GUIA DE JAVA NIVEL BÁSICO

INTRODUCCIÓN

Java se creó como parte de un proyecto de investigación para el desarrollo de software avanzado para una amplia variedad de dispositivos de red y sistemas embebidos. La meta era diseñar una plataforma operativa sencilla, fiable, portable, distribuida y de tiempo real. Cuando se inició el proyecto, C++ era el lenguaje del momento. Pero a lo largo del tiempo, las dificultades encontradas con C++ crecieron hasta el punto en que se pensó que los problemas podrían resolverse mejor creando una plataforma de lenguaje completamente nueva. Se extrajeron decisiones de diseño y arquitectura de una amplia variedad de lenguajes como Eiffel, SmallTalk, Objetive C y Cedar/Mesa. El resultado es un lenguaje que se ha mostrado ideal para desarrollar aplicaciones de usuario final seguras, distribuidas y basadas en red en un amplio rango de entornos desde los dispositivos de red embebidos hasta los sistemas de sobremesa e Internet.

OBJETIVOS DE DISEÑO DE JAVA

Sencillo, Orientado a Objetos y Familiar

Robusto y Seguro

Independiente de la Arquitectura y Portable

Alto Rendimiento

Interpretado, Multi-Hilo y Dinámico

CARACTERÍSTICAS

- Lenguaje de Propósito General.
- Lenguaje Orientado a Objetos.
- Sintaxis inspirada en la de C/C++.
- Lenguaje Multiplataforma: Los Programas Java se Ejecutan sin Variación (sin recompilar) en Cualquier Plataforma Soportada (Windows, UNIX, Mac...).
- Lenguaje Interpretado: El Intérprete a Código Máquina (dependiente de la plataforma) se llama Java Virtual Machine (JVM). El Compilador Produce un Código Intermedio Independiente del Sistema Denominado bytecode.
- Lenguaje Gratuito: Creado por SUN Microsystems, que Distribuye Gratuitamente el Producto Base, Denominado JDK (Java Development Toolkit) o Actualmente J2SE (Java 2 Standard Edition).
- API Distribuida con el J2SE muy Amplia. Código Fuente de la API Disponible.

Los programas más comunes de Java son las **Standalone Programs** (Aplicaciones) y los **Applets**. Las **Standalone Programs** son Aplicaciones que se pueden ejecutar como cualquier otro lenguaje de programación, sin necesidad de ningún elemento soporte. Los **Applets** son pequeñas Aplicaciones en Java, que se transfiere a través de la red y que necesita para su ejecución un browser (Navegador o Visualizador) de red compatible con Java.

“Escribe Una Vez y Ejecuta en Cualquier Plataforma”

El éxito imparable y sin precedentes del lenguaje de programación Java desde sus inicios en el año 1996 se debe en gran parte a que este lenguaje es independiente de cualquier **Plataforma**. Es decir, cualquier aplicación Java se puede desarrollar en la plataforma que nosotros como desarrolladores nos sintamos más cómodos y se podrá ejecutar o explotar en la plataforma que nuestros clientes elijan.

- **Plataforma de Desarrollo (Windows, Linux, Mac OS X, o cualquier otro)**
  - Primero que todo necesitamos saber cuál es la arquitectura Hardware de nuestro ordenador y que Sistema Operativo utilizamos para poder instalar la **MVJ** (Máquina Virtual Java) que corresponda a nuestra Plataforma.

- **IDE Entorno de Desarrollo (NetBeans o cualquier otro)**
  - En el mercado hay muchos entornos de desarrollo disponibles y de muy alta calidad
    - en este curso utilizaremos el IDE **NetBeans** que es un entorno de desarrollo Open Source escrito en Java y auspiciado por **Sun Microsystems**

- **MVJ (Máquina virtual Java)**
  - La MVJ no existe como tal físicamente. Es en realidad un conjunto de comandos y programas que se ejecutan en un direccionamiento de memoria separado del resto de procesos que se puedan estar ejecutando en paralelo en ese momento en nuestro ordenador.
  - **Código Java**
    - El código fuente de las Aplicaciones Java que creamos está separado por diferentes ficheros con extensión **.java**.
      - Las Clases están contenidas en los ficheros con extensión **.java**
        - Cada Clase tiene su propio fichero.
          - Ya hablaremos largo y tendido sobre las Clases durante todo el curso.
  - **Compilación**
    - Al poco tiempo de aparecer en escena el lenguaje Java, las Aplicaciones se escribían en un procesador de texto y la compilación del código fuente se realizaba desde una consola de comandos propia del Sistema Operativo que se estaba utilizando.
      - Entonces tenías que escribir en la línea de comando el comando **javac** seguido de las Clases que querías compilar.
    - Debido a que el IDE NetBeans está situado en una capa por encima de la MVJ, no nos hace falta utilizar una consola de comandos para poder compilar las Clases Java.
      - En este caso, nosotros simplemente tenemos que hacer clic sobre un botón de compilación, que ya veremos más adelante, y esta acción desencadena que el IDE invoque, de forma transparente para nosotros, el comando **javac** que está incluido en la MVJ para compilar dichas Clases.
  - **Código byte**
    - El resultado de la compilación genera ficheros con extensión **.class**
      - Estos ficheros contienen **código byte**
        - Este código byte es universal.
          - Es decir, no importa con que plataforma estemos desarrollando que el código de bajo nivel generado es 100% compatible con la MVJ
  - **Interpretación**
    - A diferencia de otros lenguajes que sus Aplicaciones se arrancan directamente ejecutando un fichero con extensión normalmente **.exe**, para ejecutar una Aplicación Java desde la línea de comando se tiene que ejecutar primeramente el comando **java.**
      - Este comando crea un entorno de ejecución Java en memoria y transforma el código byte de los ficheros con extensión **.class** en código de bajo nivel propio de cada plataforma

        - En este caso el IDE NetBeans también tiene un botón, que ya veremos en su momento, para interpretar dicho código byte y ejecutar la Aplicación para que el usuario final la pueda utilizar.
    - **Librerías estándar Java**
        - Sería impensable que todo el código de una Aplicación estuviera escrito por nosotros.
            - Como cualquier otro lenguaje, Java tiene una nutrida **API** (Application Program Interface) que contiene las librerías estándar Java.
        - En el proceso de interpretación, el interpretador Java invoca las Clases de la mencionada librería estándar Java y las interpreta.
    - Las Aplicaciones java desde sus principios siempre se han ganado la mala fama de ser más lentas que otras aplicaciones convencionales. Pero hay que decir que el tiempo corre a favor de Java debido a que los ordenadores cada vez tienen frecuencias de reloj más rápidas y capacidades de memoria cada vez más grandes.
        - **Tampoco se descarta que en un futuro la MVJ esté implementada directamente en un procesador.**

Cuando tenemos que instalar una aplicación creada por nosotros mismos en casa de un cliente, éste tiene que tener instalado en su ordenador de producción la MVJ correspondiente a su Plataforma.

- La MVJ se puede descargar de Internet como un **JDK** (Java Development Kit) o un **JRE** (Java Runtime Environment).
    - **JRE**
        - En el caso que sólo queramos ejecutar las Aplicaciones en el ordenador de producción.
    - **JDK**
        - En el caso de que además de ejecutar las Aplicaciones en el ordenador de producción, también queramos compilar en el citado ordenador de producción.
        - Cuando se instala un JDK, el Wizard o Asistente también instala automáticamente su correspondiente JRE.

Donde Bajar el JDK (Java Develompment Kit**) y el JRE (**Java Runtime Environment**)**

En la Dirección WEB http://www.java.sun.com en el área Download podrá encontrar la opción de descarga de su preferencia.

DEFINICIÓN DE TIPOS

Java es un lenguaje con control fuerte de Tipos (Strongly Typed). Esto significa que cada Variable y cada expresión tienen un Tipo que es conocido en el momento de la compilación. El Tipo limita los valores que una variable puede contener, limita las operaciones soportadas sobre esos valores y determina el significado de las operaciones. El control fuerte de tipos ayuda a detectar errores en tiempo de compilación.

En Java existen dos categorías de Tipos:

- Tipos Primitivos
- Referencias

DEFINICIÓN DE TIPOS PRIMITIVOS

Los Tipos Primitivos son los que permiten manipular valores numéricos (con distintos grados de precisión), caracteres y valores booleanos (verdadero / falso). Los Tipos Primitivos son:

- **boolean:** Puede contener los Valores True o False.
- **byte:** Enteros. Tamaño 8-bits. Valores entre -128 y 127.
- **short:** Enteros. Tamaño 16-bits. Entre -32768 y 32767.
- **int:** Enteros. Tamaño 32-bits. Entre -2147483648 y 2147483647.

- **long:** Enteros. Tamaño 64-bits. Entre -9223372036854775808 y 9223372036854775807.
- **float:** Números en Coma Flotante. Tamaño 32-bits.
- **double:** Números en Coma Flotante. Tamaño 64-bits.
- **char:** Caracteres. Tamaño 16-bits. Unicode. Desde '\u0000' a '\uffff' Inclusive. Esto es desde 0 a 65535

El tamaño de los tipos de datos no depende de la implementación de Java. Son siempre los mismos.

DEFINICIÓN DE REFERENCIAS

Las Referencias en Java no son punteros ni Referencias como en C++. Este hecho crea un poco de confusión entre los programadores que llegan por primera vez a Java. Las Referencias en Java son identificadoras de instancias de las clases Java. Una Referencia dirige la atención a un objeto de un tipo específico.

DEFINICIÓN DE VARIABLES

Una variable es un área en memoria que tiene un nombre y un Tipo asociado. El Tipo es o bien un Tipo Primitivo o una Referencia. Es obligatorio declarar las Variables antes de usarlas. Para declararlas se indica su nombre y su Tipo, de la siguiente forma:

Ejemplo:

tipo_variable nombre **;**

```
int i;        // Declaración de un entero
char letra;   // Declaración de un carácter
boolean flag; // Declaración de un booleano
```

Se pueden asignar Valores a las Variables mediante la instrucción de asignación.

Ejemplo:

```
i = 5;         // a la Variable i se le asigna el Valor 5
letra = 'c';   // a la Variable letra se le asigna el Valor 'c'
flag = false;  // a la Variable flag se le asigna el Valor false
```

La Declaración y la Combinación se pueden combinar en una sola expresión:

Ejemplo:

```
int i = 5;
char letra = 'c';
boolean flag = false;
```

DECLARACIÓN DE MÉTODOS

En Java toda la lógica de programación (Algoritmos) está agrupada en Funciones o Métodos.

Un Método es:

- Un Bloque de Código que tiene un Nombre,
- Recibe unos Parámetros o Argumentos (opcionalmente).
- Contiene Sentencias o Instrucciones para realizar algo (opcionalmente).
- Devuelve un Valor de algún Tipo conocido (opcionalmente).

Ejemplo:

```
Tipo_Valor_devuelto  nombre_método (lista_argumentos) {
      bloque_de_codigo;
}
```

Ejemplo:

```
int sumaEnteros (int a, int b) {
      int c = a + b;
      return c;
}
```

- El Método se llama **sumaEnteros**.
- Recibe dos Parámetros también **enteros**. Sus **Nombres** son **a** y **b**.
- Devuelve un **entero**.

En el ejemplo la Clausula **return** se usa para finalizar el Método devolviendo el Valor de la Variable **c**.

El Término VOID

El hecho de que un Método devuelva o no un Valor es opcional. En caso de que devuelva un Valor se declara el tipo que devuelve. Pero si no necesita ningún Valor, se declara como tipo del Valor devuelto, la palabra reservada **void.**

Ejemplo:

```
void haceAlgo() {
      . . .
   }
```

USO DE MÉTODOS

Los Métodos se invocan con su nombre, y pasando la lista de argumentos entre paréntesis. El conjunto se usa como si fuera una variable del Tipo devuelto por el Método.

Ejemplo:

```
int x;
x = sumaEnteros(2,3);
```

Aunque el método no reciba ningún Argumento, los paréntesis en la llamada son obligatorios. Por ejemplo para llamar a la Función **haceAlgo**, simplemente se pondría:

Ejemplo:

```
haceAlgo();
```

DEFINICIÓN DE CLASES

Las **Clases** son el mecanismo por el que se pueden crear nuevos **Tipos** en Java. Las **Clases** son el punto central sobre el que giran la mayoría de los conceptos de la Orientación a Objetos.

Una **Clase** es una agrupación de Datos y de Código que actúa sobre esos Datos, a la que se le da un nombre.

Una Clase contiene:

- Datos (se denominan Datos Miembro). Estos pueden ser de **Tipos Primitivos o Referencias**.
- Métodos (se denominan Métodos Miembro).

Ejemplo:

```
modificadores class nombre_clase {
      declaraciones_de_miembros ;
}
```

Ejemplo:

```
class Punto {
    int x;
    int y;
}
```

Ejemplo (Estructura del Standalone Programs)

```
class myfirstjavaprog
{
        public static void main(String args[])
        {
           System.out.println("Hello World!");
        }
}
```

NOTA: DEFINICIÓN DEL ENTORNO DE DESARROLLO NETBEANS IDE 6.5.1

Ejemplo (Código del Applets)

```
package com.chuidiang.ejemplos.applet;

import javax.swing.JApplet;
import javax.swing.JLabel;

/**
 * Ejemplo sencillo de applet
 * @author Gregory Rangel
 *
 */
public class EjemploApplet extends JApplet {
        /**
         * Pone un JLabel con el texto "Applet hola mundo" en el JApplet,
de
         * forma que es lo que se visualizará en el navegador.
         */
        public void init() {
                JLabel etiqueta = new JLabel("Applet hola mundo");
                add(etiqueta);
        }}
```

Ejemplo (Estructura del Applets en HTML)

```
<html>
   <head>
      <title>Ejemplo de Applet</title>
   </head>
   <body>
      <applet code="com.chuidiang.ejemplos.applet.EjemploApplet"
         width="500" height="200">
      Debes tener instalado java
      </applet>
   </body>
</html>
```

DEFINICIÓN DE APPLET

Pequeño Programa basado en internet y escrito en Java, los **Applets** generalmente están embebidos en Páginas WEB y pueden ser ejecutados directamente desde un navegador con soporte para Java.

Para que un Java **Applet** pueda ser ejecutado, el navegador debe contar con una Máquina Virtual Java.

Los Java **Applet** son una forma de Aplicación WEB, con los cuales puede hacerse prácticamente lo mismo que con una Aplicación tradicional. Por ejemplo, con un Java **Applet** puede incorporarse animación WEB a una Página WEB.

NOTA:

Las dos primeras líneas:

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

Son declaraciones para el compilador. Le avisan lo siguiente: hay **Clases**, que vamos a usar en este archivo, que provienen de un Grupo de Clases, que en Java se llama **Paquete**. Se indica que vamos a usar **Clases** del Paquete **java.awt**, y **Clases** del Paquete **java.applet** importar esos Paquetes de **Clases**.

El Paquete **java.awt** debe su nombre al **AWT** (**Abstract Window Toolkit**), un conjunto de **Clases** que nos permite usar métodos gráficos, y componentes, independizándonos de la plataforma. Los objetos **AWT** se pueden usar en Windows, en X/Windows de Linux o en cualquier otro entorno gráfico que tenga implementada una Máquina Virtual de Java. Por ejemplo, vamos a manejar ventanas y botones, y las aplicaciones correrán en las distintas plataformas, sin necesidad de reescribir el código, o de recompilar.

El paquete **java.applet** se encarga de definir una serie de **Clases** adicionales, en especial, la **Clase Applet**, que nos permite generar Aplicaciones del tipo **Applet**, para ejecutar desde una Página HTML.

Si no colocamos las declaraciones "**import**", nuestro ejemplo no compilaría. El compilador no encontraría cuál es, por ejemplo, la **Clase Applet** o la **Clase Graphics**, que son usadas en el código.

ESTRUCTURA DEL AWT

La Estructura “básica” del **AWT** se basa en Componentes y Contenedores. Estos últimos contienen Componentes posicionados a su respecto y son Componentes a su vez, de forma que los eventos pueden tratarse tanto en Contenedores como en Componentes, corriendo por cuenta del programador.

COMPONENTES Y CONTENEDORES

Una interfaz gráfica está construida en base a elementos gráficos “básicos”, los **Componentes**. Típicos ejemplos de estos Componentes son los botones, barras de desplazamiento, etiquetas, listas, cajas de selección o campos de texto. Los Componentes permiten al usuario interactuar con la Aplicación y proporcionar información desde el Programa al usuario sobre el estado del Programa. En el **AWT**, todos los Componentes de la interface de usuario son instancias de la **Clase Component** o uno de sus subtipos.

Los Componentes no se encuentran aislados, sino agrupados dentro de **Contenedores**. Los Contenedores contienen y organizan la situación de los Componentes; además, los Contenedores son en sí mismos Componentes y como tales pueden ser situados dentro de otros Contenedores. También contienen el código necesario para el control de eventos, cambiar la forma del cursor o modificar el icono de la aplicación. En el **AWT**, todos los Contenedores son instancias de la **Clase Container** o uno de sus subtipos.

TIPOS DE COMPONENTES

En el árbol siguiente se muestra la relación que existe entre todas las **Clases** que proporciona **AWT** para la creación de interfaces de usuario, presentando la jerarquía de **Clases** e Interfaces:

Clases:

- Adjustable
- BorderLayout
- CardLayout
- CheckboxGroup
- Color
- Component
    - Button
    - Canvas
    - Checkbox
    - Choice
    - Container
    - Panel
        - Applet
    - ScrollPane
    - Window
        - Dialog
        - FileDialog
        - Frame
    - Label
    - List
    - Scrollbar
    - TextComponent
    - TextArea
    - TextField
- Cursor
- Dimension
- Event
- FlowLayout
- Font
- FontMetrics
- Graphics
- GridLayout
- GridBagConstraints
- GridBagLayout
- Image
- Insets
- MediaTracker
- MenuComponent
    - MenuBar
    - MenuItem
    - CheckboxMenuItem
    - Menu
        - PopMenu
- MenuShorcut
- Point
- Polygon
- PrintJob
- Rectangle
- Toolkit

Interfaces:

- LayoutManager
- LayoutManager2
- MenuContainer
- Shape

PALABRAS RESERVADAS

java.import

Para importar Clases de un paquete se usa el comando **import**. Se puede importar una Clase individualmente.

```
import java.awt.Font;
```

O bien, se puede importar las Clases declaradas públicas de un Paquete completo, utilizando un **asterisco (*)** para reemplazar los nombres de Clase individuales.

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

Para crear un objeto fuente de la Clase Font podemos seguir dos alternativas.

```
import java.awt.Font;
Font fuente=new Font("Monospaced", Font.BOLD, 36);
```

O bien, sin poner la sentencia **import**

```
java.awt.Font fuente=new java.awt.Font("Monospaced", Font.BOLD,
36);
```

Normalmente, usaremos la primera alternativa, ya que es la más económica en código, si tenemos que crear varias fuentes de texto.

Se pueden combinar ambas formas, por ejemplo, en la definición de la clase BarTexto

```
import java.awt.*;
public class BarTexto extends Panel implements java.io.Serializable{
//...
}
```

PAQUETES ESTÁNDARES

Paquete	Descripción
java.applet	Contiene las Clases necesarias para crear **Applets** que se ejecutan en la ventana del navegador.
java.awt	Contiene Clases para crear una Aplicación GUI independiente de la plataforma.
java.io	Entrada/Salida. Clases que definen distintos flujos de Datos.
java.lang	Contiene Clases esenciales, se importa implícitamente sin necesidad de una sentencia import.
java.net	Se usa en combinación con las Clases del paquete **java.io** para leer y escribir Datos en la red.
java.util	Contiene otras Clases útiles que ayudan al programador.

java.util

Contiene el marco de las colecciones, herencia recogida clases, modelo de evento, fecha y hora de las instalaciones, la internacionalización, y diversas Clases de utilidad.

INTERFAZ GRÁFICA EN NETBEANS IDE 6.1

El “Constructor” de interfaces de usuario del IDE NetBeans (conocido anteriormente como el “Proyecto Matisse”) es un módulo del Entorno de Desarrollo Integrado NetBeans.

Este editor de interfaces gráficas está orientado hacia la librería gráfica Swing de Java. Es decir, que únicamente produce código fuente para Java.

En NetBeans 6.1 el generador de interfaces gráficas de usuario se ha hecho más eficiente: ahora es más potente e intuitivo, y permite a los usuarios generar interfaces gráficas de usuario de aspecto profesional sin necesidad de profundizar en el conocimiento de los administradores de diseño.

El nuevo generador de interfaces gráficas de usuario de NetBeans acaba con las dificultades inherentes a la generación de interfaces gráficas de usuario, lo que permite diseñar formularios colocando simplemente los componentes donde desee.

DEFINICIÓN DE JFRAME

El **Frame** es un Contenedor que tiene como tarea guardar nuestros Componentes y darles un sentido gráfico, digamos que el **Frame** es una ventana que tiene propiedades como tamaño, posición, titulo, etc.

DEFINICIÓN DE SWING

El paquete **Swing** es parte de la **JFC** (**Java Foundation Classes**) en la plataforma Java. La **JFC** provee facilidades para ayudar a la gente a construir GUIs. Swing abarca componentes como botones, tablas, marcos, etc.

Las componentes **Swing** se identifican porque pertenecen al paquete **javax.swing**.

Swing existe desde la **JDK 1.1** (como un agregado). Antes de la existencia de **Swing**, las interfaces gráficas con el usuario se realizaban a través de **AWT** (**Abstract Window Toolkit**), de quien **Swing** hereda todo el manejo de eventos. Usualmente, para toda componente **AWT** existe una componente **Swing** que la reemplaza, por ejemplo, la Clase **Button** de **AWT** es reemplazada por la Clase **JButton** de **Swing** (**el nombre de todas las componentes Swing comienza con "J"**).

Las componentes de **Swing** utilizan la infraestructura de **AWT**, incluyendo el modelo de eventos **AWT**, el cual rige cómo una componente reacciona a eventos tales como, eventos de teclado, mouse, etc. Es por esto, que la mayoría de los programas **Swing** necesitan importar dos paquetes **AWT**: **java.awt.*** y **java.awt.event.***.

EL ALUMNO REVISARÁ EL USO DE LAS CLASES MÁS COMUNES EN LA CREACIÓN DE JFRAME AVANZADOS.

Ejemplo:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class HolaMundoSwing {
    public static void main(String[] args) {
        JFrame frame = new JFrame("HolaMundoSwing");
        final JLabel label = new JLabel("Hola Mundo");
        frame.getContentPane().add(label);

        //frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.addWindowListener(new java.awt.event.WindowAdapter(){
           public void windowClosing(WindowEvent e){
             System.exit(0);
           }
        }
        );

        frame.pack();
        frame.setVisible(true);
    }
}
```

CONEXIÓN A BASES DE DATOS CON NETBEANS IDE 6.1

CONEXIÓN A LA BASE DE DATOS

1. Registrar (Cargar en Memoria) el controlador. Esto se puede hacer de dos formas:

De forma dinámica, por medio del método **Class.forName(String de driver)**. Esta es la forma más usual.

Ejemplo:

Ejemplos de carga dinámica:

* En MySQL local: Class.forName("com.mysql.jdbc.Driver");
* En Oracle: Class.forName("oracle.jdbc.Driver.OracleDriver");

De manera estática, usando **System.setProperties("jdbc.drivers",String de driver)**.

2. Establecer la conexión por medio del método **DriverManagerConnection conexion = DriverManager.getConnection("jdbc:mysql://localhost/ java?user=root&password=admin");**

DriverManager tiene muchos Métodos **getConnection()** con Parámetros variados. Todos son variantes de lo mismo y la información que suministramos es la misma. Aquí hemos utilizado uno con tres Parámetros **String**.

Para enviar comandos SQL a la Base de Datos, se usa la Clase **Statement** de java. Esta Clase se obtiene a partir de la conexión, de esta forma:

Statement instruccion = conexion.createStatement();

Por supuesto, dentro de un try-catch.

Statement tiene muchos métodos, pero hay dos interesantes: **executeUpdate()** y **executeQuery()**. El primero se usa para sentencias SQL que implica modificaciones en la base de datos (INSERT, UPDATE, DELETE, etc). El segundo sólo para consultas (SELECT y similares).

EL ALUMNO REVISARÁ EL USO DE LAS CLASES MÁS COMUNES EN LA CREACIÓN DE JFRAME AVANZADOS.

EJERCICIO FINAL DE JAVA COMBINANDO JFRAME Y BASE DE DATOS

Ing. Gregory Rangel