



Laboratorio de Aplicaciones Telemáticas Ingeniería Técnica de Telecomunicación – Telemática

Escuela Politécnica Superior. Universidad Carlos III de Madrid.
Leganés, a 3 de Febrero de 2003.
Duración de la prueba: 2h 30min

Cuestión 1 (1 pto.)

Existen múltiples tecnologías cliente-servidor. En cada una de ellas la ejecución del código que lo compone está distribuido entre el cliente (navegador Web) y el servidor.

1. Clasifica las siguientes tecnologías según el lugar donde ejecuta su código o donde están las rutinas para tratarlo (*cliente / servidor / cliente y servidor*):
 - a) XHTML
 - b) HTTP
 - c) Servlets
 - d) JSPs
 - e) Javascript
 - f) Scriptlets
 - g) Java Beans
 - h) Applets
 - i) Plugin

La empresa ACME S.A. ha desarrollado un novedoso y complejo sistema de codificación de señales de audio y vídeo, que reduce espectacularmente su tasa de transmisión, aunque haciendo un uso bastante elevado de recursos de CPU y memoria. El desarrollo es aprovechado para iniciar un negocio de transmisión de vídeo bajo demanda, y se ha programado un decodificador, en lenguaje C, con versiones para las arquitecturas más habituales. Para facilitar el acceso, pretenden que el usuario pueda comprar y ver los vídeos desde un navegador web. Por imposición del departamento de marketing, el plazo disponible para esta tarea es muy limitado.

2. De acuerdo con las restricciones anteriores, razona qué tecnología de programación web, de entre las mencionadas anteriormente, resultaría más adecuada para desarrollar el decodificador y su interfaz integrada en el navegador. Indica claramente el porqué.

Solución:

1.
 - a) cliente
 - b) cliente y servidor
 - c) servidor
 - d) servidor
 - e) cliente
 - f) servidor
 - g) servidor
 - h) cliente
 - i) cliente
2. *Plugin*. Obviamente, el decodificador debe ejecutarse en el cliente. Parece más adecuada la utilización de plugins en lugar de applets porque existe un desarrollo previo en lenguaje C, portado a las plataformas más habituales, lo cual permitirá reutilizar el código ya existente. Por otra parte, la utilización de recursos será más eficiente que si se utilizasen applets.

Cuestión 2 (2 ptos.)

La división de aplicaciones móviles de la empresa ACME S.A. desea desarrollar un juego sencillo de aventura gráfica. El jugador maneja a un personaje con varias propiedades: puntos de vida (valor entre 0 y 100), ubicación (número de escenario) y bolsa de objetos (puede contener pócimas, mapas, llaves, armas, etc.) En función del contenido de la bolsa de objetos y de su ubicación, al jugador se le presentan o no determinadas opciones, imágenes, etc.

La aplicación debe cumplir los siguientes requisitos:

- El jugador interactúa con un navegador con soporte para XHTML Basic y cookies.
- La aplicación se ejecuta siempre a través de la URL `http://acme.com/aventura`.
- No se almacena ninguna información de sesión en el servidor, y ésta se gestiona mediante cookies.

En un momento dado el personaje está en el escenario 5 con 49 puntos de vida. Tiene en su bolsa tres pócimas (verde, azul y ámbar) y una espada. Entre otros, el documento representado en el móvil contiene el siguiente formulario:

```
<form action='aventura' method='get'
  enctype='application/x-www-form-urlencoded'>
  <p>
    Utilizar pócima (necesita un conjuro).
  </p>
  <p>
    Conjuro: <input type='text' name='conjuro' id='conjuro' />
  </p>
  <p>
    Pócima:
    <input type='checkbox' name='tipo' value='verde' />verde
    <input type='checkbox' name='tipo' value='azul' />roja
    <input type='checkbox' name='tipo' value='ámbar' />azul
  </p>
  <p>
    <input type='submit' value='Usar pócimas' />
  </p>
</form>
```

El usuario selecciona las pócimas verde y ámbar, conjuro “abracadabra” y envía el formulario.

1. Diseña un formato de cookies para esta aplicación. Descríbelo muy brevemente.
2. Escribe la petición HTTP 1.1 generada por el navegador y la respuesta de éste. Escribe en ambos casos la primera línea, las cabeceras más relevantes, y el cuerpo del mensaje (por simplicidad, pon “cuerpo” como valor del cuerpo de la respuesta). Se proporcionan pistas sobre el formato de un mensaje HTTP al final de la cuestión.
3. El formulario anterior se envía a una aplicación CGI, que funciona en Linux con un servidor Apache. Indica cómo el servidor le comunica a la CGI las cookies enviadas por el cliente.
4. Si desde el formulario hubiese que subir un fichero al servidor, ¿qué valores de método y codificación utilizarías? Razona la respuesta.

Nota:

- El formato de la línea de solicitud es “método uri HTTP/1.1”
- El formato de la línea de estado de una respuesta es “HTTP/1.1 estado frase-explicativa”. El estado OK tiene código 200.
- Las cabeceras HTTP relevantes para este ejercicio son: Content-Type, Content-Length, Cookie, Set-Cookie y Host.

- Puedes especificar varias cookies siguiendo el formato del siguiente ejemplo: “nombre1=valor1,nombre2=valor2,...”

Solución:

1. Todas las propiedades del personaje deben formar parte de la sesión. Un posible diseño es la utilización de tres cookies distintas:
 - a) vida
 - b) ubicacion
 - c) bolsa (su valor será una lista de objetos separados por “/”)
2. Los mensajes de petición y respuesta serán, respectivamente los siguientes, si se ignoran las cabeceras no mencionadas en el enunciado:

```
GET /aventura?conjuro=abracadabra&tipo=verde&tipo=%C3%A1mbar HTTP/1.1
Host: acme.com
Cookie: vida=49,ubicacion=5,bolsa=pverde/pazul/p%C3%A1mbar/espada
```

```
HTTP/1.1 200 OK
Set-Cookie: vida=79,ubicacion=5,bolsa=pazul/espada
Content-Length: 8
Content-Type: text/html
```

(cuerpo)

3. A través de la meta-variable HTTP_COOKIE.
4. Método POST y codificación *multipart/form-data* porque la codificación de los datos es más eficiente, y no se imponen restricciones a su tamaño.

Cuestión 3 (1 pto.)

Supón que se define una extensión a HTTP con tres nuevos métodos para facilitar el soporte de transacciones (TSTART, TCOMMIT y TCANCEL). Se desarrolla un nuevo servlet genérico con soporte para estos métodos:

```
public class TransactionServlet extends HttpServlet ...
```

Gracias a estos métodos, se pueden escribir servlets como el siguiente:

```
public class MiServlet extends TransactionServlet {
    ...
    public doTStart(HttpServletRequest req, HttpServletResponse res) {...}
    public doTCommit(HttpServletRequest req, HttpServletResponse res) {...}
    public doTCancel(HttpServletRequest req, HttpServletResponse res) {...}
}
```

1. ¿Qué método es necesario reescribir en la clase *TransactionServlet* para que, cuando llegue una petición, se ejecute el método adecuado en el servlet *MiServlet*? Describe qué modificaciones habría que hacer con respecto al mismo método de *HttpServlet*.

Bajo ciertas circunstancias, la operación TCOMMIT no puede ser ejecutada con éxito, y es necesario cancelar la transacción (método TCANCEL). Para ello, desde el método *doTCommit* se puede invocar al método *TCancel*, pero es necesario comunicarle a éste un código explicativo del error encontrado. Los diseñadores han optado por introducir este código como atributo de instancia de *MiServlet*, ya que de esta forma será visible desde ambos métodos.

2. ¿Es correcto el diseño anterior? Razona clara y brevemente por qué. Si no es correcto, propón un mecanismo alternativo para hacerlo.

Solución:

1. Es necesario reescribir el método *service()*. Este método es el que analiza el método HTTP de la petición e invoca al método *do...* correspondiente. Por tanto, habrá añadir tres nuevas reglas correspondientes a los métodos TSTART, TCOMMIT y TCANCEL.
2. No es correcto. Dado que una única instancia del servlet atiende a todas las peticiones concurrentemente, este atributo será el mismo para todas ellas, lo cual provoca interferencia de unas peticiones en otras. Una solución es añadirlo como atributo de la petición con el método *HttpRequest.setAttribute()*, antes de invocar a *doTCancel*.

Cuestión 4 (1 pto.)

Responde las siguientes preguntas sobre el uso de Servlets:

1. Un determinado servlet devuelve un fichero de texto en el cuerpo de la respuesta HTTP (tipo MIME *text/plain*) con el texto "Eureka". Describe clara y brevemente la secuencia de acciones que debe realizar el servlet para establecer el cuerpo y el tipo MIME de la respuesta.
2. En un servlet se toma la decisión de que una determinada petición debe ser atendida por otro servlet más adecuado. Indica brevemente las diferencias que hay entre utilizar *sendRedirect()* y *forward()* para ello.

Solución:

1. Si el objeto *HttpServletResponse* se referencia con *res*, se deben seguir los siguientes pasos:
 - a) Establecer el tipo de contenido:
res.setContentType("text/plain")
 - b) Obtener un objeto *PrintWriter* para escribir el cuerpo:
out = res.getWriter()
 - c) Escribir el cuerpo:
out.println("Eureka")
2. La principal diferencia está en que *sendRedirect* provoca el envío de una respuesta HTTP de redirección al cliente, para que éste realice la petición a la nueva URL, mientras que *forward* transfiere la petición a otro recurso del mismo servidor, sin interacción con el cliente.

Cuestión 5 (1 pto.)

En la práctica de la AgendaWeb la aplicación debía mantener cierta información además de la almacenada en la Base de Datos. Dicha información debía mantenerse en la forma de JavaBeans para permitir su intercambio entre los elementos de la aplicación. Indique que información contenían los JavaBeans y el contexto (aplicación/sesión/petición) de los mismos que empleaba su aplicación.

Solución:

La conexión permanente con la base de datos se guardaba en un *JavaBean* en el contexto de aplicación de forma que todas las consultas compartían la misma conexión.

Los datos del usuario que se utilizaban tanto para validar su acceso (login y password) como para modificar sus datos personales estaban en un *JavaBean* en el contexto de sesión.

Normalmente para resolver cada petición era necesario intercambiar con la página JSP un *JavaBean* que contuviese los eventos seleccionados. Dicho *JavaBean* debía tener el contexto de petición.

Cuestión 6 (1 pto.)

Responda a las siguientes cuestiones referidas al uso de JSPs y JavaBeans:

1. Describa brevemente las tres operaciones definidas en la especificación de las JSP con las que es posible definir y manipular JavaBeans y sus atributos más habituales.

2. Diseñe un sencillo JavaBean con una sólo propiedad de lectura/escritura denominada `Quantity` y que tiene como valor por defecto 1.

Solución:

1. La primera etiqueta (`useBean`) sirve para indicar que se desea utilizar en la página un JavaBean. Es necesario especificar el nombre (`id`), la clase (`class`) y contexto del mismo (`scope`). La segunda etiqueta (`getProperty`) sirve para acceder a una propiedad del mismo. Se debe especificar el nombre (`name`) del JavaBean y de la propiedad (`property`). La tercera etiqueta (`setProperty`) sirve para asignar un valor a una propiedad del JavaBean. Se especifica el nombre (`name`) del JavaBean, el nombre de la propiedad (`property`) y su nuevo valor (`value`).

```
2. public class QuantityBean
{
    private int quantity = 1;

    public QuantityBean()
    {
        // Does nothing
    }

    public int getQuantity()
    {
        return quantity;
    }

    public void setQuantity ( int q )
    {
        quantity = q;
    }
}
```

Cuestión 7 (1 pto.)

¿Existe alguna forma de que un diseñador Web pueda configurar el comportamiento o la apariencia de un applet sin necesidad de modificar el código fuente del mismo? Si ha utilizado esta técnica en la práctica de Applets, describala brevemente e indique qué información era configurable en dicha práctica.

Solución:

Los Applets pueden ser configurados dentro de la página web en la que son definidos mediante etiquetas (`<param>`). Con esas etiquetas es posible asociar un nombre a un valor. Desde el Applet se puede utilizar un método (`getParameter()`) para obtener el valor asociado al nombre pedido. En la práctica se pedía que el nombre del segundo applet fuera configurable por parámetro así como los valores iniciales de los campos de los formularios.

Cuestión 8 (1 pto.)

Responda a las siguientes cuestiones referidas a la práctica de AgendaSwing:

1. Describa brevemente el método `actionPerformed()` de la interfaz `ActionListener` y bajo qué circunstancias se puede ejecutar dicho método en la práctica.
2. ¿Qué patrón de diseño emplea el método citado en el apartado anterior? Indique el nombre del patrón y cómo debe implementarse en Java.

Solución:

1. El método `actionPerformed()` puede recibir eventos (`ActionEvent`) de botones, barras de herramientas y de menú cuando alguno de sus elementos se presiona. En el caso de un campo de texto se llama cuando el usuario pulsa Return dentro del mismo. Los manejadores de eventos debe implementarlo para tratar los eventos generados por dichas acciones del usuario.

2. El patrón de diseño Comando se puede implementar en Java mediante el interfaz `Action` aunque normalmente se extiende la clase `AbstractAction`. Esta clase implementa el interfaz `ActionListener` que es donde se define el método del aparatado anterior.

Cuestión 9 (1 pto.)

Describe brevemente los modelos de la clase `JTable` que ha empleado en la práctica de `AgendaSwing`. Indique su propósito general, si ha sido necesario extender el modelo por defecto y al menos una operación de cada uno de ellos.

Solución:

La clase del modelo de datos (`AbstractTableModel`) debe extenderse para que la vista de `JTable` sea capaz de representar los eventos. Hay varios métodos que deben implementarse, los más importantes son: devolver el número de filas (`getRowCount()`) y de columnas (`getColumnCount()`) y el valor de una celda determinada (`getValueAt()`).

El modelo de selección (`ListSelectionModel`) puede utilizarse directamente y mantiene los eventos seleccionados actualmente. Ejemplos de métodos: Ver si una fila está seleccionada (`isSelectedIndex()`), si hay algún elemento seleccionado (`isSelectionEmpty()`) y obtener el índice del menor y mayor elemento seleccionado (`get(Min/Max)SelectionIndex()`).