CONTENIDO

DESCRIPCIÓN

A lo largo de la implementación de Prometeo se encontraron ciertos fragmentos de código en lenguaje JAVA o comandos de la JVM que se ejecutaban de una manera en una plataforma y los resultados obtenidos en otra plataforma (para el caso particular WINDOWS y LINUX) no eran los esperados. Las inconsistencias que se encontraron y que pueden afectar la portabilidad del código desarrollado en este lenguaje se muestran a continuación

1. Inconsistencias en los eventos de ratón

El siguiente código muestra un JFrame al que se le ha adicionado un objeto de tipo ExecuteInfo (una clase que hereda de JscrollPane y contiene un JTextArea). Al área de texto se le ha adicionado un oyente de eventos y se ha redefinido el método MouseReleased(MouseEvent e). Al detectar el evento en el ratón, a través del método MaybeShowPopup se verifica si se pulso el clic derecho o izquierdo.

```
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import java.awt.BorderLayout;

public class ExecuteInfo extends JScrollPane {
  private JTextArea salida;
  private javax.swing.JPopupMenu popup = new javax.swing.JPopupMenu();
  private javax.swing.JMenuItem itemCopiar;
  private javax.swing.JMenuItem itemBorrar;

  public ExecuteInfo() {
    salida = new JTextArea("      Area de Texto");
    itemCopiar=new javax.swing.JMenuItem("Copiar texto seleccionado");
    itemBorrar=new javax.swing.JMenuItem("Limpiar area de mensajes");
    popup.add(itemCopiar);
    popup.add(itemBorrar);
    getViewport().add(salida, null);
    salida.addMouseListener(new java.awt.event.MouseAdapter(){
      /**evento en Windows*/
      public void mouseReleased(java.awt.event.MouseEvent e) {
          maybeShowPopup(e);
        }
      });
```

```
    itemCopiar.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e) {
        }
    });
    itemBorrar.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e) {
       }
    });
  }

  private void maybeShowPopup(java.awt.event.MouseEvent e) {
    if (e.isPopupTrigger()) {
      popup.show(e.getComponent(),
                 e.getX(), e.getY());
    }
  }

  public static void main(String[] args){
    JFrame frame=new JFrame();
    ExecuteInfo info=new ExecuteInfo();
    frame.getContentPane().add(info,BorderLayout.CENTER);
    frame.setBounds(300,300,300,300);
    frame.show();
  }
}
```

Al ejecutar este código sobre la plataforma WINDOWS el resultado es el siguiente:

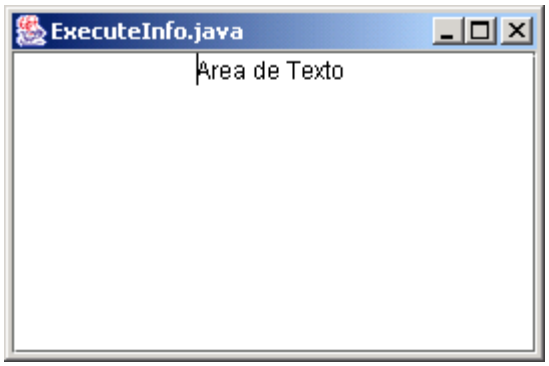

Figura 1

Al hacer clic derecho sobre el área de texto, la ventana se ve de la siguiente forma:

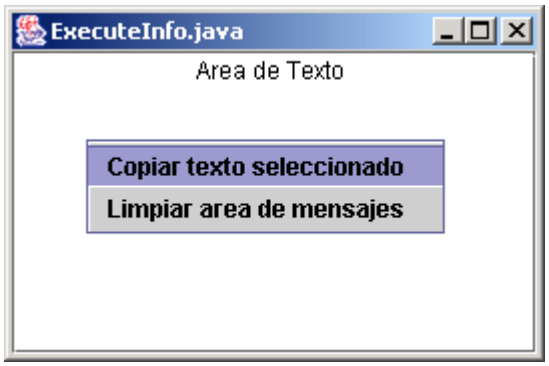

Figura 2

Esto es debido a que se reconoce el evento MouseReleased(MouseEvent e), se verifica que se ha pulsado clic derecho y entonces se despliega el componente correspondiente al PopupMenu.

Ahora, si exactamente este mismo código es ejecutado sobre la plataforma LINUX, se obtendrá el mismo resultado de la Figura 1. Sin embargo, al hacer clic derecho sobre el área de texto, el evento MouseReleased(MouseEvent e) no es activado, por lo que no se muestra el componente correspondiente al PopupMenu.

Para que el evento de clic derecho funcione tanto en WINDOWS como en LINUX, se hizo necesario implementar el Método MousePressed(MouseEvent e) en la Interfaz MouseAdapter, con esto, el código queda de la siguiente forma:

```
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import java.awt.BorderLayout;

public class ExecuteInfo extends JScrollPane {
  private JTextArea salida;
  private javax.swing.JPopupMenu popup = new javax.swing.JPopupMenu();
  private javax.swing.JMenuItem itemCopiar;
  private javax.swing.JMenuItem itemBorrar;

  public ExecuteInfo() {
    salida = new JTextArea("        Area de Texto");
    itemCopiar=new javax.swing.JMenuItem("Copiar texto seleccionado");
    itemBorrar=new javax.swing.JMenuItem("Limpiar area de mensajes");
    popup.add(itemCopiar);
    popup.add(itemBorrar);
    getViewport().add(salida, null);
```

```
    salida.addMouseListener(new java.awt.event.MouseAdapter(){

    /**evento en Windows*/
    public void mouseReleased(java.awt.event.MouseEvent e) {
      maybeShowPopup(e);
    }
    /**Evento en Linux*/
    public void mousePressed(java.awt.event.MouseEvent e){
      maybeShowPopup(e);
    }

    });
    itemCopiar.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e) {
        }
    });
    itemBorrar.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e) {
       }
    });
  }

  private void maybeShowPopup(java.awt.event.MouseEvent e) {
    if (e.isPopupTrigger()) {
      popup.show(e.getComponent(),
                 e.getX(), e.getY());
    }
  }

  public static void main(String[] args){
    JFrame frame=new JFrame();
    ExecuteInfo info=new ExecuteInfo();
    frame.getContentPane().add(info,BorderLayout.CENTER);
    frame.setBounds(300,300,300,300);
    frame.setTitle("ExecuteInfo.java");
    frame.show();
  }
}
```

2. Inconsistencias en la utilización del comando javap (Descompilador)

El comando *javap* es utilizado para descompilar archivos *.class*, esto es, extraer los métodos y atributos que contiene una clase a partir de su código objeto. Cuando se va a utilizar este comando desde una aplicación java y se van a mostrar los resultados, debe tenerse cuidado en la forma como se obtienen.

La siguiente prueba fue realizada para descompilar el archivo vista.FormError, cuyo código aparece a continuación:

```
package vista;

/**
 *@author Gabriel Vasquez
 *@version 1.0
 *Fecha de Creación:2004-08-09 10:18
 */

import javax.microedition.lcdui.*;

public class FormError extends Form implements CommandListener{

/**Atributos*/
 Command okCommand=new Command("OK",Command.OK,1);
 ImageItem errorItem;
 Image errorImage;


/**Crea un nuevo objeto de tipo FormError*/
 public FormError(){
  super("Información");
  try{
    init();
  }catch(Exception e){
   e.printStackTrace();
   }
  }
 /**Inicializacion de componentes*/
 private void init() throws Exception {
  setCommandListener(this);
  addCommand(okCommand);
  errorImage=Image.createImage("/imagenes/cancel.png");
  errorItem=new ImageItem("\nNo se pudo completar la\n
operacion\n",
                          errorImage,ImageItem.LAYOUT_CENTER, "");
  this.append(errorItem);
  }

/**Manejo de eventos*/
 public void commandAction(Command command, Displayable displayable) {
 if(command.equals(okCommand)){
    MIDletValidar.instance.setForm(0);
  }
 }
}
```

Las siguientes líneas corresponden al resultado obtenido producto de la ejecución del comando:

Javap -private –classpath

/usr/local/WTK2.0/lib/mmapi.zip:/usr/local/WTK2.0/lib//midpapi.zip:/usr/home/Desktop/pruebas/validacion/clases vista.FormError

Sobre el sistema operativo Linux Slackware 10 y Linux Mandrake 9.1.

```
1.Compiled from "FormError.java"
2.public class vista.FormError extends javax.microedition.lcdui.Form
  implements javax.microedition.lcdui.CommandListener{
3.    javax.microedition.lcdui.Command okCommand;
4.    javax.microedition.lcdui.ImageItem errorItem;
5.    javax.microedition.lcdui.Image errorImage;
6.    public vista.FormError();
7.  private void init();
8.     throws java/lang/Exception
9. public void commandAction
      (javax.microedition.lcdui.Command,javax.microedition.
       lcdui.Displayable);
10. }
```

y las siguientes corresponden a la ejecución del comando:

Javap -private –classpath C:/WTK20/lib/mmapi.zip;C:/WTK20/lib/midpapi.zip;C:/Prometeo/pruebas/Validacion/clases vista.FormError

En el sistema operativo Windows 2000 Profesional.

```
1. Compiled from FormError.java
2. public class vista.FormError extends javax.microedition.lcdui.Form
   implements javax.microedition.lcdui.CommandListener {
3. javax.microedition.lcdui.Command okCommand;
4. javax.microedition.lcdui.ImageItem errorItem;
5. javax.microedition.lcdui.Image errorImage;
6. public vista.FormError();
7. private void init() throws java.lang.Exception;
8. public void commandAction(javax.microedition.lcdui.Command,
   javax.microedition.lcdui.Displayable);
9. }
```

como puede apreciarse, los resultados son casi los mismos, pero para el caso del método *private void init( ) throws Exception*, los resultados en Linux se muestran en dos líneas diferentes (7 y 8), mientras que en Windows 2000 se muestran en una sola línea (7). Esto

puede afectar debido a que la forma de capturar los resultados de comandos se hace línea por línea a través de flujos de entrada.

En el caso de prometeo, se utiliza el comando javap para descompilar archivos .class y obtener los atributos y métodos que serán mostrados en el árbol que muestra el modelo de atributos y métodos de cada clase.

El código necesario para lograr coherencia en los resultados mostrados tanto en la plataforma Windows como Linux es el siguiente:

```
package componentes;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Vector;

public class Decompilador implements java.io.Serializable{
  private String OS;
  private String line;
  private String classpath;
  private final String sep=File.pathSeparator;
  private final String javap = "javap -private -classpath ";
  private String[] metodos;


  /*Crea un nuevo objeto de tipo Decompilador*/
  public Decompilador() {
    String aux=System.getProperty("os.name");
    if(aux.lastIndexOf("Windows")!=-1)
      OS="Windows";
    else
      OS="Linux";

  }

  /**
   * devuelve el arreglo metodos
   * @param dir
   * @param classFile
   * @return
   */
  public String[] getMetodos(String dir, String classFile) {
    return this.metodos;
  }

  /**
   * Obtiene los atributos de un archivo class
   * @param dir el path del directorio que contiene el archivo class
```

```
   * @param classFile el nombre del archivo class
   * @return
   */
  public String[] getFields(String dir, String classFile) {
    Vector aux = new Vector();
    Vector mets=new Vector();
    String[] atribs;
    String cmd;
    try {

      if(OS.equals("Windows"))
        cmd = javap + "\""+classpath + dir +"\""+ " " + classFile;
      else
        cmd = javap + classpath + dir + " " + classFile;

      System.out.println("Decompilador: "+cmd);
      Runtime r = Runtime.getRuntime();
      Process p = r.exec(cmd);
      BufferedReader in = new BufferedReader(new InputStreamReader(p
          .getInputStream()));
      int n = 0;
      while ( (line = in.readLine()) != null) {
        if (line.lastIndexOf("(") != -1) {
            mets.addElement(line);
        }
        else if (n > 1)
          aux.add(line);
        n++;
      }
      in.close();
      this.metodos=new String[mets.size()];
      for(int a=0;a<mets.size();a++){
       metodos[a]=mets.elementAt(a).toString();
      }

      for(int i=0; i<aux.size();i++){
        String content=aux.elementAt(i).toString();
        if(content.equals("}")){
          aux.remove(i);
          i--;
        }
        else if(content.equals("")){
          aux.remove(i);
          i--;
        }
        else if(content.indexOf("throws")!=-1){
         aux.remove(i);
         i--;
        }
      }
      atribs=new String[aux.size()];
      for(int i=0;i<aux.size();i++){
        atribs[i]=aux.elementAt(i).toString();
      }
      return atribs;
    }
    catch (IOException e) {
```

```
      e.printStackTrace();
      return null;
    }
  }


  /**
   * fija el classpath
   * @param classpath
   */
  public void setClasspath(String dir) {
    try {
       String path = dir + "/lib";
       String allPath = "";
       File file = new File(path);
       String[] lista = file.list(new ClassPathFilter());
       for (int i = 0; i < lista.length; i++) {
         allPath = allPath + path + "/" + lista[i] + sep;
       }
       this.classpath = allPath;
      }
     catch (NullPointerException e) {}

  }

}
```