11.- Programación de GUI.

11.1.- Interfases gráficas para el usuario.

Para interactuar con los sistemas de información los usuarios requieren facilidades gráficas que agilicen dicha interacción, mostrar, leer y validar un conjunto de datos es una de las operaciones más comunes. A dichas facilidades se les denomina GUI por sus siglas en inglés *Graphical User Interface* la herramienta inovaProg proporciona dos instrucciones potentes que permiten generar este tipo de interfaces:
Ip.wleeDatos, Ip.wmuestraDatos, Ip.actUnit, Ip.actTab e Ip.MenuBarras las cuales permiten respectivamente leer y validar por configuración un grupo de datos y a su vez mostrar sin permitir modificación alguna un conjunto de datos.

11.2- Leer un grupo de datos.

Cuándo requerimos leer y validar por configuración un grupo de datos podemos utilizar Ip.wleeDatos. Previamente es necesario declarar las siguientes variables:

```
static int numeroCampos  =  6;
//Numero de campos en el conjunto de datos
//Los letreros que aparecerá antes del campo para identificarlo
static String[] etiquetas          =  new String [numeroCampos] ;
//Los tipos de los campos
// "S".-Cadena "I".-Entero,"F".-Flotante y "D".- Fecha"
static String[] tiposCampos        =  new String [numeroCampos] ;
//Este arreglo contiene los valores de los campos que se muestran
// y se capturan.
static String[] valoresCampos      =  new String [numeroCampos] ;
//Es el título que aparece en la ventana
static String titulo          = "Ejemplo de captura";
```

para leer un conjunto de datos por ejemplo supongamos que se requiere leer datos de un usuario-cliente.
Podemos utilizar el siguiente código:

```
int numeroCampos  =  6;
String   etiquetas[]    =  new String [numeroCampos] ;
String[] tiposCampos    =  new String [numeroCampos] ;
String[] valoresCampos  =  new String [numeroCampos] ;
String titulo = "Ejemplo de captura";
//Inicializacion de campos
etiquetas[0]="Usuario";
etiquetas[1]="Contraseña";
etiquetas[2]="Perfil";
etiquetas[3]="Fecha";
etiquetas[4]="Credito";
 etiquetas[5]="Tel.";
 tiposCampos[0]="S";
 tiposCampos[1]="S";
 tiposCampos[2]="I";
 tiposCampos[3]="D";
 tiposCampos[4]="F";
 tiposCampos[5]="S";
```

```
for (int i=0; i<numeroCampos; i++)
      {valoresCampos[i]="";}
valoresCampos[0]="Manuel";
valoresCampos[1]="453beta";
valoresCampos[2]="5";
valoresCampos[3]="12/11/2010";
valoresCampos[4]="43000.00";
valoresCampos[5]="19191919";
Ip.wleeDatos(titulo, etiquetas, valoresCampos, tiposCampos);
```

Este trozo de código despliega la ventana mostrada en la Figura 11.1

Figura 11.1 Lectura de datos utilizando Ip.wleeDatos.

En la ventana obtenida se pueden modificar los datos y al oprimir el botón de aceptar, se lleva a cabo la validación de configuración, según el tipo del campo, si alguna configuración falla, se obtiene un mensaje de error y el cursor se ubica en el campo responsable del error. El botón de Limpiar pone en blcanco todas las cajas de texto mostradas, el botón Restaurar vuelve a escribir los campos que el usuario envió al hacer la llamada a Ip.wleeDatos. Finalmente el botón de Cancelar regresa al programa llamador sin hacerle cambios a los campos. El programa adjunto 1102LeeyMuestraDatos.java implementa el código mostrado e ilustra su ejecución. Una observación es que los valores iniciales o por defecto que se envían a Ip.wleeDatos, son opcionales y si se decide no enviar ningún valor los campos deben de estar inicializados con una cadena de caracteres vacía eso es “”.

El procedimiento Ip.leeDatos està sobrecargado, o sea que permite diversos tipos de llamadas, por ejemplo se puede llamar de las siguientes formas:

Si solo se desean leer los primeros tres campos se puede llamar de la siguiente forma:
numeroCampos=3;

Ip.wleeCampos(titulo, etiquetas, valoresCampos, tiposCampos, numeroCampos);

O bien si se desean leer todos los campos y todos son cadenas de caracteres:
Ip.wleeCampos(etiquetas, valoresCampos);

Cuándo no se desea utilizar etiquetas y todos los campos son cadenas de caracteres se puede llamar de la siguiente manera:

Ip.wleeCampos(valoresCampos);
El programa adjunto 1101LeeYMuestraDatosSimple.java ejemplifica el uso de este tipo de llamadas y nos proporciona ejemplos de las ventanas obtenidas en cada caso.

11.3- Mostrar un grupo de datos.

Para mostrar grupos de campos sin permitir que dichos campos se modifique se puede utilizar una llamada a Ip.wmuestraDatos, es muy similar a la Ip.wmuestraDatos con la diferencia que los campos no permiten modificaciones en la ventana que se muestran.

11.4- Una GUI completa.

Finalmente se pueden utilizar las dos facilidades anteriores para programar un GUI estándar para almacenar datos de distintas entidades, así tenemos el ejemplo que describimos a continuación y que ilustra cómo se puede programar utilizando solo el paradigma de programación por procedimientos.
El ejemplo adjunto se encuentra en el programa 1104Gui.java que se forma de cuatro partes principales:
Declaraciones e inicialización de variables
Menu
Rutinas de Servicio y de entrada/salida a archivos.
Programa Principal

El código de las rutinas de declaración e inicialización de variables se muestra a continuación, éste código es el que varía de un contexto a otro, esto es de un grupo de datos a otro, ya que el resto del programa es independiente del contexto.

```
import javax.swing.*;
import inovaProg.*;
import java.io.*;
class Gui{
  //Declaraciones
 static String tabla="Usuario";
 static int numeroCampos  =  6;
 static int numCamposLlave=0;
 static String LLave ="";
 static String extension ="txt";
 static String preFijo =tabla + "_";
 static String directorioDeDatos =tabla + "\\";
```

```
 static String[]   etiquetas    =  new String [numeroCampos] ;
 static String[] tiposCampos    =  new String [numeroCampos] ;
 static String[] valoresCampos  =  new String [numeroCampos] ;
 static boolean[]  esLLave      =  new boolean [numeroCampos] ;
 static String titulo = "Ejemplo de captura";

 public static void inicializaCampos() {
 //Inicializacion de campos
 //Descripciones para cada una de las cajas de texto de los campos
        etiquetas[0]="Usuario";
        etiquetas[1]="Contraseña";
        etiquetas[2]="Perfil";
        etiquetas[3]="Fecha";
        etiquetas[4]="Credito";
        etiquetas[5]="Tel.";
 //Tipos de campos
 //  "S".- Cadena texto, "I".- Entero, "F".- Flotante, "D".- Fecha
        tiposCampos[0]="S";
        tiposCampos[1]="S";
        tiposCampos[2]="I";
        tiposCampos[3]="D";
        tiposCampos[4]="F";
        tiposCampos[5]="S";
 //Incia con valores por defecto
        for (int i=0; i<numeroCampos; i++)
               {valoresCampos[i]="";
                esLLave[i]=false;
               }
 //Valores de cada campo, aparecen por defecto
        valoresCampos[0]="Manuel";
        valoresCampos[1]="453beta";
        valoresCampos[2]="5";
        valoresCampos[3]="12/11/2010";
        valoresCampos[4]="43000.00";
        valoresCampos[5]="19191919";
//Marcar campos llave deben de estar al inicio del registro
// uno tras otro
        esLLave[0]=true;
//Cuenta el numero de campos LLave
        for (int i=0; i<numeroCampos; i++)
               {if (esLLave[i])
                      numCamposLlave++;
               }
//Tomar la apariencia de windows
        apariencia DeWindows();
        generaDirectorioDeDatos(tabla);
}
```

El siguiente bloque es el menú que no requiere cambiarse de una aplicación a otra:

```
public static void menuAplicacion() {
      int opcion=0;
      String menu   ="Menu de la aplicacion de " + tabla +
                        "\n1.-Alta Registro  " +
                        "\n2.-Consulta Registro  " +
                        "\n3.-Actualiza Registro  " +
                        "\n4.-Borra Registro   " +
                        "\n5.-Selecciona De Registros   " +
                        "\n9.-Termina   \n" ;
      opcion=Ip.wlee(menu,opcion);
```

```
      do{ switch (opcion)
             { case 1:
                  altaRegistro();
                  break;
               case 2:
                  consultaRegistro();
                  break;
               case 3:
                  actualizaRegistro();
                  break;
               case 4:
                  borraRegistro();
                  break;
               case 5:
                  seleccionaRegistro();
                  break;
                  case 9:
                  System.exit(0);
               default:
                  Ip.wescribeMensaje("Opcion "+ opcion + "Invalida");

              }
             opcion=Ip.wlee(menu,opcion);
          } while (opcion!=9);
   }
```

El bloque de Rutinas de Servicio y de entrada/salida a archivos también es independiente del contexto y se encarga de llevar a cabo todas las operaciones del programa:

```
   public static void altaRegistro() {
        Ip.wescribeMensaje("Alta de Registro");
        Ip.wleeDatos("Datos para " + tabla,
                        etiquetas,
                        valoresCampos,
                        tiposCampos,
                        numeroCampos);
        //Forma Llave
        LLave ="";
        for (int i=0; i<numCamposLlave; i++)
               {if(esLLave[i])
                      LLave +=valoresCampos[i].trim();
               }
        LLave = directorioDeDatos + preFijo+LLave+"."+extension;
        //Investigar si existe el registro
        if (Ip.existeArchivo(LLave))
            {
            Ip.escribeMensaje("El     registro     con     identificacion
("+LLave+") ya existe proceda a actualizarlo");
            return;
            }
        escribeRegistro(LLave);
      }

 public static void consultaRegistro() {
         Ip.wescribeMensaje("Consulta de Registro");
         seleccionaRegistro();
         }
  public static void actualizaRegistro() {
         Ip.wescribeMensaje("Actualiza Registro");
         leeLLave();
```

```
         leeRegistro(LLave);
         Ip.wleeDatos("Datos para " + tabla,
                       etiquetas,
                       valoresCampos,
                       tiposCampos,
                       numeroCampos);
         escribeRegistro(LLave);
        }
   public static void borraRegistro() {
         boolean r = false;
         Ip.wescribeMensaje("Borra Registro");
         leeLLave();
         leeRegistro(LLave);
         Ip.wmuestraDatos("Datos para " + tabla,
                       etiquetas,
                       valoresCampos,
                       tiposCampos,
                       numeroCampos);
         r=Ip.wleeRespuesta("¿Desea   eliminar   permanentemente   este
registro?");
         if(r)
            Ip.borraArchivo(LLave);
        }

     public static void seleccionaRegistro() {
       String nombreArchivoSeleccionado="";
       boolean r = true;
         String[] filt = new String[1];
         filt[0]=extension;
         String[] pre = new String[1];
         pre[0]=preFijo;
         Ip.wescribeMensaje("Selecciona Registro");
         //Selecciona un archivo ubicandose en el difrectorio actual
         while (r)
            {
             nombreArchivoSeleccionado=Ip.seleccionaArchivo(
                       new File(directorioDeDatos),
                        "Seleccione un registro",//encabezado
                       filt, //Filtro utilizado para post fijos
                       pre //Filtro utilizado para pre fijos
                        );


              leeRegistro(nombreArchivoSeleccionado);
              Ip.wmuestraDatos("Datos para " + tabla,
                            etiquetas,
                            valoresCampos,
                            tiposCampos,
                            numeroCampos);
              r=Ip.wleeRespuesta("¿Desea otro registro?");
              if(!r)
             LLave = nombreArchivoSeleccionado;
         }
      }
     public static void leeLLave() {
         //Inicializacion de campos
         boolean existe = false;
          Ip.wleeDatos("Identificación del registro",
                       etiquetas,
                       valoresCampos,
                       tiposCampos,
```

```
                        numCamposLlave);
        //Forma Llave
        LLave ="";
        for (int i=0; i<numCamposLlave; i++)
               {LLave +=valoresCampos[i].trim();
               }
        LLave = directorioDeDatos+preFijo+LLave+"."+extension;
        existe = Ip.existeArchivo(LLave);
        if (!existe)
            {Ip.escribeMensaje(
            "El registro con identificacion ("+
            LLave+
            ")\nno existe, seleccione un registro...");
            seleccionaRegistro();
            return;
            }
      }
    public static void escribeRegistro(String nombreArchivo) {
          Ip.abreArchSalidaNuevo(nombreArchivo);
        for (int i=0; i<numeroCampos; i++)
               {
                      Ip.escribeEnArch(valoresCampos[i].trim());
               }
        Ip.cierraArchSalida();
      }

    public static void leeRegistro(String nombreArchivo) {
          Ip.abreArchEntrada(nombreArchivo);
        for (int i=0; i<numeroCampos; i++)
               {
                  valoresCampos[i]=Ip.leeDeArch();
               }
        Ip.cierraArchEntrada();
      }
      public static void aparienciaDeWindows() {
          JFrame.setDefaultLookAndFeelDecorated(true);
            JDialog.setDefaultLookAndFeelDecorated(true);
            try
            {

      UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsL
ookAndFeel");
            }
            catch (Exception ex)
            {
                  System.out.println("Failed loading L&F: ");
                  System.out.println(ex);
            }
      }
   public static void generaDirectorioDeDatos(String t) {
          //Si ya existe el directorio de datos no lo genera
          if (!Ip.existeArchivo(directorioDeDatos+"\\."))
               Ip.ejecutaConsola("MD "+t);
      }
```

Para terminar tenemos el proceso principal que inicia la ejecución y la terminación del programa:

```
public static void main(String[] args)
      {
          inicializaCampos();
```


```
            menuAplicacion();
    }//Termina el proceso principal
}//Termina el programa
```

La mayoría del código es libre de contexto y puede reutilizarse con mucha facilidad aumentando la productividad en el desarrollo. Note como este programa así como el resto de los programas utilizados en este material solo se apoyan en el paradigma de programación por procedimientos, esto se hace con el fin de simplificar el aprendizaje de de Java y tomar como secuencia la enseñanza estricta de los paradigmas de programación en un estricto orden: Procedimientos, Objetos y Eventos, de tal manera que dicha disciplina facilite el aprendizaje de la programación. Una vez cubierto este curso el estudiante esta en posición de aprender el paradigma de objetos, utilizando Java naturalmente.

11.5.- GUI completa integrada en inovaProg.

inovaProg proporciona dos facilidades que permiten implementar la lectura, almacenamiento y consulta de un conjunto de datos mediante una interfase gráfica.
Las instrucciones involucradas son: Ip.actUnit e Ip.actTab las cuales proporcionas un actualizador unitario de registro y un actualizador tabular del mismo. Para ejemplificar utilizaremos un catálogo de artículos iniciando con el catálogo de unidades y de tipos de artículos para concluir con los datos del artículo.
Para actualizar las unidades podemos utilizar el código siguiente:

```
//Declaraciones
        int numeroCampos  =  2;
        String   etiquetas[]    =  new String [numeroCampos] ;
        String[] tiposCampos    =  new String [numeroCampos] ;
        boolean[] esCampoLlave  =  new boolean[numeroCampos] ;
        String[] tablaExistencia  =  new String [numeroCampos] ;
        String tabla = "Unidades";
        //Inicializacion de campos
        etiquetas[0]="Identificacion";
        etiquetas[1]="Descripcion de la Unidad";
        tiposCampos[0]="S";
        tiposCampos[1]="S";
        for (int i=0; i<numeroCampos; i++)
              {
                esCampoLlave[i]=false;
              }
        esCampoLlave[0]=true;
        tablaExistencia[0]="";
        tablaExistencia[1]="";
        Ip.actUnit(   tabla,
                      etiquetas,
                      tiposCampos,
                      esCampoLlave,
                      tablaExistencia);
```

Produce la ventana de la figura 11.2

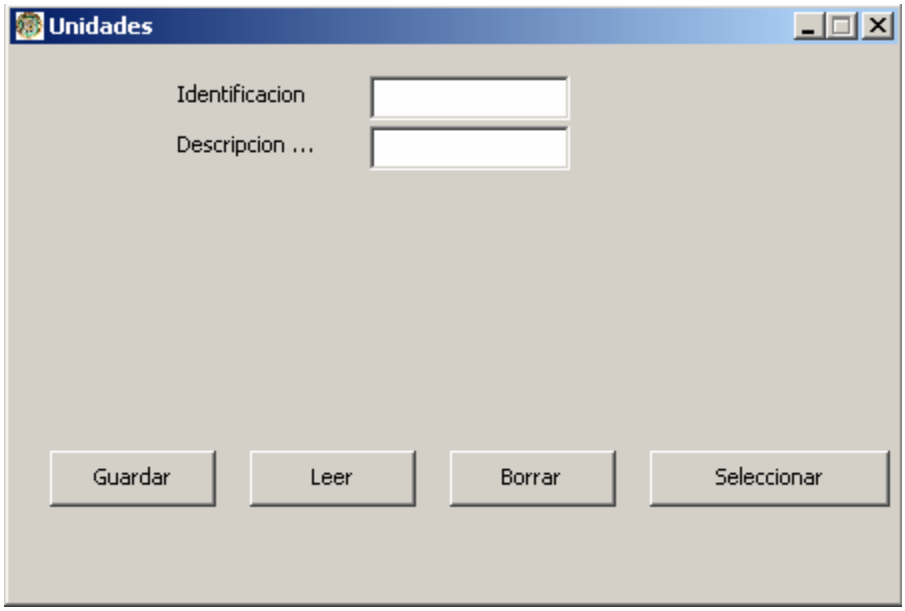

Figura 11.2 Actualizador Unitario de Unidades de Artículos

Para actualizar los tipos podemos utilizar el siguiente código:

```
//Declaraciones
        int numeroCampos  =  2;
        String   etiquetas[]    =  new String [numeroCampos] ;
        String[] tiposCampos    =  new String [numeroCampos] ;
        boolean[] esCampoLlave  =  new boolean[numeroCampos] ;
        String[] tablaExistencia  =  new String [numeroCampos] ;
        String tabla = "Tipos";
        //Inicializacion de campos
        etiquetas[0]="Identificacion";
        etiquetas[1]="Descripcion del tipo";
        tiposCampos[0]="S";
        tiposCampos[1]="S";
        for (int i=0; i<numeroCampos; i++)
              {
                esCampoLlave[i]=false;
              }
        esCampoLlave[0]=true;
        tablaExistencia[0]="";
        tablaExistencia[1]="";
        Ip.actUnit(   tabla,
                      etiquetas,
                      tiposCampos,
                      esCampoLlave,
                      tablaExistencia);
```

Que produce la ventana de la figura 11.3

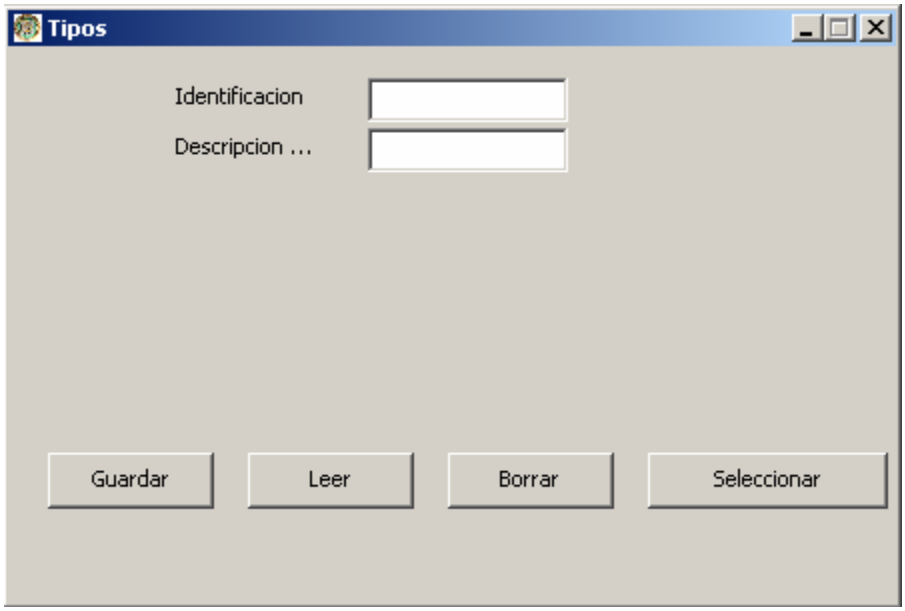

Figura 11.3 Actualizador Unitario de Tipos de Artículos

Finalmente para actualizar los datos del artículo utilizamos el código que se muestra a continuación y que produce la ventana de la figura 11.4

```
int numeroCampos  =  5;
String   etiquetas[]    =  new String [numeroCampos] ;
String[] tiposCampos    =  new String [numeroCampos] ;
boolean[] esCampoLlave  =  new boolean[numeroCampos] ;
String[] tablaExistencia  =  new String [numeroCampos] ;
 String tabla = "Articulos";
 //Inicializacion de campos
 etiquetas[0]="Identificacion";
 etiquetas[1]="Tipo";
 etiquetas[2]="Existencia";
 etiquetas[3]="Fecha Ultima Venta";
 etiquetas[4]="Unidad";
 tiposCampos[0]="S";
 tiposCampos[1]="S";
 tiposCampos[2]="F";
 tiposCampos[3]="D";
 tiposCampos[4]="S";
 for (int i=0; i<numeroCampos; i++)
        {
          esCampoLlave[i]=false;
        }
 esCampoLlave[0]=true;
 tablaExistencia[0]="";
 tablaExistencia[1]="Tipos";
 tablaExistencia[2]="";
 tablaExistencia[3]="";
 tablaExistencia[4]="Unidades";
 Ip.actUnit(   tabla,
               etiquetas,
               tiposCampos,
               esCampoLlave,
```

```
                                    tablaExistencia);
```

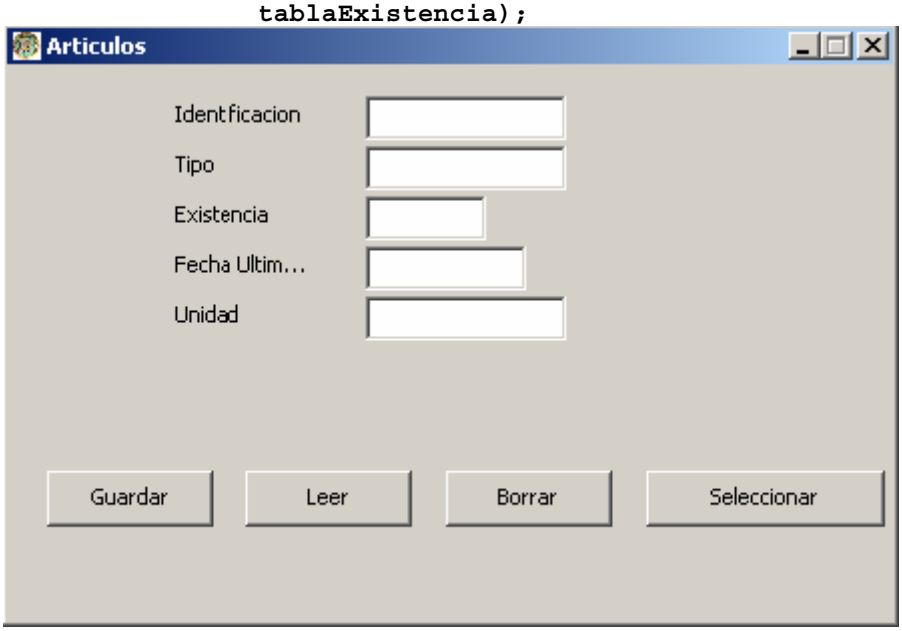

Figura 11.4 Actualizador Unitario de Artículos

Los códigos anteriores crean los subdirectorios \Unidades, \Tipos y \Articulos, respectivamente dónde se almacenan los datos grabados.
Ip.actUnit(tabla,etiquetas,tiposCampos, esLLave, tablaExistencia); por Ip.actTab(tabla,etiquetas,tiposCampos, esLLave, tablaExistencia); podemos obtener los actualizadores tabulares. Estos actualizadores se muestran completos incluyendo un codigo constructor necesario para su ejecución en un menú en los programas:
1106ActualizadorUnitarioArticulos.java
1107ActualizadorTabularArticulos.java
1108ActualizadorUnitarioTipos.java
1109ActualizadorTabularTipos.java
1110ActualizadorUnitarioUnidades.java
1111ActualizadorTabularUnidades.java

Para integrar las aplicaciones en un solo menú puede observar el programa 1112MenuAplicacion.java

```
int numOpciones=3;
      int maxNumSubOpciones=3; //numero maximo de subopciones
      String titulo="Actualizadores";
      String[] opciones        = new String[numOpciones] ;
String[][] subOpciones   = new String[numOpciones][maxNumSubOpciones] ;
String[][] acciones      = new String[numOpciones][maxNumSubOpciones] ;
      opciones[0] = "Actualizadores Unitarios";
      opciones[1] = "Actualizadores Tabulares";
      opciones[2] = "Mantenimiento";
      subOpciones[0][0]="Act. Articulos";
```

```
      subOpciones[0][1]="Act. Unidades";
      subOpciones[0][2]="Act. Tipos";
      subOpciones[1][0]="Act. Articulos";
      subOpciones[1][1]="Act. Unidades";
      subOpciones[1][2]="Act. Tipos";
      subOpciones[2][0]="Salida";
      //
      acciones[0][0]="ActualizadorUnitarioArticulos";
      acciones[0][1]="ActualizadorUnitarioUnidades";
      acciones[0][2]="ActualizadorUnitarioTipos";
      acciones[1][0]="ActualizadorTabularArticulos";
      acciones[1][1]="ActualizadorTabularUnidades";
      acciones[1][2]="ActualizadorTabularTipos";
      acciones[2][0]="Salida";
      aparienciaWindows();
         Ip.menuBarras(titulo,opciones,subOpciones,acciones);
```

Este menú requiere que se modifique la clase IpEjecuta contenida en el archivo Ip.java para que interprete las acciones a tomar en el menú:

```
class IpEjecuta {
public static boolean accion(String accion) {
      //Ip.wescribeMensaje("Ejecuta ("+ accion +")");
    if (accion.toUpperCase().equals("SALIDA"))
              {System.exit(0);
               return true;
              }
    else
if (accion.toUpperCase().equals("ACTUALIZADORUNITARIOARTICULOS"))
             {new ActualizadorUnitarioArticulos();
             return true;}
    else
if (accion.toUpperCase().equals("ACTUALIZADORUNITARIOUNIDADES"))
             {new ActualizadorUnitarioUnidades();
             return true;}
    else if (accion.toUpperCase().equals("ACTUALIZADORUNITARIOTIPOS"))
             {new ActualizadorUnitarioTipos();
             return true;}
     else
if (accion.toUpperCase().equals("ACTUALIZADORTABULARARTICULOS"))
             {new ActualizadorTabularArticulos();
             return true;}
    else
if (accion.toUpperCase().equals("ACTUALIZADORTABULARUNIDADES"))
             {new ActualizadorTabularUnidades();
             return true;}
    else if (accion.toUpperCase().equals("ACTUALIZADORTABULARTIPOS"))
             {new ActualizadorTabularTipos();
             return true;}
    else
        return false;
    }
}
```

Este programa debe ser el último en compilarse ya que requiere que todos los demas se hayan compilado anteriormente.

El menú genera la ventana que se muestra en la figura 11.5

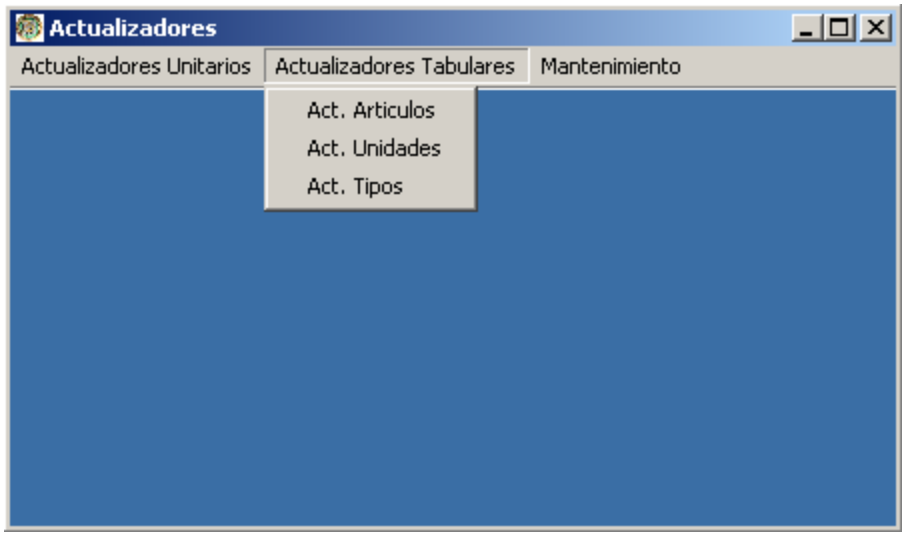

Figura 11.5 Menú de la aplicación.

De esta forma se cuenta con todos los bloques básicos para implementar un sistema sencillo.

11.6.- Actividades.

Compile y ejecute los ejemplos agregando distintos tipos de llamadas para Ip.wleeDatos e Ip.wmuestraDatos. Note como debe de instalar el paquete inovaProg.jar en la ruta de bibliotecas externas de java o, la otra opción es contar con los archivos clases Ip.class, IpwleeDatos.class e IpleeDatos.class en el mismo directorio en el cual está compilando y ejecutando su programa.

11.7.- Ejercicio propuesto.

Escriba un programa que capture los datos de un catalogo de artículos en dónde cada artículo debe de tener los siguientes datos: id entero, descripción cadena de caracteres, existencia flotante, tipo de articulo, el cual debe existir en otra colección de objetos llamada tipos la cual debe también contar con un actualizador para su mantenimiento. Forme un menú para ejecutar sus facilidades.