Ejercicios de creación de Servicios Web

Índice

Nota:

En las plantillas de la sesión se incluyen únicamente los ejemplos vistos en los apuntes. Para realizar los ejercicios estas plantillas no son necesarias.

1. Creación de un servicio web básico

Vamos a comenzar creando un servicio web básico. Este servicio web será un *Hola Mundo* en forma de servicio. Tendrá como nombre `HolaMundoSW` y una única operación `String saluda(nombre)`, que nos devolverá un mensaje de saludo incluyendo el nombre proporcionado. Por ejemplo, si al parámetro de entrada `nombre` le damos como valor `"Miguel"`, como salida producirá la cadena `"Hola Miguel"`.

Implementar el servicio utilizando BEA Workshop, en un proyecto con nombre `servcweb-sesion02-hola`, y probarlo con *WebLogic Test Client* para comprobar su correcto funcionamiento.

2. Invocar un servicio web desde otro servicio

Vamos a implementar con BEA Workshop un servicio de conversión monetaria que nos permita convertir entre euros y ptas y entre euros y dólares. Este servicio deberá llamarse `ConversionSW`, y estará dentro de un proyecto con nombre `servcweb-sesion02-conversion`.

Nota:

En los apuntes de teoría se puede encontrar una guía paso a paso para crear este servicio.

Se pide:

a) Implementar las operaciones de conversión de euros a ptas y de ptas a euros. Dado que conocemos que la tasa de cambio es fija (1 euro = 166.386 ptas) podemos implementar directamente estas operaciones:

```
double ptas2euro(int ptas)
int euro2ptas(double euros)
```

Probar que estas operaciones funcionan correctamente en *WebLogic Test Client*.

b) Ahora implementaremos funciones para convertir una cantidad de euros a dólares y de dólares a euros:

```
double dolar2euro(double dolar)
double euro2dolar(double euros)
```

En este caso necesitaremos recurrir a un servicio externo para obtener la tasa de cambio actual. Utilizaremos el servicio *Currency Exchange Service* de *XMethods*. Utilizar controles para acceder a este servicio remoto, cuyo documento WSDL se puede encontrar en:

```
http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl
```

3. Validación de NIFs (*)

Vamos a implementar un servicio con una serie de métodos que nos permitan validar un NIF. El servicio tendrá como nombre `ValidaDniSW`, y estará dentro de un proyecto `servcweb-sesion02-dni`. Se pide:

a) Implementar la siguiente operación:

```
boolean validarDni(String dni)
```

Esta función tomará como entrada un DNI, y como salida nos dirá si es correcto o no. El resultado será `true` si el DNI es correcto (está formado por 8 dígitos), y `false` en caso contrario. Podemos utilizar un código similar al siguiente para validar el DNI:

```
Pattern pattern =  Pattern.compile("[0-9]{8,8}");
Matcher matcher = pattern.matcher(dni);
return matcher.matches();
```

b) Implementar la siguiente operación:

```
char obtenerLetra(String dni)
```

Esta función tomará como entrada un DNI, y como salida nos dirá la letra que corresponde a dicho DNI. Lo primero que deberá hacer es validar el DNI (puede utilizar para ello el método definido en el apartado anterior), y en caso de no ser correcto lanzará una excepción de tipo `SOAPFaultException` indicando el mensaje de error adecuado. Una vez validado, calcularemos la letra que corresponda al DNI. Para ello deberemos:

- Obtener el índice de la letra correspondiente con:

```
dni % 23
```

- La letra será el carácter de la siguiente cadena que esté en la posición obtenida anteriormente:

```
"TRWAGMYFPDXBNJZSQVHLCKE"
```

c) Implementar la siguiente operación:

```
boolean validarNif(String nif)
```

Esta función tomará como entrada un NIF, y como salida nos dirá si es correcto o no. El resultado será `true` si el NIF es correcto (está formado por 8 dígitos seguidos de la letra correcta), y `false` en caso contrario. Para hacer esta comprobación se pueden utilizar las dos funciones anteriores: se comprobarán los 8 primeros caracteres con la función `validarDni` y posteriormente se comprobará si la letra es la correcta utilizando la función `obtenerLetra`.

4. Tienda de DVDs (*)

Nuestro negocio consiste en una tienda que vende películas en DVD a través de Internet. Para dar una mayor difusión a nuestro catálogo de películas, decidimos implantar una serie de Servicios Web para acceder a información sobre las películas que vendemos.

De cada película ofreceremos información sobre su título, su director y su precio. Esta información podemos codificarla en una clase `DatosPelicula` como la siguiente:

```
public class DatosPelicula {
  private String titulo;
  private String director;
  private float precio;

  public DatosPelicula() {}

  public DatosPelicula(String titulo, String director, float precio) {
    this.titulo = titulo;
    this.director = director;
    this.precio = precio;
  }

  // Getters y setters
  ...
}
```

Vamos a permitir que se busquen películas proporcionando el nombre de su director. Por lo tanto, el servicio ofrecerá una operación como la siguiente:

```
DatosPelicula [] buscaDirector(String director)
```

Proporcionaremos el nombre del director, y nos devolverá la lista de películas disponibles dirigidas por este director.

En un principio, podemos crear una lista estática de películas dentro del código de nuestro servicio, como por ejemplo:

```
final static DatosPelicula[] peliculas = {
    new DatosPelicula("Mulholland Drive", "David Lynch", 26.96f),
    new DatosPelicula("Carretera perdida", "David Lynch", 18.95f),
    new DatosPelicula("Twin Peaks", "David Lynch", 46.95f),
    new DatosPelicula("Telefono rojo", "Stanley Kubrick", 15.95f),
    new DatosPelicula("Barry Lyndon", "Stanley Kubrick", 24.95f),
    new DatosPelicula("La naranja mecánica", "Stanley Kubrick",
22.95f)
};
```

Se pide:

a) Implementar el servicio utilizando BEA Workshop. El servicio se llamará `TiendaDvdSW`, y estará dentro de un proyecto con nombre `servcweb-sesion02-tienda`.

Para construir una lista con las películas cuyo director coincida con el nombre del director que se ha solicitado, podemos utilizar un código similar al siguiente, donde se ha proporcionado un parámetro `director`:

```
director = director.toLowerCase();

ArrayList<DatosPelicula> list = new ArrayList<DatosPelicula>();

for (DatosPelicula pelicula : peliculas) {
  if (pelicula.getDirector().toLowerCase().indexOf(director) != -1) {
    list.add(pelicula);
  }
}

DatosPelicula [] result = new DatosPelicula[list.size()];
list.toArray(result);

return result;
```

Una vez implementado el servicio, desplegarlo en Weblogic y probarlo mediante *Weblogic test client*. Observar los tipos de datos definidos en el documento WSDL generado y en los mensajes SOAP para la invocación del servicio.

b) Vamos a implementar una segunda operación en nuestro servicio:

```
DatosPelicula [] buscaDirectorImportacion(String director)
```

En este caso no obtendrá la lista de películas de nuestra base de datos local, sino que las buscará en Amazon mediante los servicios web que ofrece esta compañía.

Generar los tipos de datos para acceder al servicio de Amazon mediante JAX-RPC, para

obtener una representación más parecida a la vista en los ejercicios del tema anterior. De esta forma podremos utilizar un código similar para acceder al servicio, con la diferencia de que una vez obtenidos los datos, en lugar de imprimirlos en la consola los encapsularemos en una serie de objetos `DatosPelicula` que devolveremos como resultado.

Nota:

No debemos devolver directamente el objeto que nos devuelve el servicio de Amazon, ya que debido a su complejidad el cliente de prueba de Weblogic no es capaz de interpretarlo correctamente y obtendríamos un error.