

**Tema**  
**Características de los**  
**Compiladores en los Lenguajes**  
**de Programación**

**Facultad: Ingeniería**  
**Escuela: Computación**  
**Asignatura: Teoría de**  
**Lenguajes de Programación**

## I. OBJETIVOS

- Conocer los conceptos básicos que intervienen en el desarrollo de lenguajes de programación.
- Identificar las diferencias entre compiladores, intérpretes y procesadores intermedios como Java.

## II. INTRODUCCIÓN

La programación de computadoras se realiza en los llamados *lenguajes de programación*, éstos posibilitan la comunicación entre el programador y la computadora, a través de un conjunto de instrucciones u órdenes especificadas por el lenguaje.

Un lenguaje de programación puede definirse como: “Notación formal para describir algoritmos o funciones que serán ejecutados por una computadora”, o bien, un lenguaje para comunicar instrucciones al computador.

Diferentes puntos de vista para clasificar los lenguajes de programación:

- a) Su grado de independencia con la máquina.
- b) La forma de sus instrucciones.
- c) Por generaciones.

Los lenguajes de programación según su **grado de independencia de la máquina** pueden ser:

- Lenguaje máquina (representación binaria o hexadecimal.).
- Lenguaje ensamblador o de bajo nivel (versión simbólica de un lenguaje máquina).
- Lenguaje de medio nivel (lenguaje C).
- Lenguaje de alto nivel (FORTRAN, COBOL, Pascal).

Los lenguajes de programación según **la forma de sus instrucciones** pueden ser:

- **Lenguaje imperativos o procedimentales:** lenguajes orientados a instrucciones (Pascal, C, C++, Ada, FORTRAN).

- **Lenguajes declarativos:**

*Funcionales o aplicativos:* sus construcciones se basan en llamadas a funciones matemáticas, utilizados a menudo en Inteligencia Artificial ( LISP, ML, APL, Haskell ).

*Lógicos:* manejan relaciones entre objetos. Las relaciones se especifican con reglas y hechos. La ejecución de programas lógicos, consiste en la demostración de *hechos* sobre las relaciones por medio de preguntas. (PROLOG es el más utilizado de este tipo).

- **Lenguajes orientados a objetos:** si soportan tipos de datos abstractos y clases. (C++, Smalltalk, Java, Prolog++).

Los lenguajes de programación por **generaciones** son aquellos nombrados como: 1<sup>a</sup>., 2<sup>a</sup>., 3<sup>a</sup>. 4<sup>a</sup>. generación y aquellos identificados como *generación visual* y *generación internet*.

- **Procesadores de Lenguaje**

*Procesadores de lenguaje* es el nombre genérico que reciben todas las aplicaciones en las cuales uno de los datos fundamentales de entrada en un **lenguaje**.

La definición anterior afecta a una gran variedad de herramientas de SW, algunas de ellas son las siguientes:

Traductores (translators)  
Compiladores (compilers)  
Ensambladores (assemblers)  
Interpretes (interpreters)  
Editores (editors)

A continuación se definen algunos procesadores de lenguajes que serán analizados durante la práctica:

### Traductor

Un traductor es un programa que procesa un texto fuente y genera un texto objeto o lenguaje objeto (p. e.: lenguaje máquina, ensamblador, alto nivel).

### Compiladores

Un traductor que transforma textos fuente de lenguajes de alto nivel a lenguajes de bajo nivel se le denomina *compilador*.

- El tiempo que se necesita para traducir un lenguaje de alto nivel a lenguaje objeto se denomina *tiempo de compilación*.
- El tiempo que tarda en ejecutarse un programa objeto se denomina *tiempo de ejecución*.

### Intérpretes

Los intérpretes son programas que simplemente ejecutan las instrucciones que encuentran en el texto fuente. En muchos casos coexisten en memoria el programa fuente y el programa interprete.

Nótese que en este caso todo se hace en tiempo de ejecución. Un ejemplo de lenguaje interpretado es BASIC. La ejecución de un programa compilado es mucho más rápida que la de un programa interpretado. Sin embargo los intérpretes son más interactivos y facilitan la puesta a punto de programas.

### ▪ **Arquitectura de las Computadoras**

En la disciplina de los *procesadores de lenguajes*, los compiladores son los más utilizados por los programadores para el desarrollo de aplicaciones. En el caso particular del desarrollo de compiladores, hay que tener bien definida la arquitectura de la computadora.

¿Cómo se pueden ejecutar las aplicaciones desarrolladas para otras arquitecturas de computadoras en la nueva arquitectura?

El problema planteado anteriormente no sólo es aplicable a la construcción de nuevas arquitecturas, sino también cuando es necesaria la compatibilidad de aplicaciones entre diferentes sistemas operativos y arquitecturas de computadoras.

Algunas de las técnicas utilizadas para realizar migraciones de aplicaciones entre distintas arquitecturas y sistemas operativos se muestran en la siguiente figura:

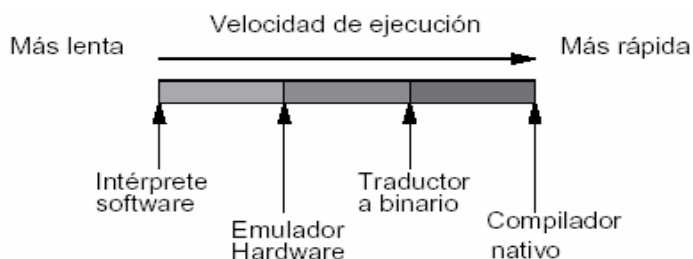


Fig. 4

Un **interprete software** o **emulador de software** es un programa en código binario que lee una a una las instrucciones binarias de la arquitectura antigua de las computadoras, que se encuentran en un fichero ejecutable, y las interpreta. Los intérpretes no son muy rápidos, pero pueden construir y adaptar a distintas arquitecturas sin excesivos costos de desarrollo.

Algunos ejemplos de intérpretes software binario son:

- Emuladores de DOS para distintos sistemas operativos (SoftPC de Insignia Solutions).
- Intérpretes de la máquina abstracta JVM (Java Virtual Machine) para distintas plataformas y que permiten ejecutar los códigos binarios denominados *bytecode* (ficheros con la extensión *.class*).

Un **emulador hardware** trabaja de forma similar a un intérprete de software, pero está implementado en HW de forma que decodifica las instrucciones de la arquitectura antigua y las traduce a la nueva arquitectura.

Un ejemplo de emulador de hardware son los microprocesadores Java, que emulan la máquina abstracta JVM (Java Virtual Machine).

Un **traductor entre códigos binarios** o **ensambladores** son conjuntos de instrucciones de la nueva arquitectura que reproducen el comportamiento de un programa en la arquitectura antigua (la información de la máquina antigua se almacena en registros de la nueva máquina).

Un ejemplo de estos traductores son los desarrollados por DEC (Digital Equipment Corporation) para traducir instrucciones de la arquitectura VAX y MIPS a la nueva arquitectura ALPHA.

Un **compilador nativo** es aquél que toma un programa fuente antiguo y lo vuelve a compilar (recompilar) con compiladores desarrollados para la nueva arquitectura. Son los más rápidos y óptimos con respecto a las demás técnicas, pues produce la mejor calidad del código objeto.

### III. REQUERIMIENTOS:

#### 1. Materiales y equipo a utilizar:

DESCRIPCIÓN	CANTIDAD
Guía de laboratorio	1
Editor de texto de Visual C++, QBasic, Bloc de notas	1
Compilador de Visual C++, QBASIC, JDK	1
Disco Flexible (1.44 Mb)	2

#### 2. Estudiar la guía, para evaluación al iniciar el laboratorio.

### IV. PROCEDIMIENTO

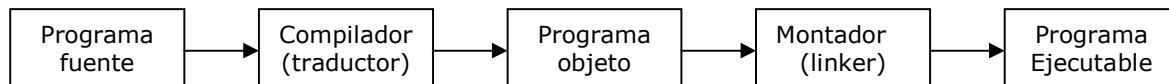
#### PARTE I. (Programas Compilados – Lenguaje C y C++)

Los lenguajes de programación compilados (C, C++, FORTRAN, etc.), necesitan pasar por una serie de pasos que son definidos por su compilador.

La compilación (como se menciono anteriormente), es el proceso de traducción de programas fuente a programas objeto. El programa objeto obtenido de la compilación ha sido traducido normalmente a código máquina.

Para conseguir el programa máquina real se debe utilizar un programa llamado *montador* o *enlazador* (*linker*). El proceso de montaje conduce a un programa en lenguaje máquina ejecutable.

A continuación se presenta una figura con los pasos necesarios para ejecutar un programa escrito en un lenguaje de programación y compilado (mediante un compilador):



### EJERCICIO 1

Para analizar como es el proceso de ejecución de un programa en C, se describe a continuación los pasos necesarios para compilar y ejecutar un pequeño código de programa en C:

- a) La primera operación es introducir las instrucciones (sentencias) del programa, con un editor de texto.
- Crear una carpeta, en el raíz del c (Digitar su nombre)
  - Crear un proyecto en Visual C++, dentro de la carpeta de trabajo (la que creo con su nombre)
  - Utilizar un editor C, y digite el siguiente código

```
//Este programa contiene un error para que sea detectado por el compilador
#include <stdio.h> //libreria de C
#include <conio.h>
int main(){
    int n; //declaracion de variable
    printf("Digite un numero: "); //mensaje en pantalla
    scanf("%d", &x); //lectura del numero ( x no esta declarado)

    printf("%d", n); //se imprime el numero
    getch();
    return 0;
}
```

- Guarde el programa como EjemploC.

b) Etapa de Compilación.

Aquí se traduce el código fuente escrito en lenguaje C a código máquina (entendible por la computadora). Ahora bien, cada compilador se construye para un determinado lenguaje de programación, en este caso, se ejecutara un programa bajo un compilador de C que forma parte del programa **EID** (Entorno Integrado de Desarrollo) de Visual C++.

El compilador de Visual C++ (compatible para lenguaje C), leerá el programa del archivo de texto creado anteriormente y comprueba que siga las reglas de sintaxis del lenguaje de programación. Cuando el programa es compilado, el compilador traduce el código fuente C (las sentencias del programa) en un código máquina (código objeto).

El *código objeto* consta de instrucciones máquina e información de cómo cargar el programa en memoria antes de su ejecución, normalmente se almacena en disco con extensión **.obj** o **.o**

- Realice lo siguiente:
  - Compilar el archivo editado, en el compilador de Visual C++ (c, o cpp)
  - Note que se muestra un mensaje indicando que el compilador ha detectado un error:  
'x' : undeclared identifier

- Corregir el error, y cambie la letra 'x' por 'n'. (`scanf("%d", &n)`).
- Como ya no se detectaron errores, en este momento ya esta creado el archivo objeto en la carpeta Debug del proyecto, (ejemploc.obj)
- Para crear el archivo ejecutable (ejemploc.exe), ejecute el archivo ya compilado ejemploc, el resultado se verificará en las carpeta Debug del proyecto.

El archivo objeto contiene solamente la traducción del código fuente. Esto no es suficiente para ejecutar el programa. Es necesario incluir los archivos de biblioteca (p.e. **stdio.h**). Una biblioteca es una colección de código que ha sido programada y traducida para ser utilizada en el programa.

c) Etapa de montaje (Linkage).

Normalmente un programa consta de diferentes unidades o partes de programa que se han compilado independientemente. Por consiguiente puede haber varios archivos objeto. Un programa especial llamado **enlazador (linker)**, toma el archivo objeto y las partes necesarias de la biblioteca del sistema y construye un **archivo ejecutable (.exe)**.

d) Etapa de ejecución

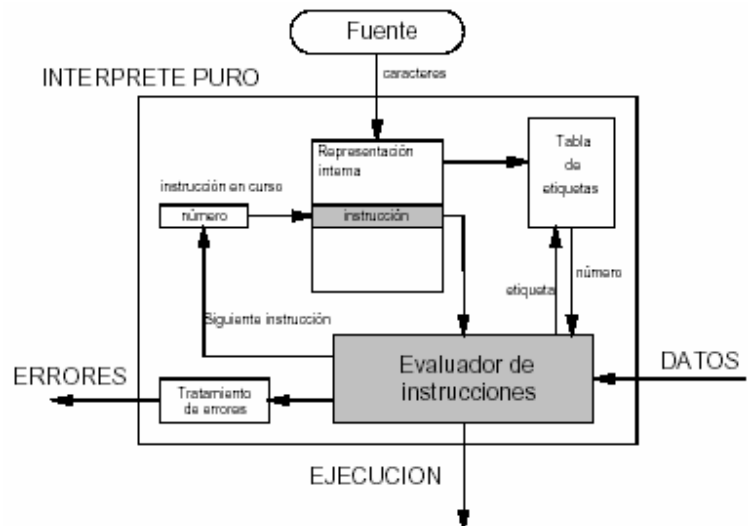
Finalmente el programa es ejecutado (\*.exe).

- Para ejecutar el programa digite el nombre del archivo ejecutable (ejemploc.exe), desde la ventana de comandos (pero debe ubicarse en la carpeta donde esta almacenado el archivo .exe).

**ejemploC.exe**

- Compruebe la ejecución satisfactoria del programa.

**Nota:** Cabe señalar que el proceso de compilar y enlazar un programa, se repite hasta que no se produzcan errores.



## PARTE II. (Programas Interpretados - Qbasic)

Un intérprete (como se menciono anteriormente) es un traductor que toma un programa fuente, lo traduce y a continuación lo ejecuta. Los programas intérpretes clásicos como BASIC, prácticamente ya no se utilizan, aunque versiones como Qbasic y QuickBASIC todavía se encuentran en algunos sistemas operativos.

Los intérpretes analizan una sentencia y la ejecutan (línea por línea), y así sucesivamente para todo el programa fuente. El principal problema de un intérprete es que si a mitad del programa fuente se producen errores, se debe volver a comenzar el proceso de análisis y ejecución del programa.

## EJERCICIO 2

Para analizar como es el proceso de ejecución de un programa interpretado, realice los siguientes pasos para probar y ejecutar un pequeño código de programa escrito en Qbasic:

- Desde la línea de comando digite lo siguiente: **qbasic** (*qbasic*).
- Una vez cargada la aplicación escriba el siguiente código en el editor de Qbasic:

```
'Programa en qbasic  
  
DIM n AS INTEGER 'declaracion de variable  
  
INPUT "Digite un numero: ", x 'lectura de una variable no declarada 'x'  
  
PRINT "El numero digitado fue: ", n  
  
END
```

- Guarde el programa como ejemplo **B.bas** en la carpeta de trabajo (la que ud. tiene con su nombre).
- Para ejecutar el programa presione **F5**.
- Durante la ejecución del programa pruebe las siguientes entradas:
  1. Cuando el programa muestra el mensaje de “**Digite un número:** ” en lugar de un número digite una letra y observara como Qbasic genera un mensaje de error y vuelve a interpretar esa línea.
  2. Como usted habrá notado al intérprete no le interesa si **x** está declarado, esta es una característica propia del lenguaje. Obviamente es en **x** que se guarda el valor digitado y no en **n**.
- Salga de **qbasic** y describa a continuación lo que entiende por un compilador y un intérprete:\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_.

### PARTE III. (Programas Compilados e Interpretados – Java)

Antes de probar la ejecución de un programa en JAVA, se mencionan a continuación algunas de sus características:

- Es simple: Los programadores en Java dejan fuera muchas de las capacidades innecesarias de otros lenguajes de alto nivel. Por ejemplo, Java no soporta matemáticas de puntero, casting implícito de tipos de datos, estructuras, uniones, operadores cargados (overloading), plantillas, archivos de cabecera, o herencia múltiple.
- Orientado a objetos: Así como C++, Java usa clases para organizar el código en módulos lógicos. En tiempo de ejecución un programa crea objetos a partir de las clases. Dichas clases pueden heredar propiedades de otras clases.
- Tipado estático: Todos los objetos utilizados en un programa deben ser declarados antes de ser usados. Esto permite que el compilador de Java localice y reporte los conflictos de tipos.
- Compilado e Interpretado: Antes de ser ejecutado, un programa Java debe ser compilado, lo que resulta en un archivo objeto (byte-code), similar a lenguaje máquina, el cual puede ser ejecutado por cualquier sistema operativo que posea un interprete de Java. Esto hace pensar que Java es un lenguaje tanto compilado como interpretado.

### ▪ Por qué Java es diferente a otros lenguajes de programación

Un programa escrito en lenguaje Java (extensión **.java**), se compila primero para traducir el programa fuente en un lenguaje intermediario llamado Java *bytecodes*, es decir, genera un archivo con formato binario denominado *bytecode* (extensión **.class**). Este archivo es independiente de la plataforma y es interpretado posteriormente en cualquier sistema operativo que disponga de un intérprete de *bytecode* (normalmente un navegador de Internet).

El intérprete analiza y ejecuta cada instrucción *bytecode* de Java en la computadora. La compilación solo ocurre una vez, la interpretación ocurre cada vez que el programa es ejecutado. La siguiente figura ilustra lo descrito anteriormente:

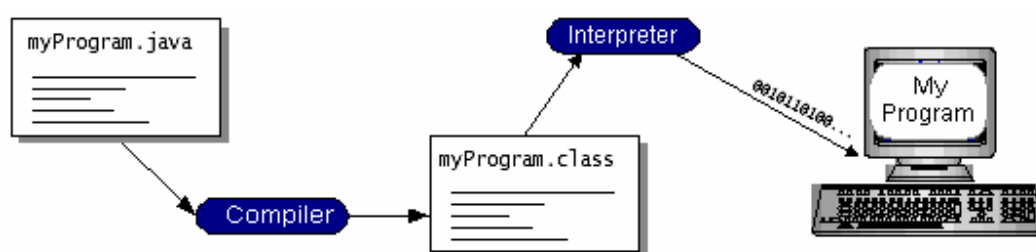


Fig. 9

Se puede pensar que los *bytecodes* Java son como las instrucciones en lenguaje máquina para la máquina virtual de Java (JVM). Cada intérprete de Java, ya sea una herramienta de desarrollo o un navegador Web que pueden correr *applets*, es una implementación de máquina virtual de Java. Por lo tanto, un programa en JAVA puede correr en Windows 2000, Solaris o Macintosh siempre y cuando se tenga una implementación de Java VM(máquina virtual).

### **EJERCICIO 3** (Creación e Inserción de un applet en una página Web)

Para analizar como es el proceso de ejecución de un programa en Java, se utilizarán **applets**. Los **applets** son similares a las aplicaciones, pero ellos no corren de forma independiente. En su lugar, utilizan un navegador o browser (con Java habilitado o compatible) para su ejecución.

Antes de crear el applet, deberá verificar si existe en la configuración de una variable de entorno "path", de modo que contenga el directorio de binarios de Java (JDK):

- Primero verifique que la variable de entorno **PATH** este establecida para su sistema. Para hacer esto, en la ventana de "DOS" abierta escriba: **path**. Se le mostrarán los directorios que han sido exportados como parte de PATH.
- Si no esta configurado el directorio, escriba la orden siguiente: **path=%path%;ESCRIBIR LA RUTA DE LA CARPETA**
- Nuevamente verifique el estado de la variable de entorno PATH. Podrá ver como el directorio donde se encuentra la aplicación ha sido añadida al PATH. De esta manera podrá ejecutar el compilador de Java (**javac**) estando en cualquier directorio de la unidad C.
- Cámbiese al directorio de trabajo (el que tiene su nombre, ahí debe guardar los archivos .java).



## Creación del Applet (Despliegue de texto)

Una de las operaciones más sencillas de entrada/salida en una applet es desplegar una línea de texto. Pero debido a que la salida de texto debe ser gráfica, es necesario utilizar las funciones de texto gráfico de Java. De estas la más comúnmente usada es **drawString()**, la cual forma parte de la clase **Graphics**, contenida en el paquete **awt**.

**Nota:** Un **paquete** no es más que una colección de clases relacionadas. El paquete **awt** (abstract windows toolkit) contiene todas las clases que manejan gráficos y ventanas gráficas

- Realice los siguientes pasos:
  - Desde la línea de comando digite **notepad**.
  - A continuación digite el siguiente código de un applet sencillo que despliega una línea de texto:

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

public class Applet1 extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Primer programa de Java!", 60, 75);
    }
}
```

- Guarde el código mostrado en un archivo llamado **Applet1.java** dentro de la carpeta de trabajo.

## Como compilar el applet

La compilación usando JDK (Java Development Kit) es sumamente sencilla. Solamente debe seguirse la misma sintaxis propuesta en la línea de comando siguiente (por supuesto, debe estar ubicado en el directorio de trabajo):

**javac Applet1.java**

A esta línea solamente debe cambiarse el nombre del código fuente para compilar otros applets. Cuando se realiza la compilación se crea un archivo *bytecode* (.class), que en este caso se llama **Applet1.class**, el cual deberá ser incrustado en una página web para ser ejecutado. Si llegase a tener errores el código fuente, estos son indicados, pero no es generado el archivo con extensión .class, por tanto debe corregirlos y luego volver a compilarlo.

**Nota:** Tenga en cuenta que el compilador de Java es "case-sensitive", así que si en la línea de código anterior, el nombre del applet se ha escrito iniciando con mayúsculas (tal como en el ejemplo), el nombre del archivo Applet1.java obligatoriamente debe escribirse también con mayúsculas.

## Inserción del applet en una página Web (Intérprete)

La sintaxis general para la inserción de un applet es la siguiente:

```
<applet y atributos>
  parametros
  contenido alternativo
</applet>
```

A continuación se muestra todo el proceso a seguir para insertar un applet en una página de modo que pueda ser mostrado en una ventana del navegador:

- Cree una nueva página Web. Para esto, abra el programa **Notepad** (bloc de notas).
- Digite lo siguiente:

```
<html>
<head>
  <title>Prueba de Applets Java</title>
</head>
<body>
  <center><h1>Pagina de Prueba</h1></center>
  <center>
    <applet
      code="Applet1.class"
      width=200
      height=150
      name="Applet1">
    </applet>
  </center>
</body>
</html>
```

- Guarde el archivo como **ejemplo1.html**.
- Abra el archivo prueba.html usando **Internet Explorer**. En este momento podrá ver como el navegador interpreta y ejecuta el applet.

Para el caso del Applet (Applet1.class), es necesario explicar su funcionamiento general:

- Primero veamos el método **paint()**. Java hace una llamada a **paint()** siempre que el applet aparece por primera vez en la pantalla.
- **paint()** llama a **drawString()**, que es el método que en realidad despliega el texto del ejemplo.
- La **g** seguida del **punto**, indica que se esta llamando a un método que en realidad pertenece a la clase **Graphics**, y por tanto el objeto **g** lo hereda.
- Los valores entre paréntesis de **drawString** son los argumentos necesarios para llamar al método. En este caso son el texto a desplegar y la columna (60) y la fila (75) de despliegue. La posición es medida en píxeles y no en caracteres.

### Conclusión

Cuando un navegador (con Java habilitado) encuentra una etiqueta **<APPLET>**, este reserva un área de trabajo con un tamaño específico para el applet, carga los bytecodes generados y crea una instancia de la subclase Applet.

### **EJERCICIO 4** (Ejecución de un applet sin un navegador)

Aunque hoy en día prácticamente todos los navegadores han sido habilitados para dar soporte a applets de Java, todavía podemos ver la ejecución de un Applet sin necesidad de un navegador. Esto gracias a la herramienta **Appletviewer** provista como parte del JDK (Java Development Kit). Esta herramienta es una aplicación de Windows, que puede ser ejecutada desde la línea de comando de DOS.

Para ejecutar el applet creado anteriormente usando el Appletviewer, digite lo siguiente:  
**appletviewer ejemplo1.html**

#### **EJERCICIO 5** (Obteniendo entradas del usuario)

En este caso, debido a los despliegues eminentemente gráficos que se obtienen en la ejecución de un applet, no es tan sencillo llamar a una función o comando "input". Primero debe crearse un área en la pantalla en la cual el usuario pueda digitar y editar su entrada. Existen diversas maneras de hacerlo, pero una de las más prácticas es añadir un control de clase **TextField** al applet. Digite el código presentado a continuación y guárdelo como **Applet2.java**.

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

public class Applet2 extends Applet
{
    TextField textField; /* Caja de texto */
    public void init()
    {
        textField = new TextField(20);
        add(textField);
    }
}
```

Explicación del Applet (Applet2.class):

- Primero declaramos un campo de datos llamado **textField**, como un objeto de la clase **TextField** (tenga en cuenta mayúsculas y minúsculas), el cual representa al control que será muy parecido a una caja de texto de Windows.
- En este caso se utiliza el método **init()**. Este método también se llama de manera automática, tan pronto se inicia la ejecución de Applet2. Básicamente realiza el trabajo de inicialización del entorno del applet.
- Cuando **init()** es llamado, **textField** ya ha sido declarado como un objeto de la clase mencionada, pero aún no se le ha asignado ningún valor. Esto se logra con la línea **textField = new TextField(20);** El valor de 20 es el ancho del control.
- El siguiente paso es añadir el objeto al área de despliegue del applet, lo cual se logra con la línea **add(textField);** El método **add()** recibe un argumento simple que es el nombre del control que será añadido al applet.

#### **Ejercicio adicional:**

Compile Applet2.java, de modo que obtenga Applet2.class. Luego inclúyalo en una página Web y verifique el resultado de la ejecución del Applet.

**Recomendación:**

Siempre que haga cambios en un applet y lo recompile, será necesario cerrar la ventana del navegador (ya sea Netscape o Internet Explorer), para que el archivo de clase especificado en la página vuelva a ser leído. De lo contrario no podrá ver el resultado de dichos cambios.

**Programa 6** (Recuperación de texto desde controles)

- Digite el siguiente código y guárdelo como **Applet3.java**.

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

public class Applet3 extends Applet
{
    TextField textField;
    public void init()
    {
        textField = new TextField(20);
        add(textField);
    }
    public void paint(Graphics g)
    {
        String s = textField.getText();
        g.drawString(s, 40, 50);
    }
    public boolean action(Event event, Object arg)
    {
        repaint();
        return true;
    }
}
```

Explicación del Applet (Applet3.class):

- En este applet se utiliza la función **action** que retorna un valor de tipo **boolean**. Lo novedoso es que se incluye un argumento de tipo **Event**. Esto significa que el applet esta esperando que se ejecute una acción del usuario. (Action listeners). Cuando el usuario hace clic en un botón, se presiona la tecla enter o se da un retorno de carro (return) en un **text field**, se considera que ha ocurrido una acción (evento de usuario). Como resultado de la acción, se envía un mensaje a la función **action** por medio del parámetro Event y ésta se ejecuta.
- Al ocurrir un evento del usuario (action event), dentro de la función **action** se llama al método **repaint()**. Este método llama lo antes posible al método **update()**. El método *update()* hace dos cosas: primero re-dibuja la ventana del applet con el color de fondo y luego llama al método **paint()**.

## V. INVESTIGACIÓN Y EJERCICIOS COMPLEMENTARIOS

Realizar los siguientes ejercicios:

Investigue en Internet los siguientes tópicos:

- ¿Existen diferencias entre la inclusión de un applet de java en una página HTML, y la escritura de código JavaScript directamente en el código de la página?
- ¿Qué son los Java Beans?
- ¿En que consiste el lenguaje de programación XML y cual es la diferencia entre HTML y XML?
- En qué tipo de procesadores de lenguajes se ubican HTML, XML, ASP, Perl y Pascal, Visual Basic, Fortran.
- Qué es considerado Token en un compilador.
- Los Token son considerados estándar en los lenguajes de programación. Explique.
- Ejemplos de Tokens según estos lenguajes C, Java y Pascal.

## VI. FUENTES DE CONSULTA

- ☒ Fundamentos de Compiladores. Cómo Traducir al Lenguaje de Computadora. Karen A. Lemone. Editorial Continental.
- ☒ Pratt W Terrence. Lenguajes de Programación, Tercera edición, Editorial Prentice may.
- ☒ Lenguaje de programación, Paradigma y práctica. Mc Grawhill.