

Nombre: \_\_\_\_\_

Duración: 2 horas, 30 minutos

Puntuación 9 puntos (+ 1 punto para la presentación de tecnologías Web)

No se permite el uso de libros ni apuntes.

**Conteste cada pregunta en una hoja separada.**

### Ejercicio 1. (2 puntos)

(a) **(1 punto)** Proporcione un modelo de datos conceptual de la base de datos del Ejercicio 2 en forma de un diagrama de clases UML, donde su modelo tiene que tomar en cuenta las siguientes consideraciones:

- Se considera que un libro está compuesto de ejemplares.
- Cada libro se clasifica de una única forma.
- Las clasificaciones forman una estructura de árbol que puede modelarse mediante una asociación es\_hijo\_de (con roles y multiplicidades adecuados). La restricción adicional requerida para restringir el modelo a un árbol – que exista exactamente un camino entre dos clasificaciones cualesquiera a través de instancias de la asociación es\_hijo\_de (no necesita entender por qué esta restricción es suficiente) – puede añadirse simplemente como un comentario en lenguaje natural.
- Hay tres operaciones que se corresponden con sacar, renovar y devolver un ejemplar.
- Algunos de los atributos utilizados en la base de datos no deberían aparecer en el modelo conceptual; si los dos atributos de recuento deberían o no estar entre éstos es debatible, aquí se pide que tales atributos de recuento se incluyan.
- Una de las clases se modela mejor como una clase asociación.

No necesita incluir ninguna información de tipado.

(b) **(1 punto)** Proporcione un diagrama de estados UML para describir el comportamiento de la clase EJEMPLAR, donde su modelo debe tomar en cuenta las siguientes consideraciones:

- Hay tres transiciones, el evento de disparo de cada una de ellas es la invocación de uno de los tres métodos mencionados en la sección anterior.
- Las acciones que han de realizarse conciernen la gestión de Recuento\_prestamos del correspondiente usuario y la gestión de Fecha\_salida y Fecha\_devolucion del correspondiente préstamo.
- Su modelo debe incluir la información de que un usuario no puede sacar más de 6 libros a la vez y no puede renovar un préstamo si ya ha expirado.

Debe usar la sintaxis correcta de las etiquetas de las transiciones de estado UML, pero es libre de usar cualquier sintaxis para los diferentes elementos de estas etiquetas.

### Ejercicio 2. (1,5 puntos)

Una base de datos de una biblioteca contiene las siguientes tablas:

LIBROS

ISBN	Titulo	Autor	Ref_Dewey	Info_editorial	Anio_publicacion	Recuento_ejemplares
------	--------	-------	-----------	----------------	------------------	---------------------



Nombre: \_\_\_\_\_

- (c) **(0,4 puntos)** El usuario con ID\_usuario "007" devuelve el ejemplar de un libro con ID\_ejemplar "T42.4Me.4U". Modifique la base de datos para tomar esto en cuenta usando dos comandos SQL. No necesita preocuparse de los aspectos transaccionales de los dos comandos.

### Ejercicio 3 (2 puntos)

- (a) **(1 punto)** Estudie el siguiente código de un JSP y responda a las preguntas que figuran a continuación [Ayuda: el RFC 2396 titulado *Uniform Resource Identifiers (URI): Generic Syntax* indica que "una referencia URI que no contiene un URI es una referencia al documento actual"].

```
1. <%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
2. <%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>
3.
4. <h1>Video Rental Service Registration</h1>
5.
6. <c:if test="${param.submitted}">
7.
8.     <c:if test="${empty param.name}" var="noName" />
9.     <c:if test="${empty param.email}" var="noEmail" />
10.    <c:if test="${empty param.age}" var="noAge" />
11.
12.    <c:catch var="ageError">
13.        <fmt:parseNumber var="parsedAge" value="${param.age}" />
14.        <c:if test="${parsedAge < 16}" var="youngAge" />
15.    </c:catch>
16.    <c:if test="${not empty ageError}" var="badAge" />
17.
18.    <c:if
19.        test="${not (noName or noEmail or noAge or badAge or youngAge)}">
20.        <c:set value="${param.name}" var="name" scope="request"/>
21.        <c:set value="${param.email}" var="email" scope="request"/>
22.        <c:set value="${param.age}" var="age" scope="request"/>
23.        <jsp:forward page="registrationFormHandler.jsp" />
24.    </c:if>
25.
26. </c:if>
27.
28. <form method="post" action="">
29.     <p>
30.         Please sign up for our video rental service.
31.     </p>
32.
33.     <p>
34.         Enter your name:
35.         <input type="text" name="name"
36.             value="<c:out value="${param.name}"/>" />
37.     <br />
38.     <c:if test="${noName}">
39.         <small><font color="red">
```

Nombre: \_\_\_\_\_

```
40.     Note: you must enter a name
41.     </font></small>
42. </c:if>
43. </p>
44.
45. <p>
46. Enter your email address:
47. <input type="text" name="email"
48.     value="<c:out value="{param.email}"/>" />
49. <br />
50. <c:if test="{noEmail}">
51.     <small><font color="red">
52.     Note: you must enter an email address
53.     </font></small>
54. </c:if>
55. </p>
56.
57. <p>
58. Enter your age:
59. <input type="text" name="age" size="3"
60.     value="<c:out value="{param.age}"/>" />
61. <br />
62. <c:choose>
63.     <c:when test="{noAge}">
64.         <small><font color="red">
65.         You must enter your age
66.         </font></small>
67.     </c:when>
68.     <c:when test="{badAge}">
69.         <small><font color="red">
70.         The format of the age you entered was not correct
71.         </font></small>
72.     </c:when>
73.     <c:when test="{youngAge}">
74.         <small><font color="red">
75.         I'm sorry, you are too young to rent videos without
76.         your parents' approval
77.         </font></small>
78.     </c:when>
79. </c:choose>
80. </p>
81.
82. <input type="submit" value="Register" name="submitted"/>
83.
84. </form>
```

- (i) ¿Cuál es el propósito del parámetro de la petición (*request parameter*) submitted?
- (ii) ¿Por qué razón se definen tres variables de ámbito de la petición (*request scope*) antes de hacer el *forward* a otra página (líneas 20-22), en lugar de dejar que la página de destino obtenga los valores directamente a partir de los correspondientes parámetros de la petición (*request parameters*)?

Nombre: \_\_\_\_\_

(iii) ¿Qué hace el código de las líneas 36, 48 y 60? ¿Cuál es el efecto de que el usuario no proporcionase un valor para el parámetro cuando el formulario se envió previamente?

(iv) Describa brevemente lo que hace globalmente la página.

- (b) **(1 punto)** A continuación se presenta una versión ligeramente editada del código de un servlet, contenida en un fichero denominado `Controller.java`, que constituye la parte central de una aplicación Web que permite a los usuarios registrados hacer login y enviar a continuación datos para su almacén en la base de datos. Estudie el código y responda a las preguntas que aparecen a continuación.

El código asume la existencia de páginas JSP `login.jsp`, `dataInput.jsp` y `goodbye.jsp`. Adicionalmente asume que:

- el valor del atributo `method` de la etiqueta `FORM` de la página `login.jsp` es `POST` y el formulario tiene dos variables, denominadas `UID` y `PWD`, y una variable oculta (*hidden*) `referrer` con valor `login`.
- el valor del atributo `method` de la etiqueta `FORM` de la página `dataInput.jsp` es `POST` y el formulario tiene muchas variables, los detalles de las cuales no se necesitan conocer en este ejercicio, y una variable oculta `referrer` con valor `dataInput`.

Note que los atributos `loginError` y `dataInputError` se usan en las páginas `login.jsp` y `dataInput.jsp` respectivamente, con el fin de saber si el usuario está accediendo a la página por primera vez o si se ha devuelto el control a la página debido a un error.

```
import javax.servlet.*;
import java.sql.* ;
import javax.sql.DataSource;
import javax.naming.Context;
import javax.naming.InitialContext;
import java.io.IOException;

public class Controller extends HttpServlet {

    DataSource ds;
    HttpSession session;

    public void init() {
        /* initialize the servlet; use JNDI to look up a DataSource          */
    }

    void doPost (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        session = object1.method1 ();
        session.removeAttribute("loginError");
        session.removeAttribute("dataInputError");

        String referringPage = object1.method2("referrer");

        if (referringPage != null) {

            if (referringPage.equals("login")) {
                String uid = req.getParameter("UID");
                String pwd = req.getParameter("PWD");
                if (authenticate(uid, pwd)) loginSuccess(req, res);
                else loginFailure(req, res);
            }
            else if (referringPage.equals("dataInput")) {
                String validUser = (String) object2.method1("uid");
                if (validUser != null) {
                    if (storeData(req)) dataInputSuccess(req, res);
                    else dataInputFailure(req, res);
                }
            }
        }
    }
}
```

Nombre: \_\_\_\_\_

```
        }
        else loginFailure(req, res);
    }
    /* should be an exception: unrecognised value of "referrer" variable in request */
    else loginFailure(req, res);

}
/* should be an exception: no "referrer" variable in the request */
else loginFailure(req, res);

}

private void gotoPage(String page, HttpServletRequest req, HttpServletResponse res) {
    throws ServletException, IOException {
    RequestDispatcher dispatcher = object1.method3(page);
    if (dispatcher != null) {
        dispatcher.forward(req, res);
    }
}

private void loginSuccess(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    object2.method2("uid", uid);
    gotoPage("/WEB-INF/jsp/dataInput.jsp", req, res);
}

private void loginFailure(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    object2.method2("loginError", "y");
    gotoPage("/login.jsp", req, res);
}

private void dataInputSuccess(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    object2.method3("uid");
    gotoPage("/WEB-INF/jsp/goodbye.jsp", req, res);
}

private void dataInputFailure(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    object2.method2("dataInputError", "y");
    gotoPage("/WEB-INF/jsp/dataInput.jsp", req, res);
}

private boolean authenticate(String uid, String pwd) {
    /* Connect to the database and check if the pair (uid, pwd) is registered */
    /* - return true if this is the case (i.e the user is valid) */
    /* - return false if it is not, or if a database connection problem occurred */
}

private boolean storeData(HttpServletRequest req) {
    /* Connect to the database and store the input data */
    /* - return true if data was stored OK */
    /* - return false if a database connection problem occurred */
}

public void destroy() {}

}
```

- (i) Explique brevemente el papel que desempeñan los dos parámetros de los métodos doPost y doGet en el caso general, esto es, los objetos de las clases HttpServletRequest y HttpServletResponse (en este ejemplo se denominan req y res respectivamente).

Nombre: \_\_\_\_\_

- (ii) Proporcione los nombres de los dos objetos y seis nombres de métodos para las llamadas a métodos que se han reemplazado por las cadenas de caracteres *objetoM*, para  $M = 1,2$  y *metodoN*, para  $N = 1,2,3$ .
- (iii) Juzgando a partir de la información aquí disponible, ¿sigue esta aplicación Web el patrón arquitectural MVC?; proporcione una breve justificación de su respuesta.

### Exercise 4 (2 points)

#### (a) (0,5 points)

- (i) **(0,1 points)** Diga qué es una transacción.
- (ii) **(0,4 points)** Explique las clases de transacciones que soporta el contenedor EJB, identificando sus diferencias

#### (b) (1,5 points)

Mire el siguiente código (conforme a la especificación EJB 2.1) de las interfaces remotas de una clase *bean*, y luego responda a las preguntas que aparecen a continuación. Note que ciertas partes del código se han reemplazado por XXX.

##### Remote Interface

```
package examen;

public interface BookRemote extends XXX {

    public String getAuthor() throws XXX;
    public String getTitle() throws XXX;
    public String getCategory() throws XXX;
    public String getPublisher() throws XXX;
    public double getPrice() throws XXX;

}
```

##### Remote local interface

```
package examen;

import javax.ejb.CreateException;
import javax.ejb.DuplicateKeyException;
import javax.ejb.FinderException;

public interface BookHomeRemote extends XXX {

    public BookRemote create(String id, String title, String author,
        double price) throws XXX, DuplicateKeyException, CreateException;

    public BookRemote create(String id, String title, String author,
        String category, String publisher, double price) throws XXX,
        DuplicateKeyException, CreateException;

    public BookRemote findByPrimaryKey(String id)
        throws XXX, FinderException;

    public BookRemote findByTitle(String title)
        throws XXX, FinderException;

}
```

Nombre: \_\_\_\_\_

- (i) **(0,5 points)** Reemplace las partes del código XXX por las clases correspondientes.
- (ii) **(1 point)** Escriba la clase del bean de entidad asociado a dichas interfaces, cuya persistencia es gestionada por contenedor.

Nota: la clase debe incluir tanto los métodos del ciclo de vida (incluidos los de "callback") como los de negocio. Debe haber siete métodos de "callback": dos relacionados con la activación, dos con la sincronización con la base de datos, dos con la gestión del contexto y uno con el borrado del bean.

Nombre: \_\_\_\_\_

### **Ejercicio 5. (1,5 puntos)**

#### **Instrucciones para el test sobre las tecnologías Web:**

- Para cada pregunta hay una y sólo una respuesta correcta
- Marque la respuesta correcta rodeando el número de la respuesta con un círculo
- En caso de que se equivoque en una pregunta, tache toda la columna de las letras de las respuestas de la pregunta y escriba una nueva columna de letras al lado.
- Las respuestas correctas valen 1 punto, las incorrectas -0.33 puntos y las que no se hayan contestado valen 0.

*< Aquí había 15 preguntas de tipo test sobre las información impartida en las presentaciones >*