: ¿Qué es el deployment descriptor?

Un deployment descriptor es un documento XML con extensión .xml que describe la configuración de deployment de una aplicación, módulo o componente. En el caso de una aplicación Java Web, el archivo deployment descriptor se llama web.xml. Dicho deployment descriptor forma parte de la especificación JavaEE, y por lo tanto es reconocido por cualquier implementación de servidor JavaEE. También pueden existir un deployments descriptors específicos de la implementación de servidor JavaEE donde se esté desplegando la aplicación.

Pregunta 2 (5 puntos): ¿Qué operaciones define la interfaz Filter?

Un filtro (filter) es un objeto que puede transformar el encabezado (header) y/o el contenido (content) tanto de pedidos (request) como de respuestas (response). Un filtro difiere de un componente web (ej: servlet) ya que generalmente no genera una respuesta, sino que filtra la respuesta de un componente web. Un filtro provee funcionalidades que pueden ser adjuntadas (attach) a cualquier componente web. Por tanto, un filtro debe ser lo más independiente posible de los demás componentes web, de forma de poder filtrar varios tipos de componentes

¬ Las funcionalidades principales de un filtro son: • Inspeccionar un pedido (request) y actuar en consecuencia. • Bloquear un pedido/respuesta. • Modificar el encabezado y/o contenido de un pedido o de una respuesta mediante el uso de una versión personalizada del pedido o respuesta original. • Interactuar con recursos externos. ¬ Ejemplos: autenticación, logueo, conversión de imágenes, compresión de datos, encriptación, etc.Analista de Sistemas - Escuela de Sistemas - BIOS 29 Filtros (3) ¬ Los filtros siguen la arquitectura Pipes and Filters que se verá en el curso Ingeniería de Software. ¬ Por tanto, éstos pueden combinarse en lo que se denomina “cadena de filtros” (filter chain). ¬ Cada filtro se aplicará sobre la entrada / salida de un componente web y se invocarán entre sí siguiendo la cadena definida. ¬ La cadena es definida en el deployment descriptor y es instanciada cuando el contenedor carga el componente web.Analista de Sistemas - Escuela de Sistemas - BIOS 30 Filtros (4) ¬ Los pasos para utilizar un filtro son: 1. Codificar el filtro (clase Java). 2. Codificar los pedidos (request) y respuestas (response) personalizados (clases Java). 3. Definir la cadena de filtros para cada componente web que se desea afectar (en el deployment descriptor). Esto se denomina filter mapping. ¬ Las interfaces involucradas son: • javax.servlet.Filter • javax.servlet.FilterChain • javax.servlet.FilterConfigAnalista de Sistemas - Escuela de Sistemas - BIOS 31 Filtros (5) ¬ Para codificar un filtro se debe implementar la interfaz Filter, la cual posee 3 operaciones: 1. void init(FilterConfig config) ◊ se invoca al instanciar el filtro y permite acceder a parámetros de inicialización. 2. void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) ◊ se invoca para realizar el filtrado (para hacer lo que tiene que hacer) y luego invocar al próximo filtro (sin saber quien es). 3. void destroy() ◊ se invoca al momento de destruir la instancia del filtro

Pregunta 3 (5 puntos): ¿Qué es un JavaBean?

¬ Si bien los JavaBeans no son parte de la especificación JavaEE, su uso dentro de la arquitectura JavaEE amerita su introducción. ¬ Un JavaBean es una clase Java que sigue cierto estándar (ej: contener un constructor por defecto, atributos privados y getters/setters públicos). ¬ Los JavaBeans típicamente se utilizan para el flujo de datos entre componentes de una aplicación JavaEE. ¬ Incluso pueden ser utilizados declarativamente (ej: desde una página JSP). ¬ No confundir con los Enterprise Java Beans (EJB

Pregunta 4 (5 puntos): ¿Para qué se utiliza la directiva @taglib?

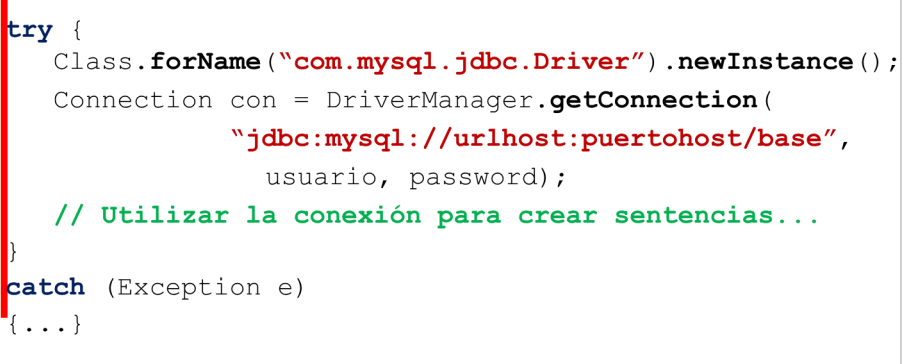
Las librerías de tags (**taglibs**) son conjuntos de etiquetas HTML personalizadas que permiten encapsular determinadas acciones, mediante un código **Java** subyacente. ... Se utilizan así los tags mitag y otrotag.

Pregunta 1 (5 puntos): ¿Para qué sirve en Java la palabra throws (con una S al final)?

La palabra reservada o cláusula **throws** permite lanzar un método. Por tanto tiene que ir declarada en el método

Pregunta 2 (5 puntos): ¿Qué pasos deben seguirse para trabajar con transacciones mediante código JDBC?

¬ Pasos a seguir: 1. Cargar el driver adecuado. 2. Crear una conexión que utilice ese driver.Analista de Sistemas - Escuela de Sistemas - BIOS 11 Crear una Conexión JDBC (3) A. Se cargó el driver que va a ser utilizado, en este caso el Connector/J que sirve para MySQL y se creó una conexión. B. Notar que se invocó a newInstance() aunque no es necesario en todos los casos (depende del fabricante del driver). C. Es necesario capturar las posibles excepciones que puedan ocurrir, tanto al cargar el driver (ej: que éste no se encuentre) como al obtener la conexión



Pregunta 3 (5 puntos): ¿Para qué se utiliza el atributo method de la etiqueta

Principio del formulario

de HTML, y qué ocurre si no se utiliza?

method: indica el método de envío de los datos, que puede ser alguno de estos dos: 1. get: Es el método predeterminado. Los datos (parámetros del pedido) viajan al servidor en el “query string”, esto es, agregados a la url del action seguida de un signo de interrogación (?) y a continuación los nombres de los parámetros con sus respectivos valores indicados mediante el signo de igual (=). Cada parámetro se separa del siguente mediante un signo de ampersand (&). 2. post: los datos viajan al servidor codificados de manera similar (sin la url del action ni el signo de interrogación) pero dentro del cuerpo del pedido, por lo que no se visualizan en la barra de direcciones del navegador

Pregunta 4 (5 puntos): ¿Cómo se declara el tiempo de expiración de la sesión en el archivo web.xml ?

<session-config>

<session-timeout>

30

</session-timeout>

</session-config>

Pregunta 5 (5 puntos): ¿Qué diferencias existen entre un request de tipo GET y un request de tipo POST?

**La diferencia** entre los métodos **get** y **post** radica en la forma de enviar los datos a la página cuando se pulsa el botón “Enviar”. Mientras **que** el método **GET** envía los datos usando la URL, el método **POST** los envía de forma **que** no podemos verlos (en un segundo plano u "ocultos" al usuario).

Pregunta 6 (5 puntos): Describa brevemente los 3 tipos de scriptlets vistos en el curso.

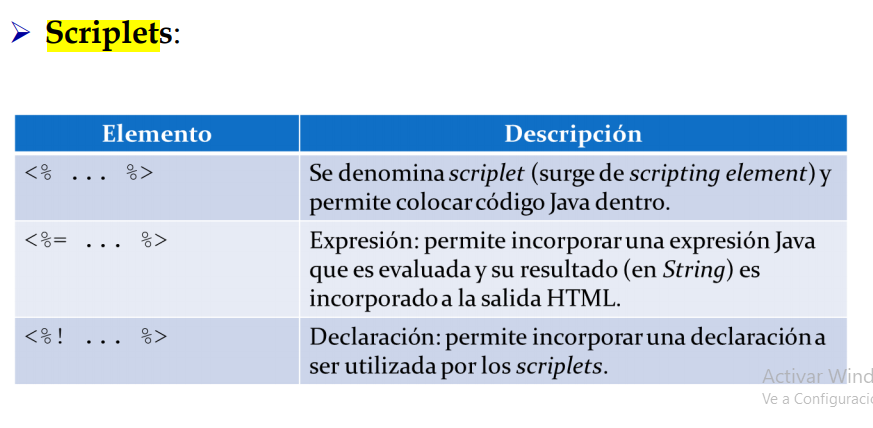
Existen 3 tipos de elementos JSP: • Directivas (directives): especifican información acerca de la página que se mantiene igual a lo largo de todos los pedidos. • Acciones (actions): permite realizar alguna acción basada en información al momento de hacer el pedido.

• Scriplets (scripting): permiten incorporar código Java dentro de una página JSP.

1 scriplets a secas

2 expresion

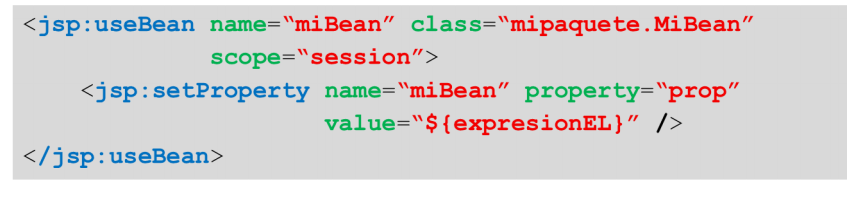
3 declaracion



Pregunta 7 (5 puntos): ¿Qué son las expresiones EL y para qué sirven?

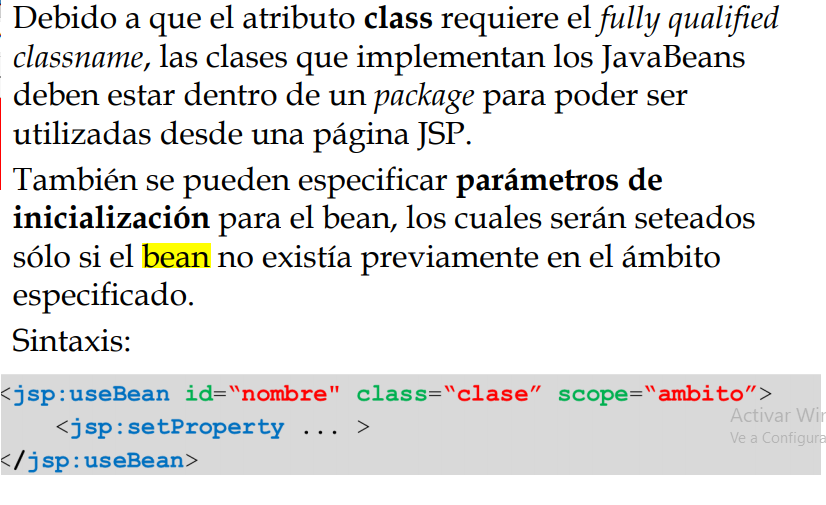
¬ Desde la versión JSP 2.0 existe un lenguaje de expresiones denominado EL (Expression Language)

Dicho lenguaje permite a los autores de páginas JSP (particularmente a quienes no son programadores) escribir expresiones relativamente sencillas para leer valores de un JavaBean dinámicamente y/o establecer valores de atributos de acciones basados en datos recolectados en tiempo de ejecución. ¬ Estas expresiones permiten potenciar el poder de las acciones JSP, ya sean las acciones estándar o las incluidas en JSTL (Java Standard Tag Library).Analista de Sistemas - Escuela de Sistemas - BIOS 9 Expresiones EL (2) ¬ Dicho lenguaje (EL) contiene características similares a JavaScript. ¬ Una expresión EL comienza con los caracteres “${” y termina con el carácter “}” ¬ Ejemplo de uso de una expresión EL para el seteo de un atributo de un JavaBean (el uso de JavaBeans será abordado más adelante):



Pregunta 8 (5 puntos): ¿Qué es un bean, cómo se implementa, y para qué se suele utilizar cuando se trabaja con el patrón MVC?

**Un Bean es un componente software que tiene la particularidad de ser reutilizable y así evitar la tediosa tarea de programar los distintos componentes uno a uno. Se puede decir que existen con la finalidad de ahorrarnos tiempo al programar.**

****

**Inner Class**

Una Inner Class (o clase anidada) es una clase definida dentro del cuerpo de otra clase. ¬ Puede ser privada, protegida o pública. ¬ Tiene acceso a todos los miembros definidos en su clase huésped, incluso aquellos marcados como privados. ¬ La definición de una Inner Class dentro de una clase se justifica cuando su existencia depende de la existencia de la clase huésped. ¬ Generalmente se define para utilizarla exclusivamente dentro de la clase huésped

¬ Si se piensa instanciar la clase anidada fuera de la clase huésped, además de definirle visibilidad public o protected, debe marcarse como estática.

¬ En tal caso los miembros de la clase anidada no podrán acceder a miembros de instancia de la clase huésped aunque sí obviamente a los marcados como static.

¬ **Una inner class anónima** es una clase que se define justo en el momento en que se necesita una instancia de dicha clase, y se sabe que no se necesitará en otro lugar. ¬ No tiene nombre (anónima) y se define a partir de una clase base existente o más comúnmente de una interfaz. ¬ En el mismo lugar donde se define, se utiliza el operador new para crear la instancia. ¬ En el cuerpo de la inner class anónima, generalmente se redefinen operaciones de su clase baso o interfaz. Es decir, son útiles para proporcionar nuevas implementaciones a las operaciones de la clase base o interfaz que serán utilizadas en un único lugarAnalista de Sistemas - Escuela de Sistemas - BIOS 50 Inner Class Anónima (2) ¬ Por ejemplo, si un método necesita que le pasen por parámetro una instancia de A, se le puede pasar una nueva instancia de una inner class anónima que extienda o implemente A (según A sea una clase o interfaz), definida dentro de la propia invocación al método. ¬ Dentro de la inner class anónima se podrán redefinir operaciones de A, por lo que la instancia pasada al método, será un A (por reemplazabilidad), pero con una implementación particular

Que es el Web Container?

Que es Scoope request?

1) ¿Qué es una inner class anónima?

Una inner class es una clase que se define dentro del cuerpo de otra clase. En el caso de una inner class anónima, se la define justo en el lugar donde se necesita una instancia de dicha clase y se sabe que no se necesitará ninguna instancia de esa clase en otro sitio del código. La inner class anónima se define a partir de una clase abstracta o interface, no recibe nombre, y en su cuerpo típicamente se redefinen las operaciones necesarias de su clase abstracta base o interface. Ejemplo:

// Suponiendo que la firma de unMetodo es unMetodo(InterfaceA). Al invocarlo...

unMetodo(new InterfaceA() {

// Esto es una inner class anónima.

// Aquí se redefinen las operaciones de InterfaceA.

});

2) ¿Qué operación del API de Reflection debe invocarse sobre un objeto Class para obtener todos sus atributos públicos? ¿Y para obtener un atributo público en concreto? En ambos casos indique y describa la firma.

A partir de un objeto de tipo Class puede invocarse la operación Field[] getFields() que devuelve un array de objetos Field correspondientes a los atributos públicos de la clase representada por el objeto Class.

Si se desea obtener un atributo concreto de dicha clase puede invocarse la operación Field getField(String) que devuelve el objeto Field correspondiente al atributo requerido. A esta operación hay que pasarle por parámetro un String con el nombre del atributo buscado.

3) ¿Qué pasos deben seguirse para implementar una transacción en JDBC? Suponga que ya ha obtenido el objeto Connection.

En primer lugar hay que deshabilitar el autocommit de la conexión (ya que por defecto está habilitado, significando esto que cada vez que se manda a ejecutar un Statement a través de ésta, los cambios se confirman de manera automática). Para ello la interfaz Connection dispone de la operación setAutocommit(boolean) a la que tendremos que pasarle el valor false. Luego de deshabilitar el autocommit se podrán ejecutar todos los Statements con las actualizaciones correspondientes, que no serán confirmados sino hasta que se finalice la transacción invocando al método commit() sobre la conexión. En caso de producirse algún tipo de error deberá invocarse (típicamente en un bloque catch) al método rollback() de la conexión que finalizará la transacción dejando sin efecto los cambios realizados. Por último, si se desea seguir utilizando la conexión para otras tareas, es posible volver a habilitar su autocommit pasándo esta vez el valor true al método setAutocommit(boolean) de la misma.

4) Describa 2 de los scopes utilizados en Java Web.

WebContext / Application: Permite mantener estado global que es accesible por cualquier componente web de la aplicación (servlet, página JSP, etc.), ejecutado por cualquier usuario. Dicho estado se pierde cuando se reinicia o se detiene la aplicación en el servidor.

Session: Permite mantener estado individual que es accesible por cualquier componente web de la aplicación, ejecutado por un usuario concreto. Dicho estado se pierde cuando caduca la sesión del usuario (el tiempo de expiración de la sesión de usuario por inactividad se puede configurar en el deployment descriptor).

Request: Permite mantener estado que es accesible por cualquier componente web de la aplicación, pero sólo durante el request actual. Dicho estado se pierde cuando finaliza el request actual y se envía la respuesta al cliente.

Page: Permite mantener estado que es accesible únicamente por una página JSP. Dicho estado se pierde fuera del contexto de la página JSP.

Grupo 20:30

1) Describa la clase DateFormat y al menos dos de sus métodos vistos en el curso.

La clase DateFormat permite trabajar con formatos estándar de fechas y horas. Los dos usos más frecuentas de esta clase son:

- A partir de un objeto Date con un valor de fecha / hora, obtener un String con el valor de la fecha / hora en un formato estándar.

- A partir de un String con la representación de una fecha / hora en un formato estándar, obtener un objeto Date con el valor de dicha fecha / hora.

Para estos dos usos cuenta con los métodos String format(Date), y Date parse(String) respectivamente.

La instancia de DateFormat puede obtenerse con los métodos getDateInstance(int), getTimeInstance(int) o getDateTimeInstance(int, int). Los enteros que reciben por parámetro estas operaciones corresponden a constantes definidas en la clase DateFormat que representan los formatos estándar soportados: DEFAULT, SHORT, MEDIUM, LONG y FULL.

2) ¿Qué operación del API de Reflection debe invocarse sobre un objeto Class para obtener todos sus métodos públicos? ¿Y para obtener un método público en concreto? En ambos casos indique y describa la firma.

A partir de un objeto de tipo Class puede invocarse la operación Method[] getMethods() que devuelve un array de objetos Method correspondientes a los métodos públicos de la clase representada por el objeto Class.

Si se desea obtener un método concreto de dicha clase, puede invocarse la operación Method getMethod(String, Class...) que devuelve el objeto Method correspondiente al método requerido. A esta operación hay que pasarle por parámetro un String con el nombre del método buscado, y opcionalmente una cantidad variable de parámetros de tipo Class que representan los tipos de datos de los parámetros del método en cuestión.

3) ¿Cuándo y para qué se utilizan las constantes TYPE\_FORWARD\_ONLY, TYPE\_SCROLL\_INSENSITIVE y TYPE\_SCROLL\_SENSITIVE en JDBC?

Estas constantes sirven para configurar el tipo de desplazamiento del cursor de un ResultSet. Se utilizan al momento de crear o preparar el Statement (junto con las constantes que definen la configuración de actualización de los datos): conexion.createStatement(int, int), donde el primer parámetro puede ser alguna de las tres constantes mencionadas:

TYPE\_FORWARD\_ONLY: El cursor se podrá desplazar únicamente hacia adelante, de a una fila por vez, invocando al métod next().

TYPE\_SCROLL\_INSENSITIVE: El cursor se podrá desplazar en cualquier sentido, de a varias filas por vez, e incluso desplazarse a una posición específica, utilizando métodos como next(), previous(), beforeFirst(), afterlast(), first(), last(), absolute(int) y relative(int). En general (depende del fabricante) el ResultSet no será sensible a los cambios que otras conexiones realicen sobre los datos del mismo.

TYPE\_SCROLL\_SENSITIVE: Idem anterior, excepto que en general (depende del fabricante) el ResultSet sí será sensible a los cambios que otras conexiones realicen sobre los datos del mismo.

4) ¿Qué diferencias existen entre redirigir hacia otro recurso con sendRedirect o con forward?

Con sendRedirect se solicita al browser un nuevo request hacia el recurso al que redireccionamos; con forward se redirecciona el mismo request que recibió el recurso inicialmente invocado.

Como consecuencia de lo anterior, con sendRedirect se pierden los parámetros (y toda la información) del request original (ya que se origina un nuevo request). Con forward se mantienen todos los parámetros, ya que el request que le llega al recurso al que redireccionamos es el mismo que recibió el recurso originalmente invocado. Si se desean mantener los parámetros con sendRedirect, deben agregarse al query string de la url del nuevo recurso.

Con sendRedirect es posible redireccionar a recursos fuera del container / servidor. Con forward no.

Con sendRedirect el usuario puede ver como la url que le muestra el browser cambia para reflejar la url del nuevo recurso. Con forward no.

1) ¿Cómo se indica en Java que una operación puede ser redefinida en una clase derivada? Explíquelo.

En Java, todas las operaciones son redefinibles por defecto. A diferencia de C#, por ejemplo, donde hay que utilizar la palabra virtual para permitir la redefinición en una clase derivada, en Java no hay que indicar nada. Por el contrario, si se desea impedir la redefinición, hay que marcarlo explícitamente con la palabra final.

2) Indique el uso y sintaxis del método invoke de la clase Method del API de Reflection.

El método invoke de la clase Method se utiliza para invocar o ejecutar el método representado por una instancia de dicha clase.

La sintaxis sería así:

Object resultado = miMetodo.invoke(Object instancia, Object... argumentos); // miMetodo es una referencia de tipo Method.

Donde:

- instancia es la instancia sobre la cual se invocará el método (en caso de tratarse de un método de instancia). Si se tratara de un método estático, no sería necesaria ninguna instancia y habría que pasar null.

- argumentos es una lista variable de parámetros de cualquier tipo que se utiliza para pasarle los argumentos necesarios al método que se está invocando.

3) ¿Qué operaciones deben redefinirse al implementar la interfaz Filter? Explique para que sirve cada una.

Deben redefinirse las siguientes operaciones:

void init (FilterConfig filterConfig) - Se utiliza para inicializar el filtro, leer parámetros de inicio del mismo (mediante el parámetro filterConfig), etc.

void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) - Aquí hay que programar la tarea que debe realizar el filtro para luego invocar al siguiente filtro en la cadena si corresponde (mediante el parámetro chain).

void destroy() - Liberación de recursos, etc.

4) ¿Qué diferencia existe entre utilizar la acción estándar jsp:setProperty anidada dentro de la acción jsp:useBean, o fuera de ésta?

Es necesario recordar que si una acción jsp:useBean no encuentra el bean en el scope indicado, ésta lo creará (siempre que se haya utilizado el atributo class para declarar su tipo).

Si una acción jsp:setProperty aparece anidada dentro de la acción jsp:useBean, se ejecutará solamente cuando el bean no exista en el scope indicado y sea necesario instanciarlo (esta forma se utiliza para inicializar el bean que se está creando). En caso de que el bean ya exista, la o las acciones jsp:setProperty anidadas no se ejecutarán.

Por el contrario, si una acción jsp:setProperty aparece por fuera de la acción jsp:useBean, se ejecutará cada vez que se solicite la página, independientemente de la existencia o no del bean en el scope indicado.

Grupo 20:30

1) ¿Cuándo utilizaría una colección de tipo ArrayList y cuando una de tipo LinkedList?

Por su implementación interna basada en arrays, una colección de tipo ArrayList sería adecuada cuando se requieren muchos accesos aleatorios a sus elementos (por ejemplo, al elemento que está en la posición N) y pocas inserciones y eliminaciones (ya que requieren mucho mantenimiento). En cambio, debido a que una colección de tipo LinkedList contiene elementos con doble enlace (al anterior y siguiente elemento), su uso es más recomendable cuando se requieren muchas inserciones y eliminaciones (el mantenimiento solo requiere cambiar un par de referencias), pero pocos accesos aleatorios (hay que recorrer desde el principio todos los elementos, preguntando por el siguiente hasta llegar al elemento deseado).

2) Describa 3 de los métodos presentes en la clase Class del API de Reflection vistos en el curso.

método forName (estático): Permite obtener una instancia de Class a partir de una cadena de texto con su nombre completo (incluyendo paquetes).

método getName: Permite obtener una cadena de texto con el nombre completo de la clase representada por la instancia de Class (incluyendo paquetes).

método getFields: Permite obtener un array de elementos Field que representan atributos de la clase representada por la instancia de Class.

3) ¿Qué método utilizaría si desea redirigir el request del usuario hacia otro componente dentro del servidor sin perder los parámetros recibidos? Justifique.

Utilizaría el método forward presente en la interfaz RequestDispatcher:

RequestDispatcher rd = request.getRequestDispatcher(<URL\_RECURSO>);

if (rd != null) rd.forward(request, response);

El método forward redirecciona el request actual al nuevo recurso y se resuelve completamente dentro del servidor. De esta manera es posible conservar los parámetros del request original, ya que el request no cambia (el nuevo recurso recibe el mismo request que el recurso original).

Sin embargo, si se redireccionara mediante el método sendRedirect de la interfaz HttpServletResponse se originaría un nuevo request por parte del cliente, por lo que se perderían los parámetros del request original (una manera de conservarlos sería agregarlos al query string de la url del nuevo recurso).

4) Indique que tipos de componentes Java Web se utilizan en cada uno de los componentes del patrón MVC.

Al aplicar el patrón MVC en una aplicación Java Web suelen utilizarse servlets como controladores, páginas JSP como vistas y JavaBeans para manejar la información del modelo.

Un servlet controlador recibe un request por parte del cliente, interactúa con el modelo que se encarga de procesar la tarea, y luego selecciona una página JSP como vista que mostrará los resultados. Para el transporte de la información del modelo entre los componentes se utilizan los JavaBeans.

**¿Cuáles son las 3 formas de obtener un objeto Class al utilizar Reflection?**

Se verán 3 formas de obtener un objeto Class:

• A partir de un String con el nombre y ubicación de la clase: Class c = Class.forName(“logica.Persona”); ◊ Se debe manejar la excepción ClassNotFoundException.

• A partir de un objeto ya existente: Persona p = new Persona(…); Class c = p.getClass(); ◊ La operación getClass() pertenece a la clase Object.

• A partir de un tipo de datos: Class c = int.class; ó Class c = Persona.class

¿Qué diferencias existen entre las operaciones executeQuery() y executeUpdate() de la

interfaz Statement?

executeQuery() cuando el SP consulta la BD

executeUpdate() cuando el SP actualiza la BD