtable.
- *public final void wait(...)* - se usa en prog. concurrente con threads (y los dos siguientes).
- *public final void notify()*
- *public final void notifyAll()*
- *protected void finalize()* - se llama justo antes de que el objeto se destruya.

Wrappers

Las clases Boolean, Character, Integer, Long, Float y Double (Number es la clase abstracta, padre de todas ellas) son las llamadas wrappers (envoltorios): permiten utilizar tipos primitivos como si fueran clases. Por ejemplo:

```
Float miNumF = new Float( 18.45 );
Character unCar = new Character( '?' );
```

Estas clases tienen operaciones de conversión numérica. Por ejemplo podemos devolver el valor entero (int) correspondiente a un número Float:

```
int unEntero = miNumF.intValue();
```

O algunas utilidades más como manejo de valores infinitos y NaN:

```
double noNumero = Double.NaN;
// atributo de clase (cte)
if (miNumF.isInfinite()) …
```

E incluso conversiones de string a número:

```
Double d = Double.valueOf( "3425234.564356444" );
double d2 = Double.valueOf( "3425234.564356444").doubleValue();
```

Sentido de los wrappers

En cualquier caso podríamos preguntarnos (con razón) ¿y para qué queremos usar un entero, por ejemplo, como si fuera una clase si lo podemos usar sencillamente como un tipo primitivo?

La situación más habitual en la que se usan los wrappers es con estructuras de datos contenedoras. Supongamos que queremos definir una lista enlazada. Tendremos que pensar en el tipo a guardar en esa lista.

Pero pensando en las capacidades de todo lenguaje orientado a objetos, si definimos la lista conteniendo Object como elemento básico, después podremos introducir cualquier instancia de cualquier clase (Object es siempre superclase de todas las demás).

Pero la conversión que no podríamos hacer es utilizar esa lista para guardar enteros (int); bien, sí podremos guardar Integer que también es descendiente de object.

```
public class ListaEnlazada {
     NodoLista l;
     public ListaEnlazada() {
          // l = null;
     }
     public void destruir() {
          l = null;
     }
     public void insertarPrimero( Object o ) {
          l = new NodoLista( o, l );
```

```
    }

    // insertar() devuelve true si hay error, false si no.
    public boolean insertar( Object o, int posicion ) {
    // pre    posicion >= 1
        NodoLista aux = l;
        for (int i = 1; aux != null && i < posicion -1;
            i++ ) {
            aux = aux.siguiente;
        }
        if (posicion == 1) {
            l = new NodoLista( o, l );
            return false;
        }
        else {
            if (aux == null) return true;
            else {
            aux.siguiente = new NodoLista(
                                    o, aux.siguiente );
            return false;
            }
        }
    }

    public String toString() {
        String s = "( ";
        NodoLista aux = l;
        while (aux != null) {
            s = s + aux.elemento + " ";
            aux = aux.siguiente;
        }
        return s + ")";
    }
    /* borrarPrimero() devuelve true si hay error, false         * si no.
     */
    public boolean borrarPrimero() {
        if (l == null) return true;
        else { l = l.siguiente; return false; };
    }
    /* main - prueba de ListaEnlazada */
    public static void main( String[] a ) {
        ListaEnlazada l = new ListaEnlazada();
        l.insertarPrimero( new Integer( 1 ) );
        l.insertarPrimero( new Double( 2.001 ) );
        // lista heterogénea!
        l.insertarPrimero( new Integer( 3 ) );
        l.insertarPrimero( new Integer( 4 ) );
        System.out.println( l );
}   }
```

```
class NodoLista {
     Object elemento;
     NodoLista siguiente;
     NodoLista( Object o, NodoLista l ) {
          elemento = o;
          siguiente = l;
}    }
```

Paquete de E/S (java.io)

Contiene toda la gestión de entradas y salidas a ficheros o strings conceptualizadas como streams (flujos). Veamos la jerarquía con algunas clases:

- InputStream
  - A. ByteArrayInputStream
  - B. FileInputStream
  - C. FilterInputStream
  - D. BufferedInputStream
  - E. DataInputStream
  - F. LineNumberInputStream
  - G. java. PushbackInputStream
  - H. java.io.ObjectInputStream
  - I. PipedInputStream
  - J. SequenceInputStream
  - K. StringBufferInputStream
- OutputStrea
  - L. ByteArrayOutputStream
  - M. FileOutputStream
  - N. FilterOutputStream
  - O. BufferedOutputStream
  - P. DataOutputStream
  - Q. PrintStream
  - R. ObjectOutputStream
  - S. PipedOutputStream
- RandomAccessFile
- Interfaces como DataInput, DataOutput, ObjectInput, ObjectOutput.

```
public class Copia {
     public static void main( String[] args ) throws IOException
     {
          if (args.length < 2)
               System.out.println( "Uso: java Copia fic-ent fic-sal" );
          else
          {
               Copia c = new Copia( args[0], args[1] );
          }
     }
```

```
        public Copia( String ficDesde, String ficHasta )
                                        throws IOException
        {
              long inicio = new Date().getTime();
              System.out.println( "Inicio: " + inicio ) ;
              System.out.println( "Fichero de entrada: " + ficDesde );
              System.out.println( "Fichero de salida: " + ficHasta );
              // Abrimos los ficheros:
              FileInputStream ent = null;
              FileOutputStream sal = null;
              try
              {
                    ent = new FileInputStream( ficDesde );
                    sal = new FileOutputStream( ficHasta );
              } catch ( FileNotFoundException exc )
              {
                    System.out.println( "Error: fichero " +ficDesde+ " no existe." );
                    System.exit( 1 );
              }
              // Leemos y escribimos usando arrays de bytes:
              int num;
              while ( (num = ent.read()) != -1 )
                    sal.write( num );
              // Cerramos los ficheros:
              ent.close();
              sal.close();
              long finale = new Date().getTime();
              System.out.println( "Final: " + finale );
              System.out.println( "Tiempo transcurrido: " + (finale  - inicio) );
        }
  }
```

Paquetes de ventanas (java.awt)

El llamado AWT (Abstract Windowing Toolkit) permite gestionar ventanas independientemente del sistema operativo, junto con todos los componentes visuales y los eventos relacionados con el interfaz gráfico. Esto es, ventanas, cuadros de diálogo, botones, listas, menús, barras de desplazamiento, campos de entrada, etc.

Además del paquete java.awt se estructuran internamente otros cuatro paquetes: java.awt.event, java.awt,image, java.awt.peer y java.awt.datatransfer.

Hablaremos exclusivamente de AWT en futuras clases.

Paquete de applets (java.applet)

Orientado a la gestión de applets. Contiene la clase Applet, superclase de todos los applets, y unas pocas clases más de apoyo.

Paquetes de Utilidades (java.util)

Incluye utilidades varias estándar de Java, como estructuras de datos (tablas de dispersión -hash-, vectores dinámicos, pilas, vectores de bits), fechas y hora, y números aleatorios. También se incluye la clase StreamTokenizer, útil para separar unidades léxicas en cadenas de caracteres.

A partir de la versión 1.1 se incluye también el paquete java.util.zip con clases para compresión y decompresión de ficheros.

Los nombres de algunas clases significativas son:

- **BitSet** - conjunto de bits (como el set de Pascal).
- **Date** - gestión de fecha y hora.
- **Dictionary** - clase abstracta de mappings de claves a valores: T. Hashtable - tabla de dispersión genérica (sobre strings y Objects).
- **Random** - permite generar una cadena de números pseudoaleatorios.
- **Vector** - vector dinámico genérico. U. Stack - Pila genérica
- **StringTokenizer**

Ejemplo: TablaHash

```
public class TablaHash {
     public static void main( String[] a ) {
          Hashtable primos = new Hashtable();
          primos.put("dos", new Integer(2));
          primos.put("tres", new Integer(3));
          primos.put("cinco", new Integer(5));
          // Ahora accedemos
          Integer n = (Integer)primos.get("cinco");
          if (n != null) System.out.println("Cinco = " + n );
     }
}
```

Ejemplo: Tokenizer

```
/* Ejemplo de utilización del StringTokenizer.
 * Modo de utilización: el primer parámetro son los caracteres que
 * separan los tokens. Los demás parámetros marcan los strings a parsear.
 */
public class Tokenizer
{
    public static void main( String argv[] ) {
        StringTokenizer st;
        int count = 0;
        for ( int i = 1; i < argv.length; i++ )
        {
            st = new StringTokenizer( argv[i], argv[0],false );
```

```
                count += st.countTokens( );
                while ( st.hasMoreTokens( ) )
                    System.out.println(st.nextToken( argv[0] ) );
            }
            System.out.println( "Total tokens = " + count );
        }
    }
```

Paquete de red (java.net)

Permite gestionar todas las operaciones relacionadas con una red: acceso a través de internet (URLs, TCP, IP), conectividad, conversión de binario a texto, etc.

5.4.- OTROS PAQUETES

Hasta aquí los paquetes, digamos, centrales en la arquitectura de la máquina virtual. Los que quedan son importantes también, aunque se escapan del objetivo de un curso de introducción a Java:

Paquete de JavaBeans (java.beans)

La arquitectura de JavaBeans permite definir nuevos componentes utilizables en cualquier aplicación. Este paquete contiene todas las operaciones necesarias para ello.

Paquete matemático (java.math)

Tiene algunas funciones matemáticas adicionales, fundamentalmente la gestión de enteros y reales muy grandes (de magnitud no limitada).

Paquete de invocación remota de métodos (java.rmi)

Permite la invocación remota de métodos (RMI, Remote Method Invocation).

Paquete de seguridad (java.security)

Encapsula toda la gestión seguridad en applets y aplicaciones: firmas digitales, encriptación, autentificación, etc.

Paquete de SQL (java.sql)

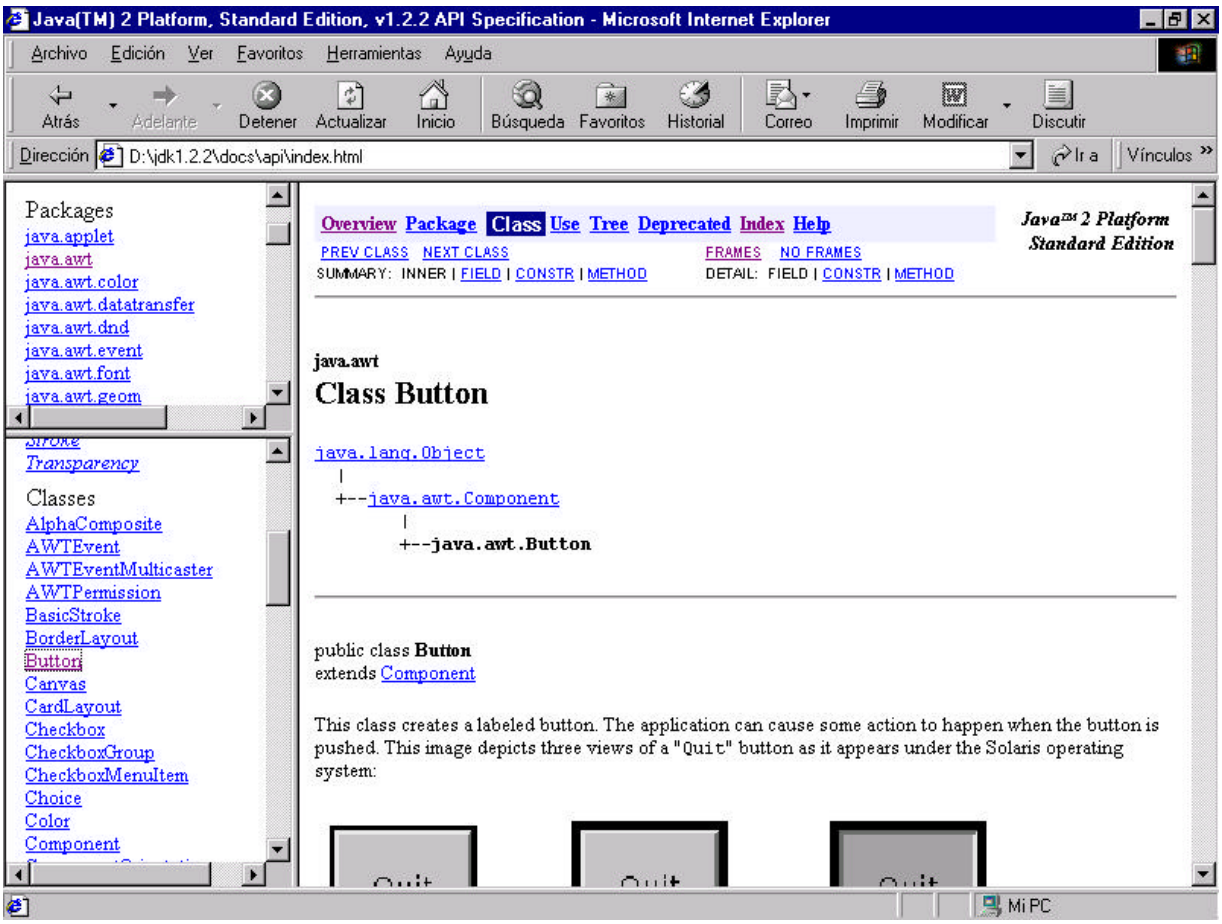
Paquete de gestión de base de datos de Java (JDBC).

Paquete de Texto (java.text)

Paquete que integra un conjunto de utilidades relacionadas con texto (formato de números, fechas, recorrido de strings, etc.)

5.5.- AYUDA DEL API

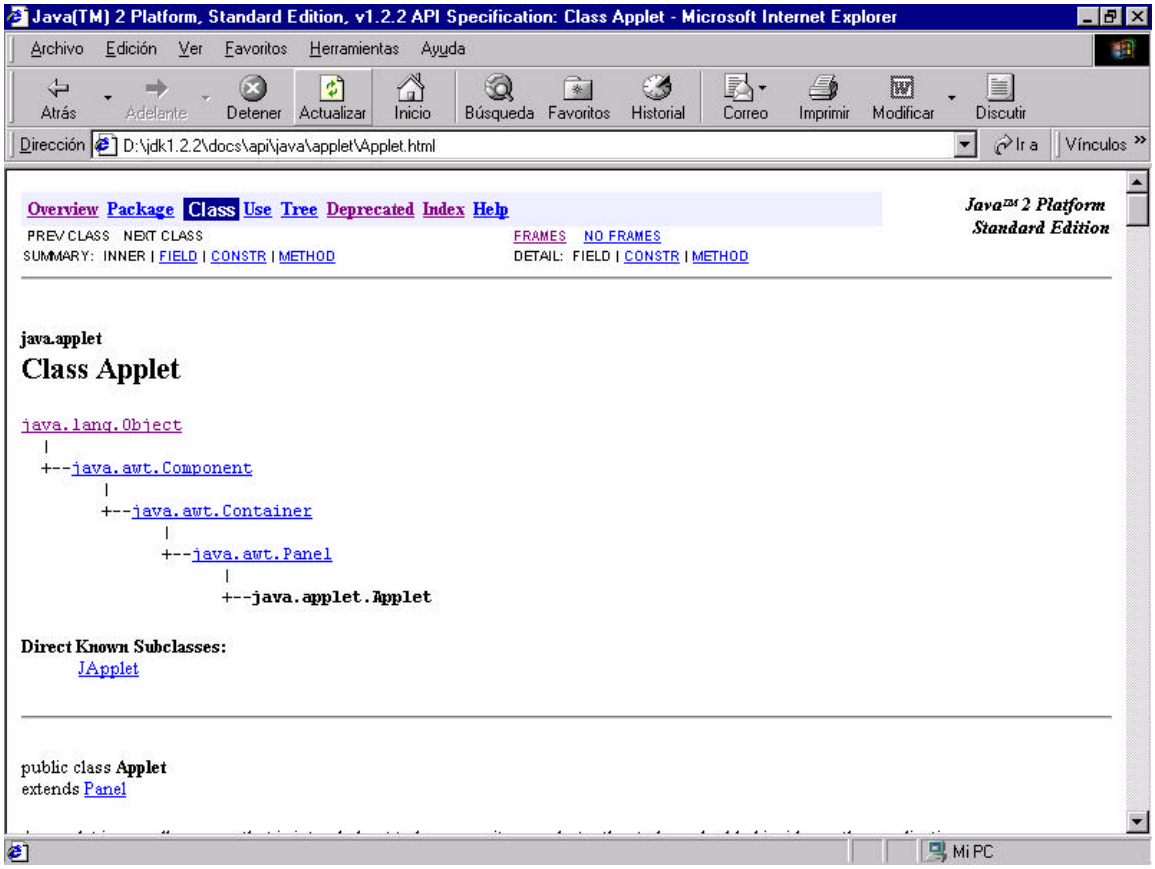
Cuando accedemos a la ayuda del API de Java (C:\<JDK_Directory>\docs\index.html) podemos hacerlo en la versión con o sin frames. Si lo hacemos en la versión con frames (C:\<JDK_Directory>\docs\api\index.html) la pantalla se divide en tres partes, tal y como se muestra a continuación:



En la parte izquierda tenemos dos frames. En el superior están todos los paquetes del API, en orden alfabético, y cuando seleccionamos algún paquete, en la parte inferior aparecen todas las clases e interfaces del paquete, y cuando pinchamos sobre una de ellas, en la parte derecha de la pantalla aparece la ayuda sobre la clase/interface seleccionada.

Si seleccionamos la ayuda sin frames (C:\<JDK_Directory>\docs\api\overview-summary.html) la única diferencia es que no aparecen los frames de paquetes y clases.

A continuación vamos a ver qué información se nos muestra sobre cada clase:



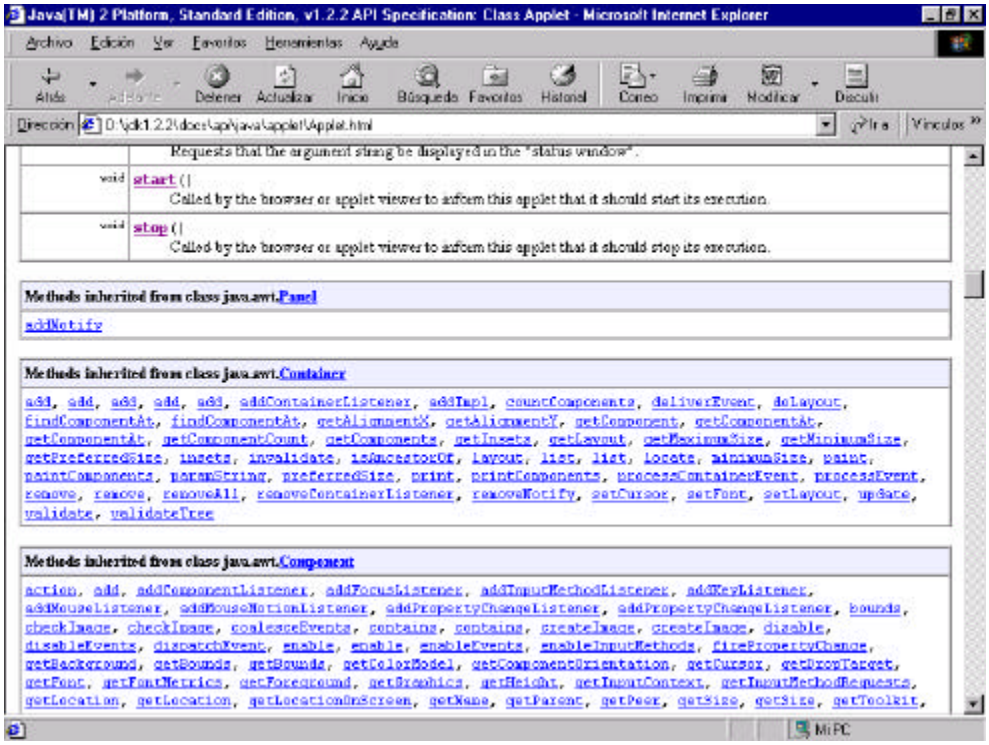
En la parte superior podemos ver un menú para buscar las clases por paquetes (package), ver un árbol de la jerarquía del API (Tree), ver un índice alfabético (index), ver en qué clases se utiliza la clase consultada (Use), ver las clases deprecadas (Deprecated), etc.

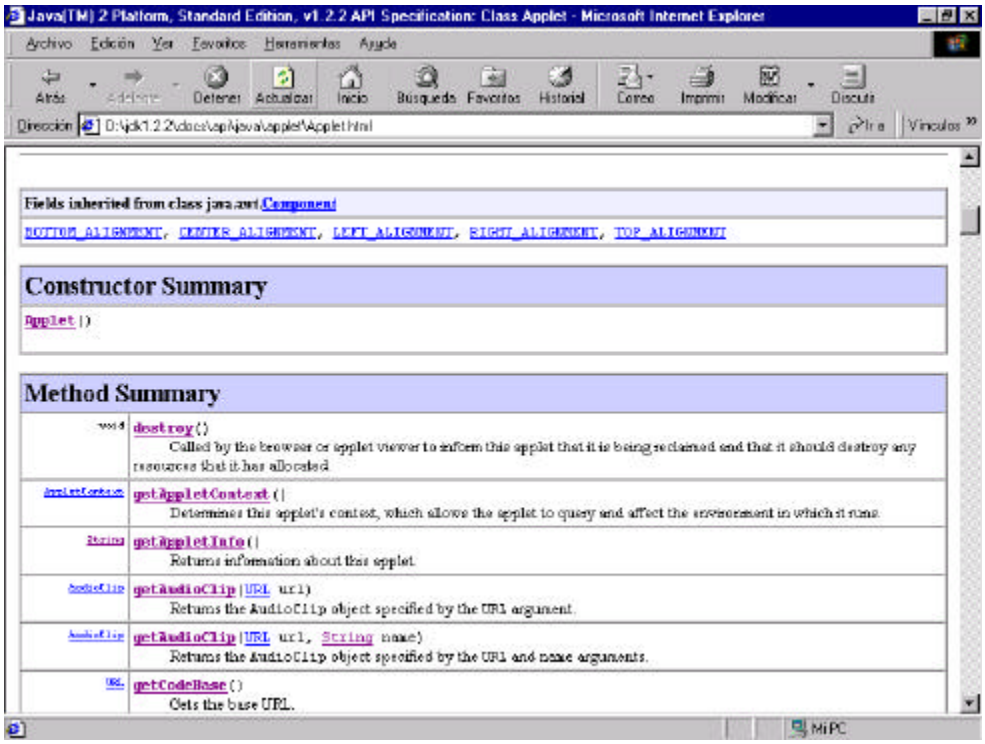
A continuación se muestra el paquete en el que está la clase, en este caso JAVA.LANG, el nombre de la clase (CLASS APPLET), la jerarquía de clases de la que desciende la clase, así como las clases que derivan directamente de ella (clases hijas) y los interfaces que implementa esa clase.

A continuación se presenta el modo en que se muestra la jerarquía de clases de la que desciende la clase APPLET:

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Panel
                    |
                    +--java.applet.Applet
```

Después de una descripción de la clase encontramos unas tablas que muestran los atributos (primero los propios y después los heredados junto con la clase de la que los hereda), constructores y métodos (primero los propios y después los heredados junto con la clase de la que los hereda), junto con una breve descripción de cada uno de ellos:





Finalmente se muestran todos los métodos y atributos, junto con una descripción más detallada, y en algunos casos un ejemplo de uso.