Universidad de Salamanca

IHM - JAVA

Depto. Informática y Automática
Máster en Sistemas Inteligentes
Dr. J.R. García-Bermejo Giner

IHM - Java

Primera parte - Java

Vamos a exponer las soluciones de varios problemas recurrentes en IHM

- Importación de información heredada (tratamiento de archivos de texto).
- Generación de un léxico (uso de colecciones).
- Visualización de una interfaz gráfica de usuario (mecanismo de activación).

IHM - Java

Descarga de software: maxus.fis.usal.es/DOCTORADO

Universidad de Salamanca	Departamento de Informática y Automática	**Bienvenidos a Maxus!** © 2006- José R. García-Bermejo Giner

Docencia	▪ Programación  (26/02/2009 12:26) ▪ Sistemas de Información  (20/05/2009 18:41) ▪ Curso de Tecnologías Avanzadas en Java (Zamora 2008) (26/02/2009 12:26)

IHM - Java

La interfaz TextUtils.java contiene una colección de métodos que permiten leer y escribir ficheros de texto con distintos formatos de manera cómoda y razonablemente rápida:

```
package com.coti.textfiletools;
import java.io.File;

public interface TextUtils {
      public static final String lineSeparator = System.getProperty("line.separator");
      public String readString(File f);
      public boolean writeString(String str, File f);
      public boolean writeStringAppending(String str, File f);
      public boolean writeList(String[] list, File f);
      public boolean writeListAppending(String[] list, File f);
      public String[] readList(File f);
      public boolean writeDelimitedTable(String[][] table, String delimiter, File f);
      public boolean writeColumnarTable(String[][] table, int[] lengths, File f);
      public boolean writeColumnarTable(String[][] table, int[] lengths);
      public String[][] readDelimitedTable(String delimiter, File f);
      public String[][] readColumnarTable(int[] field_length, File f);
}
```

IHM - Java

La clase AuxText implementa TextUtils empleando una técnica que permite seleccionar métodos óptimos para la plataforma en que se esté ejecutando el programa, de manera automática. Esto exige, desde luego, efectuar pruebas en todas las plataformas consideradas.

```
public class AuxText implements TextUtils {
    private AuxTextLo lo = null;
    private AuxTextHi hi = null;
    private final int UNIX        = 0;
    private final int WINDOWS     = 1;
    TextUtils[][] preferred = null;
    private int platform;
    private Component owner = null;
    public AuxText(Component owner)
    {
        this.owner= owner;
        lo = new AuxTextLo(owner);
        hi = new AuxTextHi(owner);
        TextUtils[][] tuned = {
            /* Unix, Windows */
            {hi, lo}, /*  0 readString                        */
            {hi, hi}, /*  1 writeString             hi hi     */
            {hi, hi}, /*  2 writeStringAppending    hi hi     */
            {lo, lo}, /*  3 readList                          */
            {hi, hi}, /*  4 writeList               hi hi     */
            {hi, hi}, /*  5 writeListAppending      hi hi     */
            {lo, hi}, /*  6 writeDelimitedTable     lo hi     */
            {lo, lo}, /*  7 readDelimitedTable                */
            {lo, lo}, /*  8 writeColumnarTable                */
            {lo, lo}, /*  9 writeColumnarTable                */
            {lo, lo}  /* 10 readColumnarTable                 */
        };
```

IHM - Java

La clase AuxTextTest muestra una forma de probar AuxText. Debe ejecutase en distintas plataformas, para así determinar el rendimiento real de AuxText. Esto permite estimar cual de las versiones de estos métodos resulta ser más rápida. Esa será la versión seleccionada.

```
import java.io.*;
import com.coti.textfiletools.*;
/*
+-----------------------------------------------------------------------------------------+
| This is a time-measuring procedure for AuxText{Hi,Lo} Procedure is simple: a time reading is   |
| taken  immediately before calling a method, and another is taken upon returning. Time shown is |
| the difference between them.                                                            |
+-----------------------------------------------------------------------------------------+
    Latest changes 20080507
*/
public class AuxTextTest {

    public static void main(String[] args) {
        final int NUMROWS = 1000;
        final int NUMCOLS = 100;
        AuxText at = new AuxText(null);
        File inputData = new File("inputdata.txt");
        File outputData = new File("outputdata.txt");
        String[] slist = null;
        String[][] resultingTable = null;
        String[][] stable = at.createTable(NUMROWS, NUMCOLS);
        int[] columnWidths = new int[NUMCOLS];
        long startTime, endTime;
        for(int i=0; i < NUMCOLS; i++)
            columnWidths[i] = 15;
```

IHM - Java

El problema de generación de un léxico, determinación de las frecuencias de aparición de las distintas palabras y creación de listas ordenadas por ambas claves se observa en numerosas ocasiones. Este método permite calcular ambas cosas; de hecho se puede aplicar a volúmenes razonables de información (El Quijote!)

```
import java.util.*;
import java.io.File;
import com.coti.tools.*;

public class Lexico_y_Frecuencias {
    public static void main(String[] args) {
        Map<String, Integer> m         = new TreeMap<String, Integer>();
        AuxText at                     = new AuxText(null);
        File f                         = null;
        String texto                   = null;
        String[] listaDeFragmentos     = null;
        String temp                    = null;
        String[] faw                   = null;
        int numeroDeFragmentos         = 0;
        int numeroDePalabrasDistintas  = 0;
        Set<String> keySet             = null;
        char[] encabezado              = new char[50];
        Integer frecuencia             = null;
        if (args.length == 0)
          {
            System.out.print("\nUso: java -jar laf.jar archivo_entrada\n\n");
            return;
          }
        try
          {
            f = new File(args[0]);
          }
        catch(Exception e)
          {
            System.out.println("\nNo existe el archivo " + args[0] + ".\n");
             return;
          }
```

IHM - Java

Java permite serializar la totalidad de una aplicación para después recargarla y devolverle la totalidad de su estado. Este primer ejemplo muestra la forma de guardar en disco la totalidad del estado de una aplicación. El ejemplo siguiente muestra la forma de recuperar todo el estado sin ejecutar de nuevo el proceso de construcción de la interfaz gráfica de usuario!

```
    static public void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                File archivo = new File("aplicacion.nib");
                OutputStream fos = null;
                ObjectOutputStream oos = null;
                try {
                    fos = new FileOutputStream(archivo);
                    oos = new ObjectOutputStream(fos);
                    JFrameAI marco = new JFrameAI("Demostración de campos de texto",
                            JFrameAI.PLATFORM,
                            0.0);
                    DemoCamposTexto dct = new DemoCamposTexto();
                    oos.writeObject(dct);
                    fos.close();
                    marco.getContentPane().add(dct, BorderLayout.CENTER);
                    marco.pack();
                    marco.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
    });
```

IHM - Java

Este es el proceso inverso: se carga todo el estado de la aplicación, que queda en disposición de funcionar normalmente sin necesidad de volver a ejecutar el laborioso proceso de construcción de la interfaz gráfica de usuario.

```
    static public void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                File archivo = new File("aplicacion.nib");
                FileInputStream fis = null;
                ObjectInputStream ois = null;
                try {
                    fis = new FileInputStream(archivo);
                    ois = new ObjectInputStream(fis);
                    JFrameAI marco = new JFrameAI("Demostración de campos de texto",
                            JFrameAI.PLATFORM,
                            0.0);
                    DemoCamposTexto dct = (DemoCamposTexto)ois.readObject();
                    ois.close();
                    marco.getContentPane().add(dct, BorderLayout.CENTER);
                    marco.pack();
                    marco.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
```

IHM - Java

Conclusiones

El procedimiento que se ha descrito en los dos últimos ejercicios plantea la posibilidad de almacenar aplicaciones en forma de imágenes binarias que solo precisan ser cargadas (no ejecutadas) para funcionar correctamente.

En el caso de aplicaciones que contengan datos de lectura costosa, o que precisen efectuar cálculos relativamente complejos, este procedimiento supone un agradable incremento de la velocidad de arranque.

Una compleja interfaz gráfica de usuario no tiene por qué suponer la virtual paralización de la aplicación en el momento del arranque.

Problema: no todos los componentes de Java admiten actualmente la serialización:

Warning: Serialized objects of this class will not be compatible with future Swing releases. The current serialization support is appropriate for short term storage or RMI between applications running the same version of Swing. A future release of Swing will provide support for long term persistence.

Pero la idea final de Java apunta en la dirección sugerida.

IHM - Java

El paso siguiente --->
==========

Las técnicas descritas han sido desarrolladas extensamente en el lenguaje predecesor de Java, Objective-C.

Este lenguaje, que forma parte de GNUC, tiene asociadas bibliotecas de clases destinadas a la implementación de interfaces gráficas de usuario (GNUStep, Cocoa).

A continuación se estudia Objective-C como lenguaje, para después aplicarlo a la construcción de aplicaciones en Cocoa.

Universidad de Salamanca

IHM - JAVA

Depto. Informática y Automática
Máster en Sistemas Inteligentes
Dr. J.R. García-Bermejo Giner