



Laboratorio de Aplicaciones Telemáticas Ingeniería Técnica de Telecomunicación – Telemática

Universidad Carlos III de Madrid
Escuela Politécnica Superior
Leganés, 4 de septiembre de 2009

Duración de la prueba: 2 h 15 min
No se permite el uso de libros ni apuntes.

Cuestión 1 (0,5 puntos)

Indica qué se encuentra habitualmente dentro del directorio WEB-INF (incluyendo sus subdirectorios) de una aplicación Web basada en Java.

Cuestión 2 (0,5 puntos)

Explica por qué, salvo en contadas excepciones, no suele ser correcto guardar datos como atributos de instancia de un *servlet*.

Cuestión 3 (0,5 puntos)

Indica qué API permite a un programa JavaScript acceder al código del documento HTML al cual está asociado, así como modificarlo dinámicamente.

Problema 1 (3,5 puntos)

Se desea implementar algunas funciones relacionadas con la gestión del carro de la compra en el sistema Web de una tienda virtual.

Uno de los componentes el sistema es el *servlet* que añade un producto al carro de la compra, cuya URL en el servidor es `/comprar`. Desde el catálogo de productos de la tienda (una página XHTML con un listado de distintos productos disponibles), cada producto enlaza a este *servlet* mediante un botón de compra. Este botón de compra no es más que una imagen (`/img/comprar.png`) enlazada, mediante el elemento `a`, al *servlet*.

Apartado 1 (1 punto)

Indica cómo harías, desde el catálogo de productos, para comunicar al *servlet* el código del producto cuyo botón ha presionado el usuario para añadirlo al carro de la compra. Dada tu solución, escribe, a modo de ejemplo, el fragmento de código XHTML del botón de compra para un producto con identificador 9999.

Apartado 2 (1,5 puntos)

Escribe el código del *servlet* que añade el producto indicado al carro de la compra. El carro de la compra es volátil, esto es, no se guarda en base de datos, pero debe estar accesible durante la sesión actual de navegación del usuario. Utiliza objetos de la clase `Producto` para representar los productos guardados en el carro. Puedes suponer que esta clase cuenta con un constructor que, dado un identificador (tipo entero) para el producto, inicializa todos sus atributos convenientemente. Al finalizar el procesado, el *servlet* simplemente redirige el control mediante *forward* al recurso con URL `/catalogo`.

Nota: para simplificar la solución, no es necesario añadir código para gestionar el caso de que el identificador del producto no sea un entero o no corresponda con un producto del catálogo.

Apartado 3 (1 punto)

Escribe una página JSP que muestre el contenido del carro de la compra. Debe mostrar una tabla con dos columnas (nombre del producto y precio) y una fila por cada producto. Puedes suponer que los objetos de la clase `Producto` tienen métodos tipo `get` que devuelven tanto el nombre como el precio del producto. Si el carro no tiene productos, debe mostrar simplemente un texto indicándolo, en vez de la tabla.

Problema 2 (5 puntos)

La aplicación swing que se lista a continuación muestra una interfaz gráfica con una tabla de datos:

```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4
5 public class MiTabla extends JPanel {
6     Vector<String> datos, nombreCols;
7     public MiTabla() {
8         super(new GridLayout(1,0));
9
10        nombreCols = new Vector<String>();
11        nombreCols.add("Puesto");
12        nombreCols.add("Sueldo");
13        nombreCols.add("Fecha Fin");
14
15        Vector<OfertaBean> ofertas= new Vector<OfertaBean>();
16        ofertas.add(new OfertaBean(0,0,0,0,"DATOS","NO VALIDOS","", "", "28/02/2009"));
17        //getDatos() obtiene un nuevo vector solo con los campos a mostrar
18        datos= getDatos(ofertas);
19        final JTable tabla = new JTable(datos, nombreCols);
20        tabla.setPreferredScrollableViewportSize(new Dimension(450, 600));
21        tabla.setFillViewportHeight(true);
22        JScrollPane panel = new JScrollPane(tabla);
23        add(panel);
24    }
25
26    private static void createAndShowGUI() {
27
28
29
30
31
32
33
34    }
35
36    public static void main(String[] args) {
37        javax.swing.SwingUtilities.invokeLater(new Runnable() {
38            public void run() {
39                createAndShowGUI();
40            }
41        });
42    }
43 }
```

Vamos a modificar y añadir nuevo código a esta clase para mostrar los datos de ofertas de trabajo de una base de datos remota.

Apartado 1 (1 punto)

¿Cuál es la razón de utilizar el método estático de `SwingUtilities.invokeLater()` en el método `main()`?

Apartado 2 (1 punto)

Proporcione el código del método `createAndShowGUI()` que debe crear un `JFrame` en el que se muestre el `JPanel MiTabla`.

Apartado 3 (0.5 puntos)

Suponiendo que el siguiente código es suficiente para conectarse a la base de datos:

```
java.sql.Connection conn;
try {
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    conn= DriverManager.getConnection
        ("jdbc:mysql://mysql.example.com/BD", "user1", "passwd1");
} catch (SQLException ex) {
    ex.printStackTrace();
    ...
} catch (Exception ex) {
    ex.printStackTrace();
    ...
}
```

¿Dónde considera conveniente poner dicho código? ¿En un método estático de la clase MiTabla, en un método de instancia, o en otra clase/paquete independiente? Motive su respuesta.

Apartado 4 (1.5 puntos)

Suponga una clase Java Bean `OfertaBean` que modela las ofertas de trabajo, y que incluye como propiedades cuatro enteros (`oid`, `eid`, `uid`, `pid`), cuatro cadenas (`descripcion`, `requisitos`, `sueldo`, `jobref`) y la fecha de fin de validez. Suponga un constructor de `OfertaBean` que tome como parámetros todas sus propiedades. Suponga el siguiente método `BD.listaOfertas()` que utilizando la propiedad `Connection cx` (una conexión a la base de datos abierta en la inicialización de la clase), devuelve un vector de `OfertaBean` con todas las ofertas de la base de datos.

```
Vector<OfertaBean> listaOfertas() throws SQLException{
    Statement s = cx.createStatement();
    ResultSet rs= s.executeQuery("Select * from Ofertas;");
    OfertaBean oferta;
    while(rs.next()){
        oferta=new OfertaBean(rs.getInt(1), rs.getInt(2),
                               rs.getInt(3), rs.getInt(4),
                               rs.getString(5), rs.getString(6),
                               rs.getString(7), rs.getString(8),
                               rs.getDate(9));
        vector.add(oferta);
    }
    return vector;
}
```

Se desea modificar la clase `MiTabla` para incluir un botón (`JButton actualizar`) que cuando se pulse refresca la tabla con los valores contenidos en ese momento en la base de datos. El tratamiento del evento lo realizaría el código siguiente:

```
TableModel modelo = tabla.getModel();
Vector<OfertasBean> vOfertas = BD.listaOfertas();
modelo.getDataVector().clear();
modelo.setDataVector(getDatos(vOfertas), nombreCols);
```

Indique qué interfaz debe implementar para recoger eventos del botón (0.5 puntos). Proporcione el código que inicializa el botón (en el contenedor por defecto) y el método que trata el evento. (1 punto)

Apartado 5 (1 punto)

Una vez modificada la clase `MiTabla`, se le añade otro botón para mostrar una ventana emergente, con los datos de la columna seleccionada. El código que trata el evento es el siguiente:

```
JFrame frame = new JFrame("Datos seleccionados");
StringBuffer sel= new StringBuffer();
for (int c : table.getSelectedRows()) {
    sel.append(datos.elementAt(c)+' '\t');
}
JLabel jlabel = new JLabel("Oferta seleccionada:"+sel);
frame.getContentPane().add(jlabel, BorderLayout.CENTER);
frame.pack();
frame.setVisible(true);
```

Después de probarlo varias veces, y cerciorarse de que la funcionalidad es correcta, detecta que la aplicación está consumiendo mucha memoria, y que está se incrementa cada vez que se muestra la ventana emergente. Explique el problema y cómo lo solucionaría.