

## EXAMEN PRÁCTICO

**Trimestre:** 2 **Módulo:** Programación de Servicios y Procesos **Fecha:** FEB 2021

**DNI:** X4966439A

**APELLIDOS:** ROWE

**NOMBRE:** ADAM JAMES

**PROVINCIA EXAMEN:** MÁLAGA

**CALIFICACIÓN:**

(A rellenar por el docente)

### Normas:

- El examen práctico tiene una puntuación máxima de 10 puntos y para superarlo se requiere alcanzar un mínimo de 5 puntos.
- El examen práctico deberá ser subido a la plataforma en el tiempo establecido para ello.

### Instrucciones:

- Se ha de utilizar este documento Word, donde tendrás que añadir capturas de pantalla de los ejercicios que lo requieran y comentar los pasos dados.
- Se enviarán dos ficheros a la plataforma: el documento en Word y el mismo pero convertido a pdf, comprimidos en un único fichero zip sin contraseña. El nombre del fichero que se envía a la plataforma tendrá el siguiente formato:
  - **Apellido1\_Apellido2\_Nombre\_EX\_T2\_PRACTICO\_PSP**

## Ejercicio 1.- (4 puntos). CREACION DE UN SERVICIO WEB QUE DEVUELVE UN XML

Crea un proyecto para desarrollar el servicio web de una empresa, que devuelva el recurso **"Empleado"** en XML, que tendrá los campos id, nombre, departamento.

**Pega aquí todo el código y pantallazos que justifiquen su correcto funcionamiento (pruébalo con Restclient o Postman y enséñalo en pantallazos).**

Vamos a comenzar con la creación de un proyecto de tipo **'Web Application'** utilizando la IDE de NetBeans J2EE.

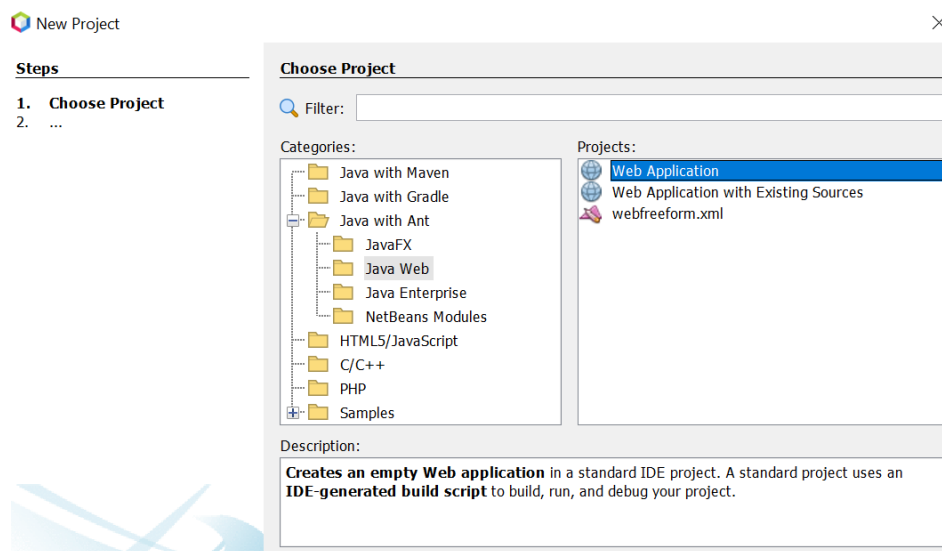


Ilustración 1: Crear Web Application.

Llamaremos el proyecto **'Empresa'**.

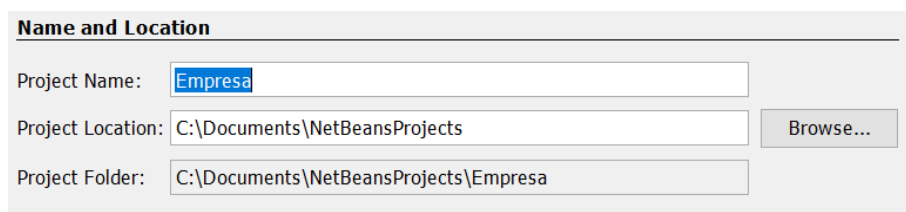


Ilustración 2: Nombre del Proyecto.

Añadiremos el servidor a utilizar donde se va a alojar nuestra **aplicación web**.

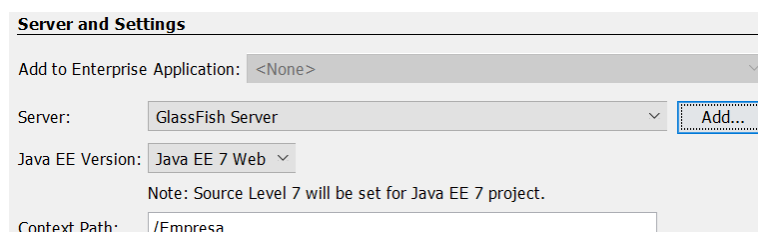


Ilustración 3: Configuración Servidor.

Continuaremos creando un fichero de tipo '**RESTful Web Service Design Patterns**'.

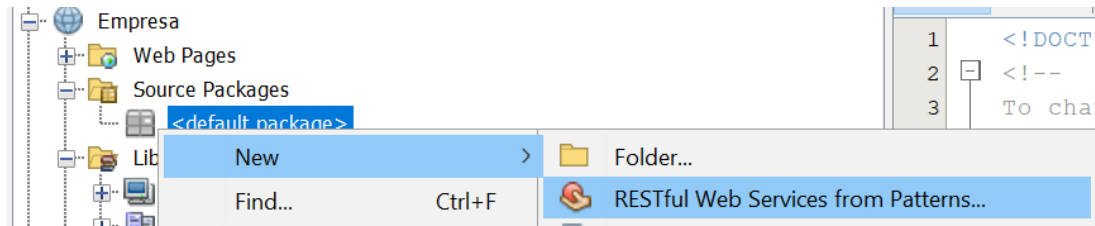
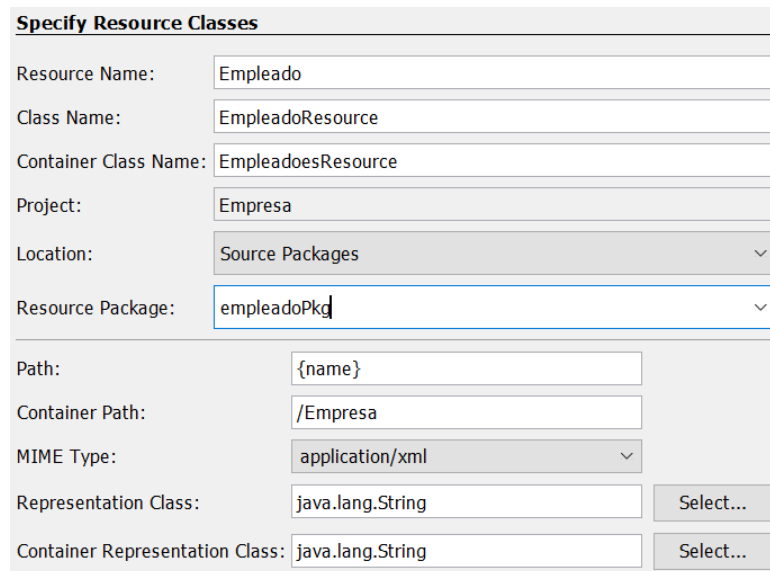


Ilustración 4: Crear Archivo RESTful.

Configuraremos los recursos *RESTful* correspondientes, asegurándonos que el *MIME Type* se encuentre en '**application/xml**' ya que en este caso queremos generar el recurso correspondiente en '**XML**'.



The 'Specify Resource Classes' dialog box is shown. It contains the following fields and values:

- Resource Name: Empleado
- Class Name: EmpleadoResource
- Container Class Name: EmpleadosResource
- Project: Empresa
- Location: Source Packages
- Resource Package: empleadoPkg
- Path: {name}
- Container Path: /Empresa
- MIME Type: application/xml
- Representation Class: java.lang.String
- Container Representation Class: java.lang.String

There are 'Select...' buttons next to the 'Representation Class' and 'Container Representation Class' fields.

Ilustración 5: Parámetros de Configuración de Recursos.

Además crearemos un modelo del recurso, siendo en este caso '**Empleado**' donde inicializaremos los campos '**id**', '**nombre**' e '**departamento**' y los encapsularemos creando sus '**getters**' y '**setters**' correspondientes.

```
@XmlRootElement
public class Empleado {

    private int id;
    private String nombre;
    private String departamento;

    // Getters y Setters variables Inicializadas
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
```

Ilustración 6: Inicializar y Encapsular.

Además añadiremos un constructor que recoja los parámetros de nuestro recurso de la siguiente manera:

```
public Empleado(int pid, String pnombre, String pdepartamento) {
    this.id = pid;
    this.nombre = pnombre;
    this.departamento = pdepartamento;
}

@Override
public String toString() {
    return new StringBuffer(" id: ").
        append(id).
        append(" nombre: ").
        append(nombre).
        append(" departamento: ").
        append(departamento).toString();
}
```

Ilustración 7: Constructores Recurso.

En el '@GET' del archivo 'EmpleadoResource' añadiremos el constructor con sus datos correspondientes.

```
@GET
@Produces("application/xml")
public Empleado getXml() {
    //TODO return proper representation object

    return new Empleado(1, "Adam James Rowe", "Computer Science");
    // throw new UnsupportedOperationException();
}
```

Ilustración 8: GetXML.

Además será necesario añadir el '.jar' de Jersey, desde propiedades del proyecto.

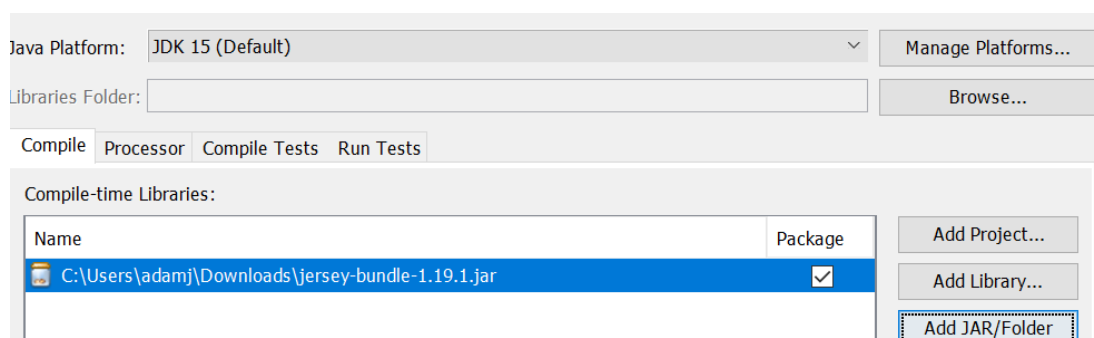


Ilustración 9: Jersey 1.19

Añadimos el fichero 'web.xml' al proyecto donde añadiremos el Servlet de jersey y su correspondiente mapeo.

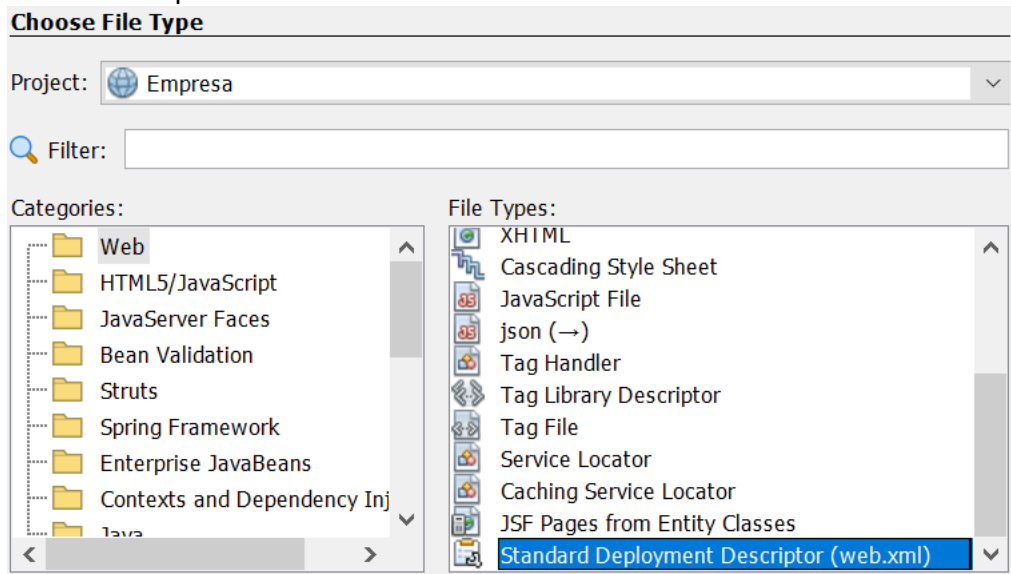


Ilustración 10: Crear Web.xml

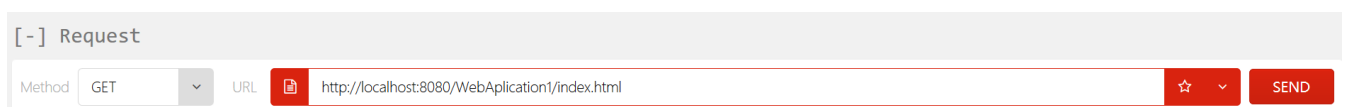
Añadimos la ruta y el Servlet de Jersey.

```
<servlet>
  <servlet-name>Jersey Web Application</servlet-name>
  <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-cl
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>Jersey Web Application</servlet-name>
  <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

Ilustración 11: Añadir Configuración Servlets.

Ya será posible hacer un '**Clean and Build**' Asegurandonos de contar con el servidor **GlassFish** correctamente iniciado y accederemos a '**RESTClient**' desde el plugin de **Firefox**.



## Ejercicio 2.- (4 puntos).

Dado el siguiente servidor escrito en java:

### SERVIDOR

```
import java.net.*;
import java.io.*;

public class Servidor {

    public static void main(String args[]) throws IOException {
        ServerSocket ss = new ServerSocket(7777);
        while(true){
            Socket s = ss.accept();
            PrintWriter out = new PrintWriter(s.getOutputStream(), true);
            out.println("¡Bienvenido!");
            s.close();
        }
    }
}
```

**a) Escribe un cliente en Java que se conecte a este servidor, lea la información que envía el servidor y la visualice por pantalla. Pega aquí el código:**

vamos a crear una nueva clase que haga la función de cliente y se conecte al servidor dado anteriormente, desde el código facilitado podremos ver que el puerto o socket que hace uso el servidor es el '7777' por ello será necesario en primer instancia en la clase cliente que utilice este puerto para realizar la conexión al servidor, además ya que el servidor se aloja en la misma máquina que el cliente podremos deducir que la dirección IP a utilizar será 'localhost' o bien '127.0.0.1', además inicializaremos un nuevo 'BufferedReader' que tendrá la función de recoger y leer texto recibido del servidor y almacenar este en su correspondiente variable.

**Podremos visualizar el resultado de la clase cliente en la página siguiente.**

```
package com.empresa.simulacro;

import java.io.BufferedReader;

public class cliente {
    public static void main (String args[]) {
        //Inicializar Socket, Buffer de Lectura.
        Socket cliente = null;
        BufferedReader ent = null;
        //Almacenar en variable la dirección IP del servidor
        String ipServer = "127.0.0.1";

        try {
            //Instanciamos el nuevo puerto con IP del servidor y puerto
            cliente = new Socket(ipServer, 7777);
            //Lectura por medio de la conexión del puerto cliente
            ent = new BufferedReader(new InputStreamReader(cliente.getInputStream()));

        } catch (UnknownHostException e) {
            System.err.println("Servidor no levantado.");
        } catch (Exception e) {
            System.err.println("Error: " + e);
        }

        try {
            //Inicializar variable con contenido de texto servidor
            String recibido;
            //Almacenar en variable contenido recibido de servidor
            recibido = ent.readLine();
            //Mostrar en consola contenido del servidor
            System.out.println(recibido);
            //Cerrar Conexión
            ent.close();

        } catch (UnknownHostException excepcion) {
            System.err.println("No se encuentra el servidor en la dirección ");
        } catch (IOException excepcion) {
            System.err.println("Error de entrada/salida");
        } catch (Exception e) {
            System.err.println("Error: " + e);
        }

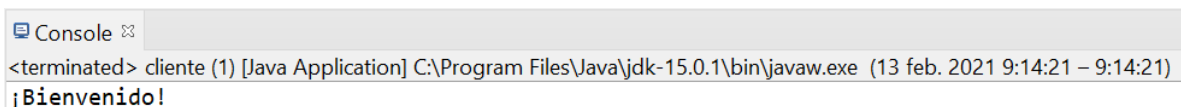
    }
}
```

Ilustración 12: Clase Cliente.

### b) Ejecuta servidor y cliente, y pega aquí pantallazo de la salida del cliente:

Para poder recibir el mensaje del servidor al cliente será necesario contar con el servidor en ejecución al iniciar la clase cliente de manera que pueda recoger el mensaje.

El mensaje será el siguiente:



```
Console
<terminated> cliente (1) [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (13 feb. 2021 9:14:21 - 9:14:21)
¡Bienvenido!
```

Ilustración 13: Mensaje Cliente Servidor Java.

### Ejercicio 3.- (2 puntos).

Cifra mediante el algoritmo César (Caesar) el siguiente mensaje en claro: "procesos". El desplazamiento a utilizar para encriptar es 4 y la cadena para usar de referencia es: "abcdefghijklmnopqrstuvwxyz".

**El mensaje cifrado es: tvsgiwsw**

Para encriptar un mensaje podremos hacer uso del siguiente método donde añadiremos nuestro mensaje en claro '**procesos**' como la variable '**mensaje**', teniendo en cuenta el desplazamiento siendo en este caso **de 4** y por último también tendremos que incluir la cantidad de caracteres de la cadena de referencia para realizar el encriptado correspondiente.

**El método de encriptar quedará de la siguiente manera.**

```
// método para encriptar
private String encriptar()
{
    String temp = "abcdefghijklmnopqrstuvwxyz ";
    String translate = new String();
    for (int i = 0; i < mensaje.length(); i++)
    {
        String t = Character.toString(mensaje.charAt(i));
        int index = temp.indexOf(t);
        index = (index+offset) % 27;
        String add = Character.toString(temp.charAt(index));
        translate += add;
    }
    return translate;
}
```

**Ilustración 14: Método Encriptar.**

A continuación añadiremos el mensaje, el offset o desplazamiento y mostraremos el mensaje encriptado por pantalla haciendo uso del método de encriptar anterior, además devolveremos por consola el mensaje desencriptado para verificar el mensaje siendo encriptado.

```
public static void main(String[] args) {
    String encriptado;
    String desencriptado;
    Caesar Crip = new Caesar();
    Crip.mensaje="proceso";
    Crip.offset=4;
    System.out.println(encriptado=Crip.encriptar());
    Crip.mensaje=encriptado;
    System.out.println(desencriptado=Crip.desencriptar());
    Crip.mensaje = desencriptado;
}
```

**Ilustración 15: Clase Principal Cesar Desplazamiento 4.**



El resultado por consola del proceso de encriptación del mensaje será el siguiente:

```
] --- exec-maven-  
tvsgius  
· proceso  
-----  
BUILD SUCCESS
```

**Ilustración 16: Consola Mensaje Encriptado.**