Laboratorio de Redes de Comunicaciones
Recursos de Sistema en Java

Properties

System

Runtime

Swing

UNIVERSIDADE DA CORUÑA

Properties (I)

- Un “property” define atributos de forma persistente.
  - Son adecuados cuando los valores de los atributos deben de mantenerse en varias invocaciones de un programa.
- Un atributo tiene dos partes:
  - Un nombre y un valor.
  - La clase java.util.Properties gestiona un conjunto de pares clave/valor.
    - Un par clave/valor es como una entrada de un diccionario.
      - La clave es la palabra
      - El valor es la definición.
    - Cada clave contiene el nombre de un atributo del sistema; como valor posee el valor actual del atributo.

Properties (II)

- Cualquier programa Java puede utilizar un objeto Properties para gestionar sus atributos.
- Proporciona métodos para:
  - Cargar pares clave/valor en un objeto Properties desde un "stream".
  - Recuperar un valor a partir de su clave
  - Listar las claves y sus valores
  - Obtener una enumeración de sus claves
  - Guardar las propiedades en un "stream".
- Y hereda todos los métodos de java.util.Hashtable.

- Métodos de properties:
  - String getProperty(String key)
  - String getProperty(String key, String defaultValue)
  - void list(PrintStream out)
  - void list(PrintWriter out)
  - void load(InputStream inStream)
  - Enumeration propertyNames()
  - void save(OutputStream out, String comments)
  - Object setProperty(String key, String value)
  - void store(OutputStream out, String comments)

System

- La plataforma Java permite a los programas acceder a los recursos del sistema a través de una API relativamente independiente del sistema implementada por la clase *java.lang.System* y por la clase *java.lang.Runtime*.
- La clase System permite a programas Java utilizar recursos del sistema aislándolos de detalles específicos de entorno.
  - por ejemplo, acceder a recursos del sistema
    - Flujos de I/O estándar y de error: System.in, System.out, System.err.

Runtime (I)

- Toda aplicación java tiene una única instancia de la clase **java.lang.Runtime** que permite a la aplicación interactuar con el entorno en el que la aplicación se ejecuta.
- No se puede crear una instancia de **Runtime**, sin embargo, se puede obtener una referencia al objeto **Runtime** que se está ejecutando actualmente llamando al método estático **Runtime.getRuntime().**
- El objeto **Runtime** permite:
  - Comunicarse con los componentes de entorno de ejecución
    - Obtener información (Por ejemplo, la cantidad de memoria total utilizada por la JVM)
    - Invocar funciones
  - Actuar de interfaz con las capacidades dependientes del sistema.

Runtime (II)

- La clase Runtime permite crear un proceso nativo del sistema operativo, y se representa con el objeto java.lang.Process
  - Los métodos de creación de procesos pueden no funcionar bien para procesos especiales en determinadas plataformas nativas, como procesos de ventanas, demonios, procesos Win16/DOS en Windows o scripts basados en shells.
  - Los subprocesos creados no tienen su propia terminal o consola. Todas sus I/O estándar (stdin, stdout, stderr) se redirigen al proceso padre a través de tres flujos: getOutputStream, getInputStream y getErrorStream.
  - El proceso padre utiliza estos flujos para comunicarse con el subproceso.
  - Con algunas plataformas nativas que poseen limitaciones en buffers de entrada/salida, pueden producirse problemas al leer/escribir, incluso pudiendo bloquear el subproceso.
  - El subproceso no se destruye cuando no quedan más referencias a él; continúa ejecutándose de forma asíncrona.
- Métodos (java.lang.Process):
  - abstract void destroy()
    - Finaliza/destruye los subprocesos.
  - abstract int exitValue()
    - Devuelve el valor de retorno para el subproceso.
  - abstract InputStream getErrorStream()
    - Devuelve el flujo de error del subproceso.
  - abstract InputStream getInputStream()
    - Obtiene el flujo de entrada del subproceso.
  - abstract OutputStream getOutputStream()
    - Obtiene el flujo de salida del subproceso.
  - abstract int waitFor()
    - Hace que el thread actual espere, si es necesario, hasta que el proceso representado por este objeto haya terminado.

Runtime (III)

- java.lang.Runtime
  - Process **exec**(String command)
  - Process **exec**(String command, String[] envp)
  - Process **exec**(String command, String[] envp, File dir)
    - Ejecuta el comando especificado en un proceso diferente, pudiendo especificar
      - Argumentos
      - Un entorno determinado [(cada string tiene la forma nombre=valor). Si envp=null ➔ el proceso hereda el proceso del padre]
      - Un directorio de ejecución.

```
try {
    String[] cmdarray = new String[1];
    cmdarray[0] = “script.bat”;
    Process p = Runtime.getRuntime().exec(cmdarray);
    int exitValue = p.waitFor();
    System.out.println(“ExitValue: “ + exitValue);
} catch (Exception e) {
    e.printStackTrace();
}
```

Runtime (IV)

- java.lang.ProcessBuilder (jdk5.0)
  - Alternativa para la creación de procesos del sistema operativo nativo.
    - Process p = new ProcessBuilder("myCommand", "myArg").start();

Interfaces gráficas en JAVA

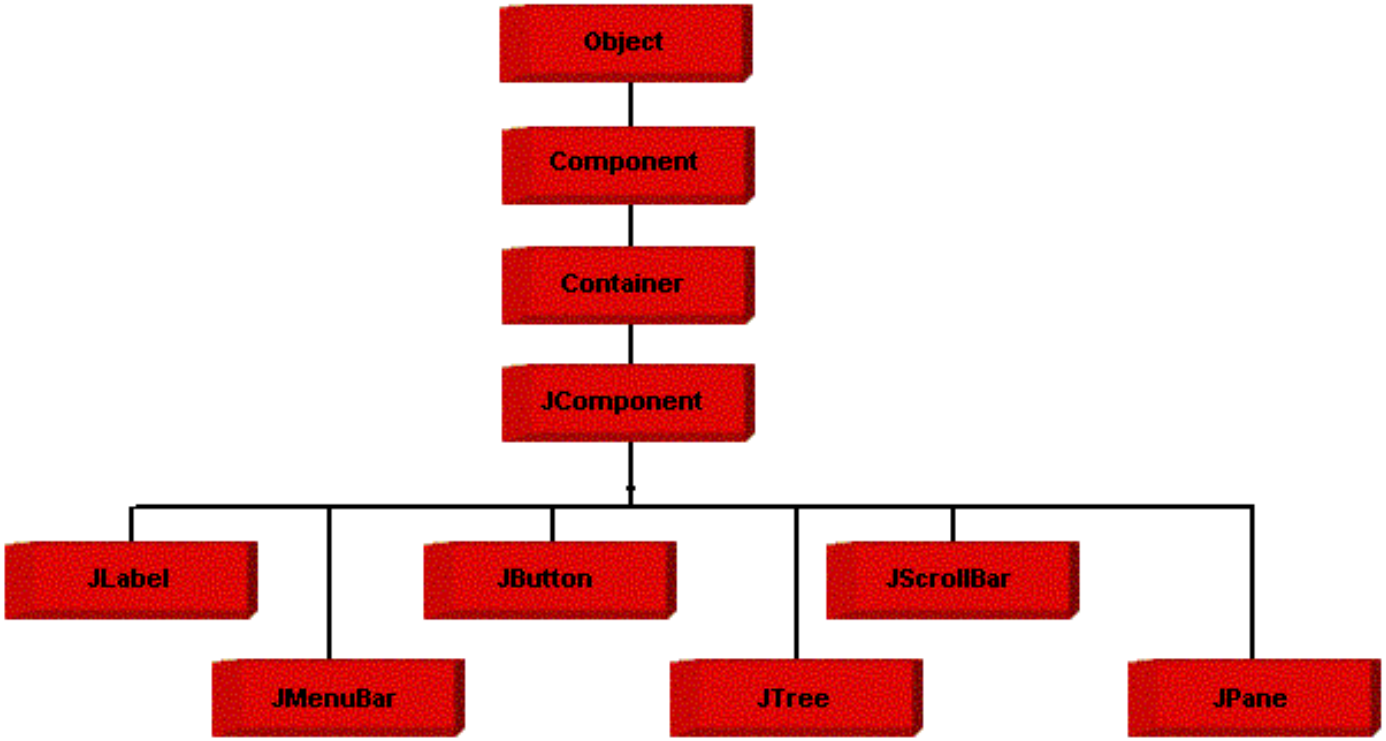
- Hay dos formas fundamentales de crear interfaces gráficas con JAVA:
  - Abstract Window Toolkit (AWT).
    - La primera en aparecer.
    - Aspecto dependiente del Sistema Operativo.
    - Al requerir recursos de la interfaz de ventanas del Sistema Operativo, se suelen denominar componentes pesados.
  - Swing.
    - Complementa y extiende AWT.
    - Aspecto independiente del Sistema Operativo.
    - Componentes más ligeros.
    - Es la utilizada mayoritariamente hoy en día.

Componentes en Java Swing (I)

- Las Interfaces gráficas en swing se crean combinando diversos componentes gráficos predefinidos (y que pueden ser extendidos).
- Cada componente define un aspecto gráfico determinado y tiene una serie de propiedades.
  - Ejemplo:
    - El componente *JFrame* sirve para crear una ventana de un tipo determinado.
    - Al crearlo, se puede indicar un título para el mismo.
- Un componente es un contenedor para otros componentes.
  - Ejemplo:
    - Al componente *JFrame* pueden añadirse, entre otros, componentes *JLabel* (para añadir etiquetas de texto), *JButton* (para añadir un botón que, al ser pulsado, desencadena una acción determinada), *JMenuBar* (para añadirle una barra de menús...).

Componentes en Java Swing (II)

- Los componentes Swing descienden de la clase *java.awt.Container*.
- La mayoría (excepto los contenedores de primer nivel) también descienden de *java.swing.Jcomponent*.



Hello World ! (II)

```
import javax.swing.JFrame;
import javax.swing.JLabel;


public class HelloWorldFrame extends JFrame{

          public static void main(String args[]){
                    new HelloWorldFrame();
          }

          HelloWorldFrame(){
                    JLabel jlbHelloWorld = new JLabel("Hello World");
                    add(jlbHelloWorld);
                    this.setSize(100, 100);
//                  pack();
                    setVisible(true);
          }
}
```

Hello World ! (II)

- Creamos una clase que extiende de *JFrame* para crear nuestro contenedor de primer nivel.
- Creamos un componente *JLabel* con el texto que queremos mostrar.
- Se lo añadimos a nuestro *JFrame*.
- Fijamos el tamaño deseado para el *JFrame* usando *setSize*.
- Una alternativa para fijar el tamaño de la ventana es hacer que se ajuste al de sus subcomponentes llamando al método *pack()*.
  - Nótese que en el ejemplo esta llamada aparece comentada. Puede comentarse *setSize* y descomentar *pack()* para ver la diferencia.
- Hacemos visible el componente.

Algunos Componentes Típicos

- *JPanel*. Contenedor común de contenido.
- *JFrame*. Tipo común de ventana. Puede mostrar un Panel.
- *JTabbedPane*. Múltiples pestañas dentro de una ventana.
- *JLabel*. Etiquetas de texto.
- *JTextField*. Permite editar una línea de texto.
- *JTextArea*. Permite editar múltiples líneas de texto.
- *JCheckBox*. Casilla de verificación.
- *JRadioButton*. Escoger una de entre varias opciones.
- *JList*. Listas de selección
- *JButton*. Botones.
- *JMenuBar*. Barra de menú.
- …

Layouts

- ¿Cómo organizar en una ventana los diferentes elementos que se añaden?
- Algunos elementos tienen lugares predefinidos (*JMenuBar*,…).
  - *JFrame* tiene un método *setMenuBar*.
- En general, podemos aplicar distintos tipos de layout a una ventana:
  - *FlowLayout*. Los componentes se van situando de izquierda a derecha y de arriba abajo.
  - *BorderLayout*. Divide la ventana en 5 regiones: NORTH, SOUTH, EAST, WEST y CENTER.
  - *GridLayout*. Divide la ventana en una serie de celdas de igual tamaño. Cada componente puede ir en una celda.
  - *GridBagLayout*. Similar al anterior pero cada componente puede ocupar varias celdas.

Acciones

- ¿Cómo asignar acciones a eventos producidos sobre la interfaz (e.g. pulsar un botón)?
- Pueden añadirse escuchadores de eventos a los componentes:

```
JButton jbnButton = new JButton("Send Message");
jbnButton.addActionListener(listener);
```

- El objeto *listener* debe implementar la interfaz *java.awt.ActionListener*.
- Cuando el botón sea pulsado, se invocará el método *actionPerformed* del objeto *listener*.
- Dicho método debe implementar la acción que se desea ejecutar cuando se pulsa el botón.

Tutoriales Swing

- http://www.beginner-java-tutorial.com/java-swing-tutorial.html
- http://java.sun.com/docs/books/tutorial/uiswing/