

# Desarrollo de Sistemas Distribuidos - 4CM3

[Tablero](#) / [Mis cursos](#) / [SISDIS-4CM3](#)

Su progreso ?



[Avisos](#)



[Lineamientos del curso - Desarrollo de Sistemas Distribuidos](#)



[Calendario académico 20-21](#)



[Primer examen parcial](#)



[Segundo examen parcial](#)



El examen es individual y se presentará en forma remota utilizando la plataforma Moodle.

El examen se podrá presentar el viernes 7 de mayo de 2021 a las 12:00 Hrs. con una duración máxima de 90 minutos.

Los temas a evaluar son los siguiente:

- Paradigma de paso de mensajes.
- Objetos locales.
- Objetos remotos.
- Paradigma de objetos distribuidos.
- Remote Method Invocation.
- Java RMI.
- Cómo usar Java RMI.
- Java RMI en una red privada y en una red pública.
- Cómo ejecutar Java RMI en la nube.
- Objetos JSON.
- Arreglos JSON.
- GSON.
- Conceptos básicos de Servicios web.
- Participación en un servicio web.
- Servicios web basados en SOAP.
- Componentes de un mensaje SOAP.
- Web Services Description Language (WSDL).
- Servicios web estilo REST.
- JAX-RS.
- Ejecución de servicios web estilo REST en Tomcat.
- Cómo invocar un método web REST utilizando Java.

---

## 3. Sistemas basados en objetos distribuidos



Clase del día - 13/04/2021

La clase de hoy vamos a iniciar con el tema Sistemas basados en objetos distribuidos.

## Paradigma de paso de mensajes

Hasta ahora hemos desarrollado programas distribuidos utilizando paso de mensajes.

El paradigma de **paso de mensajes** es el modelo natural para el desarrollo de sistemas distribuidos, ya que reproduce la comunicación entre las personas.

En el paradigma de paso de mensajes, las computadoras comparten los datos utilizando mensajes. El programador debe serializar los datos antes de enviarlos, y des-serializar los datos después de recibirlos.

El desarrollo de sistemas basados en paso de mensajes es complejo debido a que el programador debe controlar el intercambio de los mensajes, además de desarrollar la funcionalidad propia del sistema.

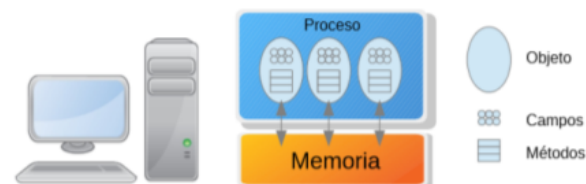
El paradigma de paso de mensajes es **orientado a datos**.

## Objetos locales

Un objeto encapsula variables (campos) y funciones (métodos). Las variables guardan el estado del objeto y los métodos permiten modificar y acceder el estado del objeto.

Un objeto local es aquel cuyos métodos son invocados por un proceso local, es decir, un proceso que ejecuta en la misma computadora dónde reside el objeto.

Los objetos locales comparten el espacio de direcciones, en otras palabras, los objetos locales son objetos que residen en la misma memoria.



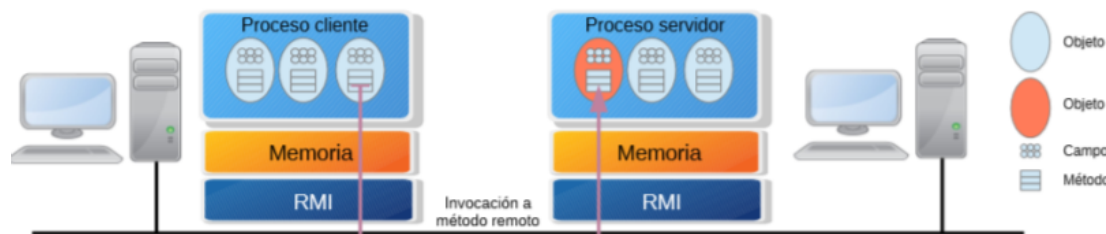
## Objetos remotos

Un objeto remoto es aquel cuyos métodos son invocados por procesos remotos, es decir, procesos que ejecutan en una computadora remota conectada mediante una red.

Los objetos que se encuentran en diferentes computadoras no comparten el espacio de direcciones, por tanto, solo comparten valores pero no referencias.

La siguiente figura muestra un proceso cliente y un proceso servidor que ejecutan en diferentes computadoras. En el proceso cliente un método local invoca un método remoto, el cual forma parte de un objeto contenido en el proceso servidor.

En este caso, la invocación de los métodos remotos se realiza mediante una capa llamada RMI (*Remote Method Invocation*).



## Paradigma de objetos distribuidos

El paradigma de objetos distribuido combina objetos locales y objetos remotos. La ventaja que tiene, comparado con el paradigma de paso de mensajes, es que el paradigma de objetos distribuidos representa una abstracción

sobre el paso de mensajes, por tanto el programador no debe preocuparse por controlar el paso de mensajes entre los nodos.

El paradigma de objetos distribuidos es **orientado a la acción**, ya que se basa en la acción que realiza el método remoto invocado.

### Remote Method Invocation

En un sistema que utiliza RMI existe un proceso llamado *registry* el cual hace las funciones de servidor de nombres.

En cada nodo, hay un proceso servidor el cual registra en el servidor de nombres los objetos que exportará. Cada objeto exportado por el servidor será identificado mediante una URL.

Para acceder a un objeto remoto, el proceso cliente consulta el servidor de nombres utilizando la URL, si el objeto es encontrado, entonces el servidor de nombres regresa al cliente una referencia que apunta al objeto remoto. Entonces el proceso cliente utiliza la referencia para invocar los métodos del objeto remoto, los cuales se ejecutan en el servidor.

El paso de parámetros y regreso de resultado es manejado automáticamente por la capa RMI.

### Java RMI

Java RMI es un API que implementa la invocación de métodos remotos. JDK incluye un servidor de nombres llamado **rmiregistry**, esta aplicación se encuentra en el directorio bin del JDK

### ¿Cómo usar Java RMI?

Para utilizar Java RMI se debe seguir los siguientes pasos:

1. Para cada objeto remoto se debe crear una interface **I** que defina el prototipo de cada método a exportar. Es necesario declarar que los métodos remotos pueden producir la excepción `java.rmi.RemoteException`. La interface **I** debe heredar de `java.rmi.Remote`.
2. El código de los métodos remotos se debe escribir en una clase **C** que implemente la interface **I**. La clase **C** debe ser una subclase de `java.rmi.server.UnicastRemoteObject`. El constructor default de la clase **C** debe invocar el constructor de la superclase. Es necesario declarar que los métodos remotos pueden producir la excepción `java.rmi.RemoteException`.
3. El proceso servidor deberá registrar la clase **C** invocando el método `bind()` o el método `rebind()` de la clase `java.rmi.Naming`. A los métodos `bind()` y `rebind()` se les pasa como parámetros la URL correspondiente al objeto remoto y una instancia de la clase **C**. La URL tiene la siguiente forma: **rmi://ip:puerto/nombre**, donde *ip* es la dirección IP de la computadora dónde ejecuta el programa **rmiregistry**, *puerto* es el número de puerto utilizado por **rmiregistry** (se puede omitir si **rmiregistry** utiliza el puerto default 1099) y *nombre* es el nombre con el que identificaremos el objeto.
4. El proceso cliente deber invocar el método `lookup()` de la clase `java.rmi.Naming` para obtener una referencia al objeto remoto. El método `lookup()` regresa una instancia de la clase `Remote`, la cual se debe convertir al tipo de la interface **I** mediante casting. Utilizando la referencia, el proceso cliente invocará los métodos remotos de la clase **C**.

Por razones de seguridad, la aplicación **rmiregistry** se debe ejecutar en la misma computadora dónde ejecuta el servidor.

Por default la aplicación **rmiregistry** utiliza el puerto 1099, si se utiliza otro puerto, se deberá pasar el número de puerto como argumento al ejecutar **rmiregistry**.

Se puede notar que el proceso servidor permanece en ejecución debido a que los métodos `bind()` y `rebind()` crean threads que no terminan.

### Ejemplo de Java RMI

Como vimos anteriormente, para crear una aplicación que utilice Java RMI es necesario crear una interface, una clase, y dos programas (un cliente y un servidor).

En este caso, vamos a crear un objeto remoto que exportará los siguiente métodos:

- `mayusculas()`, recibe como parámetro una cadena de caracteres y regresa la misma cadena convertida a mayúsculas.
- `suma()`, recibe como parámetros dos enteros y regresa la suma.
- `checksum()`, recibe como parámetro una matriz de enteros y regresa la suma de todos los elementos de la matriz.

Primeramente creamos una interface que incluya los prototipos de los métodos a exportar:

```
public interface InterfaceRMI extends Remote
{
    public String mayusculas(String name) throws RemoteException;
    public int suma(int a,int b) throws RemoteException;
    public long checksum(int[][] m) throws RemoteException;
}
```

Ahora escribimos la clase **ClaseRMI** la cual va a contener el código de los métodos definidos en la interface **InterfaceRMI**. Notar que la clase **ClaseRMI** es subclase de `UnicastRemoteObject` e implementa la interface **InterfaceRMI**.

```
public class ClaseRMI extends UnicastRemoteObject implements InterfaceRMI
{
    // es necesario que el constructor ClaseRMI() invoque el constructor de la superclase
    public ClaseRMI() throws RemoteException
    {
        super( );
    }
    public String mayusculas(String s) throws RemoteException
    {
        return s.toUpperCase();
    }
    public int suma(int a,int b)
    {
        return a + b;
    }
    public long checksum(int[][] m) throws RemoteException
    {
        long s = 0;
        for (int i = 0; i < m.length; i++)
            for (int j = 0; j < m[0].length; j++)
                s += m[i][j];
        return s;
    }
}
```

La clase **ServidorRMI** registra en el `rmiregistry` una instancia de la clase **ClaseRMI** utilizando el método `rebind()`.

```
public class ServidorRMI
{
    public static void main(String[] args) throws Exception
    {
        String url = "rmi://localhost/prueba";
        ClaseRMI obj = new ClaseRMI();

        // registra la instancia en el rmiregistry
        Naming.rebind(url,obj);
    }
}
```

El cliente **ClienteRMI** obtiene una referencia al objeto remoto utilizando el método `lookup()`, esta referencia es utilizada para invocar los métodos remotos.

```

public class ClienteRMI
{
    public static void main(String args[]) throws Exception
    {
        // en este caso el objeto remoto se llama "prueba", notar que se utiliza el puerto default 1099
        String url = "rmi://localhost/prueba";

        // obtiene una referencia que "apunta" al objeto remoto asociado a la URL
        InterfaceRMI r = (InterfaceRMI)Naming.lookup(url);

        System.out.println(r.mayusculas("hola"));
        System.out.println("suma=" + r.suma(10,20));

        int[][] m = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
        System.out.println("checksum=" + r.checksum(m));
    }
}

```

## Actividades individuales a realizar

1. Compilar la interface [InterfaceRMI.java](#), la clase [ClaseRMI.java](#) y los programas [ClienteRMI.java](#) y [ServidorRMI.java](#).
2. En una ventana de comandos de Windows (o una terminal de Linux) ejecutar el programa **rmiregistry**.
3. En una ventana de comandos de Windows (o una terminal de Linux) ejecutar el programa [ServidorRMI](#). Notar que el servidor queda en ejecución.
4. En una ventana de comandos de Windows (o una terminal de Linux) ejecutar el programa [ClienteRMI](#). El cliente invoca el método `lookup()` para obtener del `rmiregistry` una referencia al objeto remoto, entonces invoca los métodos del objeto remoto los cuales se ejecutan en el servidor.

 [InterfaceRMI.java](#)




 [ClaseRMI.java](#)



 [ServidorRMI.java](#)



 [ClienteRMI.java](#)



 [Tarea 5. Chat multicast](#)



Desarrollar un programa en Java que implemente un chat utilizando comunicación multicast mediante datagramas.

Se **deberá** ejecutar el programa en una máquina virtual con Windows 10 en Azure. Solo se admitirá la tarea si se trata de un programa en modo consola de caracteres (no se admitirá el programa en modo gráfico).

Se **deberá** pasar como parámetro al programa el nombre del usuario que va escribir en el chat. Para demostrar el programa se **deberá** utilizar los siguientes usuarios: hugo, paco y luis (no usar otros usuarios).

El programa **deberá** utilizar las siguientes funciones para enviar y recibir los mensajes multicast:

```

static void envia_mensaje_multicast(byte[] buffer,String ip,int puerto) throws IOException
{
    DatagramSocket socket = new DatagramSocket();
    socket.send(new DatagramPacket(buffer,buffer.length,InetAddress.getByName(ip),puerto));
    socket.close();
}

```

```

static byte[] recibe_mensaje_multicast(MulticastSocket socket,int longitud_mensaje) throws IOException
{
    byte[] buffer = new byte[longitud_mensaje];
    DatagramPacket paquete = new DatagramPacket(buffer,buffer.length);
    socket.receive(paquete);
    return paquete.getData();
}

```

El funcionamiento general del programa es el siguiente:

- El programa creará un thread que actuará como cliente multicast, el cual recibirá los mensajes del resto de los nodos. Cada mensaje recibido será desplegado en la pantalla. El thread desplegará el mensaje que envía el mismo nodo.
- En el método main(), dentro de un ciclo infinito se desplegará el siguiente prompt: "**Ingrese el mensaje a enviar:**" (sin las comillas), entonces se leerá una string (el mensaje). Se **deberá** enviar el mensaje a los nodos que pertenecen al grupo identificado por la IP 230.0.0.0 a través del puerto 50000. **El paquete a enviar deberá tener la siguiente forma:** *nombre\_usuario:mensaje\_ingresado*, donde *nombre\_usuario* es el nombre del usuario que pasó como parámetro al programa (hugo, paco o luis) y *mensaje\_ingresado* el mensaje que el usuario ingresó por el teclado.

Se **deberá** completar el siguiente programa:

```
class Chat
{
    static class Worker extends Thread
    {
        public void run()
        {
            // En un ciclo infinito se recibirán los mensajes enviados al grupo
            // 230.0.0.0 a través del puerto 50000 y se desplegarán en la pantalla.
        }
    }
    public static void main(String[] args) throws Exception
    {
        Worker w = new Worker();
        w.start();

        String nombre = args[0];

        // En un ciclo infinito se leerá cada mensaje del teclado y se enviará el mensaje al
        // grupo 230.0.0.0 a través del puerto 50000.
    }
}
```

Para probar el programa, se **deberá** ejecutar la siguiente conversación en tres ventanas de comandos (cmd) en la máquina virtual con Windows 10. En la primera ventana escribirá hugo, en la segunda ventana escribirá paco y en la tercera ventana escribirá luis:

hugo debe escribir:

**hola a todos**

paco debe escribir:

**hola hugo**

luis debe escribir:

**hola hugo**

hugo debe escribir:

**¿alguien sabe dónde será la fiesta el sábado?**

paco debe escribir:

**será en la casa de donald**

hugo debe escribir:

**¿a qué hora?**

luis debe escribir:

**a las 8 PM**

hugo debe escribir:

**adios**

paco debe escribir:

**adios hugo**

luis debe escribir:

**adios hugo**

Notar que los signos de interrogación y las letras acentuadas deberán desplegarse correctamente en la ventana de comandos de Windows.

Se **deberá** subir a la plataforma un archivo texto con el código fuente del programa desarrollado y un reporte de la tarea en formato PDF con portada, desarrollo y conclusiones como mínimo. El archivo PDF deberá incluir las capturas de pantalla de la compilación y ejecución del programa, se deberá incluir la captura de pantalla correspondiente a **cada paso** de la creación de la máquina virtual. No se admitirá la tarea si no incluye las pantallas correspondientes a cada paso del procedimiento de creación de la máquina virtual.

El nombre de la máquina virtual deberá ser el número de boleta del alumno, si el número de boleta del alumno es 12345678, entonces la máquina virtual deberá llamarse: B12345678. **No se admitirá la tarea** si la máquina virtual no se nombra como se indicó anteriormente.

Recuerden que deben **eliminar la máquina virtual** cuando no la usen, con la finalidad de ahorrar el saldo de sus cuentas de Azure.

No se admitirá la tarea si se envían archivos en formato RAR o en formato WORD.

Valor de la tarea: 20% (1.2 puntos de la segunda evaluación parcial)



## Clase del día – 14/04/2021

En la tarea 3 desarrollamos un programa que multiplica matrices cuadradas en forma distribuida usando paso de mensajes.

Como pudimos ver, la programación de un sistema distribuido utilizando el paso de mensajes es complicada, ya que se debe controlar explícitamente la serialización, el envío y la des-serialización de los datos, además de la lógica particular del sistema.

En la clase de hoy vamos a ver cómo desarrollar un programa distribuido que calcule el producto de matrices cuadradas utilizando Java RMI.

### Partición de los datos

Dadas las matrices A y B de tamaño  $N \times N$ , el producto  $C = A \times B$  se obtiene dividiendo la matriz A en las matrices A1 y A2, y dividiendo la **transpuesta** de la matriz B en las matrices B1 y B2.

El tamaño de las matrices A1, A2, B1 y B2 es  $N/2$  renglones y N columnas

Entonces, la matriz C se compone de las matrices C1, C2, C3 y C4, tal como se muestra en la siguiente figura:



Las matrices C1, C2, C3 y C4 se calculan de la siguiente manera:

$$C1 = A1 \times B1$$

$$C2 = A1 \times B2$$

$$C3 = A2 \times B1$$

$$C4 = A2 \times B2$$

### Multiplicación de matrices utilizando objetos distribuidos

Para multiplicar matrices utilizando objetos distribuidos, escribiremos un servidor RMI que ejecute un método remoto llamado `multiplica_matrices()`, este método recibe como parámetros dos matrices de tamaño  $(N/2) \times N$  y regresa como resultado una matriz cuadrada de tamaño  $(N/2) \times (N/2)$ .

Ahora debemos escribir un cliente RMI que inicialice las matrices, transponga la matriz B, invoque el método `multiplica_matrices()` y acomode las matrices C1, C2, C3 y C4 para formar la matriz C.

Consideremos el método `multiplica_matrices()` (este método ejecutará en el servidor RMI):

```
public int[][] multiplica_matrices(int[][] A, int[][] B) throws RemoteException
{
    int[][] C = new int[N/2][N/2];
    for (int i = 0; i < N/2; i++)
        for (int j = 0; j < N/2; j++)
            for (int k = 0; k < N; k++)
                C[i][j] += A[i][k] * B[j][k];
    return C;
}
```

Cuando el método `multiplica_matrices()` se invoca localmente, recibe como parámetros las referencias a las matrices A y B y regresa una referencia a la matriz C.

Cuando el método es invocado en forma remota, entonces la capa RMI serializa las matrices A y B en el cliente y las des-serializa en el servidor. De la misma forma, la capa RMI serializa la matriz C en el servidor y la des-serializa en el cliente.

Ahora veamos el método `separa_matriz()` el cual utilizaremos para obtener las matrices A1, A2, B1 y B2:

```
static int[][] separa_matriz(int[][] A, int inicio)
{
    int[][] M = new int[N/2][N];
    for (int i = 0; i < N/2; i++)
        for (int j = 0; j < N; j++)
            M[i][j] = A[i + inicio][j];
    return M;
}
```

El método `separa_matriz()` recibe como parámetros la matriz a dividir y el renglón inicial. El método regresará una matriz de tamaño  $(N/2) \times N$ .

Entonces, podemos obtener las matrices A1, A2, B1 y B2 de la siguiente manera:

```
int[][] A1 = separa_matriz(A,0);
int[][] A2 = separa_matriz(A,N/2);
int[][] B1 = separa_matriz(B,0);
int[][] B2 = separa_matriz(B,N/2);
```

Dadas las matrices A1, A2, B1 y B2, podemos obtener las matrices C1, C2, C3 y C4 utilizando el método `multiplica_matrices()`:

```
int[][] C1 = multiplica_matrices(A1,B1);
int[][] C2 = multiplica_matrices(A1,B2);
int[][] C3 = multiplica_matrices(A2,B1);
int[][] C4 = multiplica_matrices(A2,B2);
```

Veamos ahora el método `acomoda_matriz()`, el cual permite construir la matriz C a partir de las matrices C1, C2, C3 y C4:



```
static void acomoda_matriz(int[][] C,int[][] A,int renglon,int columna)
{
    for (int i = 0; i < N/2; i++)
        for (int j = 0; j < N/2; j++)
            C[i + renglon][j + columna] = A[i][j];
}
```

El método `acomoda_matriz()` recibe como parámetros la matriz `C`, la sub-matriz a acomodar, y la posición (renglón,columna) en la matriz `C` donde se va a colocar la sub-matriz.

Finalmente para obtener la matriz `C` podemos hacer lo siguiente:

```
int[][] C = new int[N][N];
acomoda_matriz(C,C1,0,0);
acomoda_matriz(C,C2,0,N/2);
acomoda_matriz(C,C3,N/2,0);
acomoda_matriz(C,C4,N/2,N/2);
```

## Actividades individuales a realizar

Desarrollar un programa en Java que multiplique dos matrices cuadradas de tamaño  $N \times N$  en forma local, utilizando las funciones `separa_matriz()`, `multiplica_matrices()` y `acomoda_matriz()`.

Clase del día - 16/04/2021



En la clase de hoy vamos a jugar un kahoot en la modalidad de "challenge".

Para jugar el kahoot deberán ingresar al siguiente enlace:

[Desarrollo de sistemas distribuidos - Objetos distribuidos con Java RMI](#)

Es necesario que los alumnos y alumnas ingresen su "nickname" como su nombre y apellido (por ejemplo JuanLopez), de manera que sea posible identificar a los ganadores de puntos extra.

La hora límite para jugar este kahoot es 11:00 PM del 16 de abril.



### Tarea 6. Multiplicación de matrices utilizando objetos distribuidos



Cada alumno deberá desarrollar un sistema que calcule el producto de dos matrices cuadradas utilizando Java RMI, tal como se explicó en clase.

Se deberá ejecutar dos casos:

$N=8$ , se deberá desplegar las matrices `A`, `B` y `C` y el checksum de la matriz `C`.

$N=1000$ , deberá desplegar el checksum de la matriz `C`.

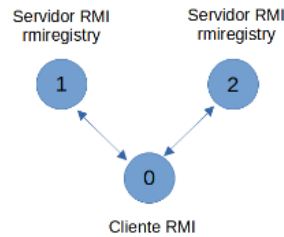
Los elementos de las matrices `A`, `B` y `C` serán de tipo `float` y el checksum será de tipo `double`.

Se deberá inicializar las matrices `A` y `B` de la siguiente manera (notar que la inicialización es diferente a la que se realizó en la tarea 3):

$A[i][j] = i - 2 * j$

$B[i][j] = i + 2 * j$

El servidor RMI ejecutará en dos máquinas virtuales (nodo 1 y nodo 2) con **Ubuntu** en Azure. El programa `rmiregistry` ejecutará en cada nodo donde ejecute el servidor RMI. El nodo 1 calculará los productos `C1` y `C2` mientras que el nodo 2 calculará los productos `C3` y `C4`.



El cliente RMI ejecutará en una tercera máquina virtual con **Ubuntu** (nodo 0). El cliente RMI inicializará las matrices A y B, obtendrá la transpuesta de la matriz B, invocará el método remoto `multiplica_matrices()`, calculará el checksum de la matriz C, y en su caso ( $N=8$ ) desplegará las matrices A, B y C.

Se deberá utilizar las funciones que vimos en clase: `separa_matriz()`, `multiplica_matrices()` y `acomoda_matriz()`.

Se **deberá** subir a la plataforma un archivo texto con el código fuente del programa desarrollado y un reporte de la tarea en formato PDF con portada, desarrollo y conclusiones como mínimo. El archivo PDF deberá incluir las capturas de pantalla de la compilación y ejecución del programa, se deberá incluir la captura de pantalla correspondiente a **cada paso** de la creación de las máquinas virtuales. No se admitirá la tarea si no incluye las pantallas correspondientes a cada paso del procedimiento de creación de las máquinas virtuales.

El nombre de cada máquina virtual deberá ser el número de boleta del alumno, un guión y el número de nodo, por ejemplo, si el número de boleta del alumno es 12345678, entonces el nodo 0 deberá llamarse: 12345678-0, el nodo 1 deberá llamarse 12345678-1, y así sucesivamente. **No se admitirá la tarea** si los nodos no se nombran como se indicó anteriormente.

Valor de la tarea: 30% (1.8 puntos de la segunda evaluación parcial)

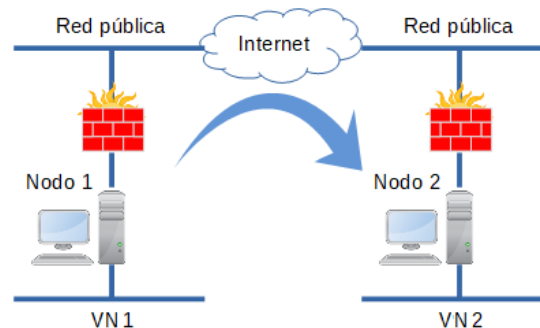
Clase del día - 20/04/2021



El día de hoy vamos a explicar algunos conceptos que serán de utilidad para realizar la tarea 6.

### Red privada y red pública

Supongamos que creamos dos máquinas virtuales en Azure, cada máquina virtual en un grupo de recursos diferente, esto implica que cada máquina virtual estará conectada a una red virtual (VN) diferente, tal como se muestra en la siguiente figura:



El firewall de una máquina virtual se puede configurar con reglas de entrada y reglas de salida, las reglas de entrada definen qué direcciones públicas y qué puertos se pueden conectar a la máquina virtual, mientras que las reglas de salida definen a qué direcciones públicas y a qué puertos se puede conectar la máquina virtual.

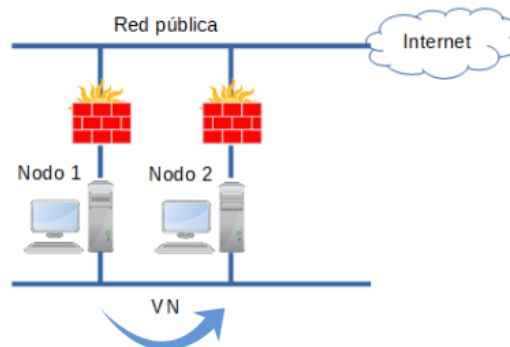
Por seguridad de la máquina virtual, las reglas de entrada suelen ser más restrictivas que las reglas de salida.

En este caso, para que el Nodo-1 se pueda conectar al Nodo-2, solo necesitamos crear una regla de entrada que

permita que el Nodo-1 se conecte a través de un puerto específico.

Por otra parte, debido a que las redes virtuales VN1 y VN2 están desconectadas, no es posible conectar el Nodo-1 y el Nodo-2 utilizando las direcciones IP privadas.

Ahora supongamos que **creamos dos máquinas virtuales en el mismo grupo de recursos**. En este caso las dos máquinas virtuales comparten la misma red virtual (VN).



Si el Nodo-1 requiere comunicarse con el Nodo-2 no es necesario crear una regla en el firewall del Nodo-2 ya que ambos nodos están conectados a través de la misma red virtual.

Notar que la comunicación entre las máquinas virtuales mediante la VN se realiza utilizando las direcciones IP privadas de las máquinas virtuales.

### ¿Cómo ejecutar Java RMI en la nube?

Para registrar un objeto remoto en el rmiregistry utilizamos el método `rebind()`.

Debido a que el servidor RMI debe ejecutar en la misma computadora donde ejecuta rmiregistry, la URL que pasa como parámetro al método `rebind()` deberá incluir el dominio **localhost**, tal como se muestra en el siguiente ejemplo:

```
public class ServidorRMI
{
    public static void main(String[] args) throws Exception
    {
        String url = "rmi://localhost/prueba";
        ClaseRMI obj = new ClaseRMI();

        // registra la instancia en el rmiregistry
        Naming.rebind(url,obj);
    }
}
```

Para que el cliente RMI pueda invocar los métodos del objeto remoto registrado por el servidor RMI, se debe obtener una referencia al objeto remoto utilizando el método `lookup()`. Entonces la URL que pasa como parámetro al método `lookup()` **deberá definir la IP privada** del nodo donde ejecuta el servidor RMI.

Supongamos que la dirección IP privada donde ejecuta el servidor RMI, es **10.0.2.4**:

```
public class ClienteRMI
{
    public static void main(String args[]) throws Exception
    {
        // en este caso el objeto remoto se llama "prueba", notar que se utiliza el puerto default 1099
        String url = "rmi://10.0.2.4/prueba";

        // obtiene una referencia que "apunta" al objeto remoto asociado a la URL
        InterfaceRMI r = (InterfaceRMI)Naming.lookup(url);
    }
}
```

```

System.out.println(r.mayusculas("hola"));
System.out.println("suma=" + r.suma(10,20));

int[][] m = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
System.out.println("checksum=" + r.checksum(m));
}
}

```

## Actividades individuales a realizar

1. Crear dos máquinas virtuales (Nodo-1 y Nodo-2) en el mismo grupo de recursos.
2. Compilar el programa ClienteRMI.java en el Nodo-1. Utilizar la IP privada del Nodo-2 en la URL.
3. Compilar el programa ServidorRMI.java en el Nodo-2. Utilizar localhost en la URL.
4. Ejecutar en el Nodo-2: `rmiregistry&`
5. Ejecutar en el Nodo-2: `java ServidorRMI&`
6. Ejecutar en el Nodo-1: `java ClienteRMI&`

**IMPORTANTE:** se debe eliminar las máquinas virtuales y todos sus recursos lo más pronto posible, ya que se deberá ahorrar saldo para poder realizar las tareas siguientes.

Clase del día – 21/04/2021



En la clase de hoy vamos a explicar cómo utilizar **JSON** para serializar y des-serializar objetos.

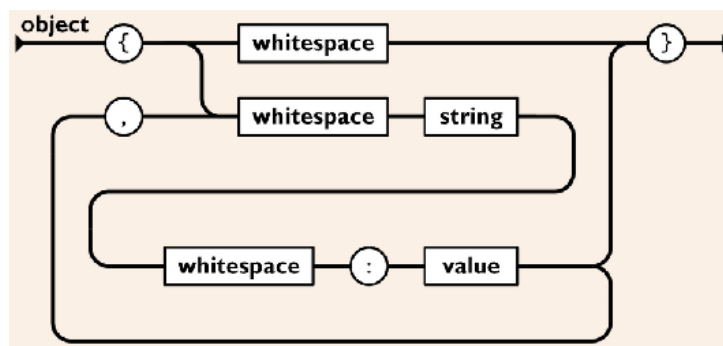
**JSON** (JavaScript Object Notation) es un formato texto para el intercambio de datos. JSON corresponde a la sintaxis utilizada en Javascript para escribir objetos.

JSON es un formato independiente del lenguaje de programación, de manera que es posible escribir fácilmente programas en cualquier lenguaje que creen mensajes en formato JSON así como programas que lean mensajes en formato JSON.

En JSON es posible crear dos estructuras: objetos y arreglos.

Un **objeto** es una colección no ordenada de parejas nombre:valor separadas por coma. Un objeto comienza con una llave que abre "{" y termina con una llave que cierra "}".

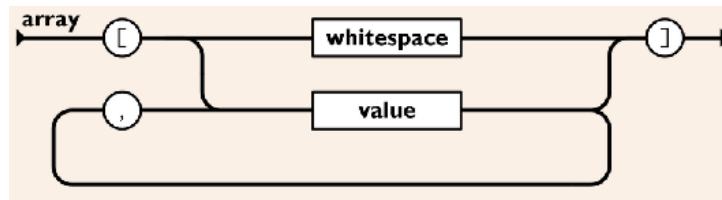
La sintaxis de un objeto es la siguiente:



Fuente: [www.json.org](http://www.json.org)

Un **arreglo** es una colección ordenada de valores separados por coma. Un arreglo comienza con un corchete que abre "[" y termina con un corchete que cierra "]".

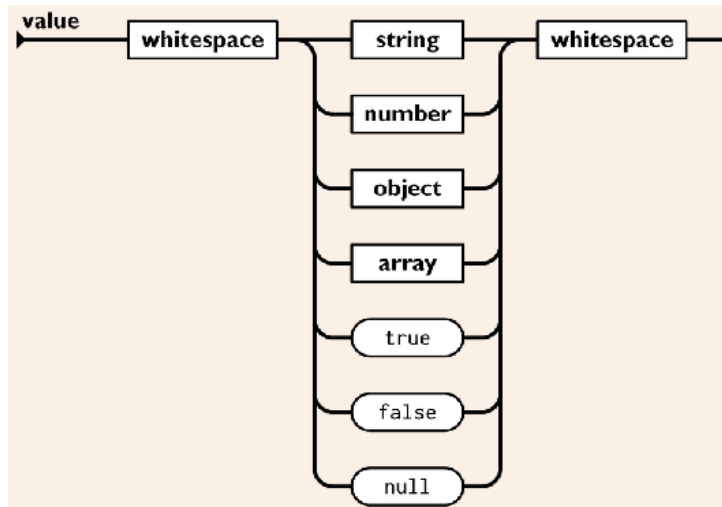
La sintaxis de un arreglo es la siguiente:



Fuente: [www.json.org](http://www.json.org)

Un **valor** puede ser una cadena de caracteres entre comillas, o un número, o un objeto, o un arreglo, o las constantes true, false o null.

La sintaxis de un valor es la siguiente:



Fuente: [www.json.org](http://www.json.org)

Una **cadena de caracteres** (string) es una secuencia de cero o más caracteres Unicode encerrados entre comillas.

Una cadena de caracteres puede contener las siguientes secuencias de escape:

-----  
Secuencia

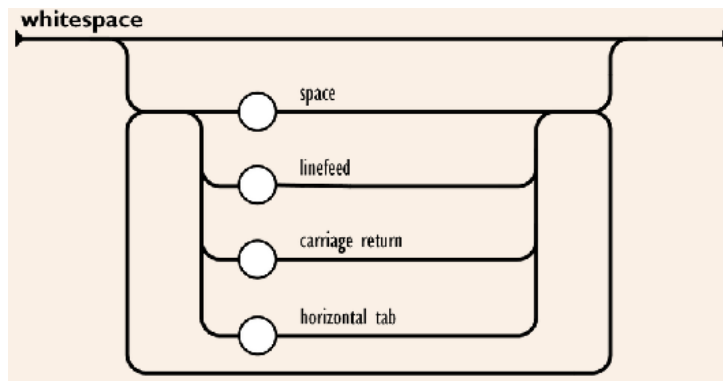
de escape Descripción

\"	comillas
\\	diagonal inversa
\/	diagonal
\b	back space
\f	form feed
\n	line feed
\r	carriage return
\t	tabulador
\uxxxx	caracter unicode con código hexadecimal xxxx

-----

Un **número** sigue la sintaxis de los números decimales en lenguaje C.

Un **whitespace** es un separador de tokens, de acuerdo a la siguiente sintaxis:



Fuente: [www.json.org](http://www.json.org)

Ahora veremos un ejemplo, utilizando GSON (implementación de JSON desarrollada por Google).

Se requiere descargar el archivo gson-2.8.6.jar de la siguiente URL:

<https://repo1.maven.org/maven2/com/google/code/gson/gson/2.8.6/gson-2.8.6.jar>

Descargar de la plataforma el programa `EjemploGSON.java` y colocarlo en el mismo directorio dónde está el archivo gson-2.8.6.jar que se descargó anteriormente.

Para compilar el programa `EjemploGSON.java` se debe ejecutar el siguiente comando:

```
javac -cp gson-2.8.6.jar EjemploGSON.java
```

Para ejecutar el programa en Windows:

```
java -cp gson-2.8.6.jar;. EjemploGSON
```

Para ejecutar el programa en Linux:

```
java -cp gson-2.8.6.jar:. EjemploGSON
```

Ahora se explicará como funciona este programa.

Primeramente se declaran los imports de las clases que se utilizarán: Gson, GsonBuilder y Timestamp.

Se define una clase `Empleado` que contiene los campos: nombre, edad, sueldo y fecha\_ingreso. Por conveniencia se define un constructor para inicializar los campos al crear una instancia.

Notar que la fecha se maneja como fecha-hora debido a que en los sistemas globales (Internet) la fecha no indica qué sucede antes y qué sucede después a nivel mundial; recordar lo que se vimos sobre tiempo UTC y tiempo local.

En la función main se crea un arreglo de 3 empleados, cada elemento se inicializa con una instancia de la clase `Empleado`, con diferentes datos.

Entonces se crea una instancia de la clase `Gson`. Notar que se utiliza el método `setDateFormat` para utilizar el formato de fecha ISO8601, el cual es el formato que utiliza Javascript para manejar fecha-hora.

Después se utiliza el método `toJson` de la clase `Gson` para serializar el arreglo de empleados. Esta clase produce la siguiente string:

```
[{"nombre":"Hugo","edad":20,"sueldo":1000.0,"fecha_ingreso":"2020-01-01T20:10:00.000"},
{"nombre":"Paco","edad":21,"sueldo":2000.0,"fecha_ingreso":"2019-10-01T10:15:00.000"},
{"nombre":"Luis","edad":22,"sueldo":3000.0,"fecha_ingreso":"2018-11-01T00:00:00.000"}]
```

Finalmente se utiliza el método **fromJson** de la clase Gson para deserializar la string anterior y producir un nuevo arreglo de empleados. Entonces se despliegan los datos de los empleados:

```
Hugo 20 1000.0 2020-01-01 20:10:00.0
Paco 21 2000.0 2019-10-01 10:15:00.0
Luis 22 3000.0 2018-11-01 00:00:00.0
```

## Actividades individuales a realizar

1. Compilar y ejecutar el programa [EjemploGSON.java](#)
2. Agregar un nuevo campo a la clase Empleado, el campo "jefe" de tipo Empleado.
3. Modificar el constructor de la clase Empleado para incluir la inicialización del campo "jefe", tal como se muestra:

```
static class Empleado
{
    String nombre;
    int edad;
    float sueldo;
    Timestamp fecha_ingreso;
    Empleado jefe;
    Empleado(String nombre,int edad,float sueldo,Timestamp fecha_ingreso,Empleado jefe)
    {
        this.nombre = nombre;
        this.edad = edad;
        this.sueldo = sueldo;
        this.fecha_ingreso = fecha_ingreso;
        this.jefe = jefe != null ? jefe : this;
    }
}
```

4. Crear los tres empleados de manera que Hugo sea jefe de si mismo (Hugo es el jefe máximo), Hugo sea jefe de Paco, y Paco sea jefe de Luis, tal como se muestra:

```
Empleado[] e = new Empleado[3];
e[0] = new Empleado("Hugo",20,1000,Timestamp.valueOf("2020-01-01 20:10:00"),null);
e[1] = new Empleado("Paco",21,2000,Timestamp.valueOf("2019-10-01 10:15:00"),e[0]);
e[2] = new Empleado("Luis",22,3000,Timestamp.valueOf("2018-11-01 00:00:00"),e[1]);
```

5. Cambiar el despliegue del arreglo deserializado, de la siguiente manera:

```
for (int i = 0; i < v.length; i++)
    System.out.println(v[i].nombre + " " + v[i].edad + " " + v[i].sueldo + " " + v[i].fecha_ingreso + " jefe:" + (v[i].jefe
!= null ? v[i].jefe.nombre : null));
```

6. Compilar y ejecutar el programa.
  - 6.1 ¿Qué despliega la serialización?
  - 6.2 Al serializar el empleado Hugo ¿Se muestra la auto-referencia que hace el empleado Hugo a sí mismo?
  - 6.3 ¿Hay alguna redundancia en los objetos serializados?
  - 6.4 ¿Qué despliega la deserialización?
  - 6.5 ¿Qué concluye sobre GSON y la forma en que serializa y deserializa las referencias a objetos?

7. Ver el video:



0:00 / 0:00

 EjemploGSON.java

Clase del día - 23/04/2021

En la clase de hoy vamos a jugar un kahoot en la modalidad de "challenge".

Para jugar el kahoot deberán ingresar al siguiente enlace:

#### [Desarrollo de Sistemas Distribuidos - JSON](#)

Es necesario que los alumnos y alumnas ingresen su "nickname" como su nombre y apellido (por ejemplo JuanLopez), de manera que sea posible identificar a los ganadores de puntos extra.

La hora límite para jugar este kahoot es 11:00 PM del 23 de abril.



Clase del día 27/04/2021

En la clase de hoy veremos los conceptos básicos de los servicios web (Web Services) así como los elementos de servicios web SOAP y REST.

#### **Conceptos básicos de Servicios web**

En el documento [Web Services Architecture](#) del World Wide Web Consortium (W2C) define un servicio web como:

"Un sistema de software diseñado para soportar la interacción interoperable de máquina-a-máquina sobre una red. Este cuenta con una interface descrita en un formato el cual puede ser procesado por una computadora (específicamente WSDL). Otros sistemas interactúan con el servicio web en una manera prescrita por su descripción usando mensajes SOAP, típicamente transportados usando HTTP con una serialización XML en conjunción con otros estándares relativos a la Web".

Un servicio web es un concepto abstracto que debe ser implementado mediante un agente concreto.

Un **agente** es el software o hardware que envía y recibe mensajes. El **servicio** es el recurso caracterizado por un conjunto abstracto de la funcionalidad que se provee. Un servicio web no cambia aún cuando cambie el agente, es decir, la funcionalidad es independiente de la implementación de ésta.

El propósito de servicio web es proveer cierta funcionalidad a nombre de su propietario (una persona o una organización). La **entidad proveedora** es aquella persona u organización que provee un agente que implementa un determinado servicio.

Una **entidad solicitante** es una persona u organización que desea hacer uso del servicio mediante un **agente solicitante** (también llamado *solicitante del servicio*) que intercambia mensajes con el **agente proveedor** (también llamado *proveedor del servicio*).

En la mayoría de los casos el agente solicitante es el que inicia la comunicación con el agente proveedor, aunque no siempre es así, no obstante se sigue llamando agente solicitante aunque no sea el que inicia la comunicación.



La **semántica** de un servicio web es la expectativa compartida sobre el comportamiento del servicio, en particular el comportamiento en respuesta a los mensajes que recibe.

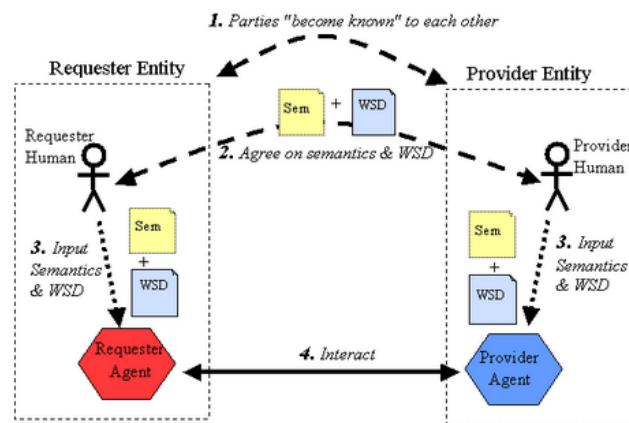
Se le llama **contrato** al acuerdo entre la entidad solicitante y la entidad proveedora. Un contrato puede ser explícito o implícito, escrito u oral, establecido entre las personas y/o las computadoras, legal o informal.

Hay dos tipos de contratos: 1) la *descripción del servicio* es el contrato que gobierna la mecánica de interacción con un servicio en particular y 2) la *semántica* del servicio es el contrato que gobierna el significado y propósito de la interacción. Sin embargo, puede haber contratos "híbridos" que incluyan elementos de descripción y elementos de semántica.

### Participación en un servicio web

Una entidad solicitante puede participar de un servicio web de diferentes maneras. La siguiente figura muestra el proceso general de participación en un servicio web.

1. Las entidades solicitante y proveedora se conocen una a la otra, o por lo menos una conoce a la otra.
2. Las entidades acuerdan la descripción (WSD: Web Service Description) y semántica del servicio.
3. La descripción y la semántica son implementadas por el agente solicitante y el agente proveedor.
4. Los agentes solicitante y proveedor intercambian mensajes.



Fuente: Web Services Architecture, W3C

### Servicios web basados en SOAP

SOAP (Simple Object Access Protocol) define un protocolo de RPC (Remote Procedure Call) basado en XML, para la interacción cliente-servidor a través de la red utilizando: 1) HTTP como la base de transporte, y 2) documentos XML para la codificación de requerimientos y respuestas.

SOAP permite la comunicación entre aplicaciones ejecutando en diferentes sistemas operativos, con diferentes tecnologías y lenguajes de programación.

### Componentes de un mensaje SOAP

Un mensaje SOAP es un documento XML compuesto por los siguientes elementos:

- Envelope (sobre). Identifica el documento XML como un mensaje SOAP.
- Header (encabezado). Contiene información de encabezado.
- Body (cuerpo). Contiene información del requerimiento y la respuesta.
- Fault (falla). Contiene errores e información de estatus.

Para implementar servicios web en Java se puede utilizar la API JAX-WS.

### Web Services Description Language (WSDL)

Un documento WSDL es un documento XML que contiene la descripción de un servicio web SOAP. Este especifica la localización del servicio y los métodos del servicio.

Un cliente puede hacer un requerimiento HTTP a un servicio web SOAP para obtener el WSDL que describe el servicio web.

Los elementos de un documento WSDL son los siguientes:

Elemento	Descripción
<types>	Define los tipos de dato usados por el servicio web
<message>	Define los elementos de datos para cada operación
<servicioportType>	Describe las operaciones que pueden ser ejecutadas y los mensajes involucrados
<binding>	Define el protocolo y el formato de los datos para cada portType

Un ejemplo de WSDL es el siguiente (fuente [www.w3schools.com/xml/xml\\_wsdl.asp](http://www.w3schools.com/xml/xml_wsdl.asp)):

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>
<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>
<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
<binding type="glossaryTerms" name="b1">
  <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
  <operation>
    <soap:operation soapAction="http://example.com/getTerm"/>
    <input> <soap:body use="literal"/> </input>
    <output> <soap:body use="literal"/> </output>
  </operation>
</binding>
```

El siguiente código, es un ejemplo de un servicio web escrito en Java utilizando el API JAX-WS:

```
package negocio;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.naming.InitialContext;
import javax.sql.DataSource;
import java.sql.Connection;

@WebService
public class ServicioSOAP
{
    static DataSource pool;
    static
    {
        pool = null;
        try
        {
            pool = (DataSource)new InitialContext().lookup("java:comp/env/jdbc/prueba");
        }
    }
}
```

```

        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    @WebMethod
    public Integer suma(@WebParam(name = "a") Integer a, @WebParam(name = "b") Integer b) throws Exception
    {
        return a + b;
    }
    @WebMethod
    public String mayusculas(@WebParam(name = "s") String s) throws Exception
    {
        return s.toUpperCase();
    }
    @WebMethod
    public void prueba_conexion_bd() throws Exception
    {
        Connection conexion = pool.getConnection();
        conexion.close();
    }
}

```

En este ejemplo podemos ver que la anotación `@WebService` define la clase correspondiente al servicio web. Cada operación del servicio web se implementa como un método de Java incluyendo la anotación `@WebMethod`. La anotación `@WebParam` se utiliza para definir los parámetros de cada operación del servicio web.

En este caso el servicio web accede a una base de datos llamada "prueba" mediante JDBC.

### Servicios web estilo REST

REST define un conjunto de principios arquitectónicos para la creación de servicios web. REST fue presentado por Roy Fielding el año 2000 en su disertación doctoral "Architectural Styles and the Design of Network-based Software Architectures".

El diseño de servicios web estilo REST sigue cuatro principios:

- Utilizar métodos HTTP de forma explícita.
  - Un servicio web utiliza los métodos de HTTP para crear un recurso (POST), leer (GET), cambiar el estado o actualizar un recurso (PUT), y borrar un recurso (DELETE).
- Los servicios son sin estado (stateless).
  - Los clientes de servicios web estilo REST deben enviar peticiones completas e independientes, es decir, las peticiones deben incluir todos los datos que permitan completar el servicio, sin la necesidad de guardar un estado entre peticiones.
- Los URIs representan una estructura de directorios.
  - Los URIs (Uniform Resource Identifier) deben ser intuitivos y auto-explicados. Un URI es una jerarquía que corresponde a la estructura de los servicios web definidos en la empresa.
- Se transfiere XML, JSON o ambos.
  - Los recursos que provee un servicio web pueden ser documentos, imágenes, videos y en general objetos. La representación de objetos mediante XML o JSON es fácil e independiente de la plataforma.

### Actividades individuales a realizar

Ver el video:



0:00 / 0:00



## Clase del día - 28/04/2021

La clase de hoy vamos realizar una práctica la cual se entregará como tarea 7.

Veremos cómo crear un servicio web estilo REST utilizando el API de Java JAX-RS sobre el servidor de aplicaciones Tomcat.

Primeramente instalaremos Tomcat y las bibliotecas necesarias para la implementación de servicios web estilo REST los cuales podrán acceder una base de datos MySQL.

### Instalación de Tomcat con soporte REST

1. Crear una máquina virtual con Ubuntu 18 con al menos 1GB de memoria RAM. Abrir el puerto 8080 para el protocolo TCP.
2. Instalar JDK8 ejecutando los siguientes comandos en la máquina virtual:

```
sudo apt update
```

```
sudo apt install openjdk-8-jdk-headless
```

3. Descargar la distribución binaria de Tomcat 8 de la siguiente URL (descargar la opción Core "zip"):  
<https://tomcat.apache.org/download-80.cgi>

4. Copiar a la máquina virtual el archivo ZIP descargado anteriormente y desempacarlo utilizando el comando unzip.

5. Eliminar el directorio webapps el cual se encuentra dentro del directorio de Tomcat. Crear un nuevo directorio webapps y dentro de éste se deberá crear el directorio ROOT.

NOTA DE SEGURIDAD: Lo anterior se recomienda debido a que se han detectado vulnerabilidades en algunas aplicaciones que vienen con Tomcat, estas aplicaciones se encuentran originalmente instaladas en los directorios webapps y webapps/ROOT.

6. Descargar la biblioteca "Jersey" de la siguiente URL. Jersey es una implementación de JAX-RS lo cual permite ejecutar servicios web estilo REST sobre Tomcat:

<https://repo1.maven.org/maven2/org/glassfish/jersey/bundles/jaxrs-ri/2.24/jaxrs-ri-2.24.zip>

7. Copiar a la máquina virtual el archivo descargado anteriormente, desempacarlo y **copiar todos los archivos** con extensión ".jar" de **todos los directorios** desempacados, al directorio "lib" de Tomcat.

8. Borrar el archivo javax.servlet-api-3.0.1.jar del directorio "lib" de Tomcat (esto debe hacerse ya que existe una incompatibilidad entre Tomcat y Jersey 2).

9. Descargar el archivo gson-2.3.1.jar de la URL:

<https://repo1.maven.org/maven2/com/google/code/gson/gson/2.3.1/gson-2.3.1.jar>

10. Copiar el archivo gson-2.3.1.jar al directorio "lib" de Tomcat.

11. Ahora vamos a instalar el driver de JDBC para MySQL. Ingresar a la siguiente URL:

<https://dev.mysql.com/downloads/connector/j/>

Seleccionar "Platform independent" y descargar el archivo ZIP.

12. Copiar el archivo descargado a la máquina virtual, desempaquetarlo y copiar el archivo mysql-connector...jar al directorio "lib" de Tomcat.

### Iniciar/detener el servidor Tomcat

1. Para iniciar el servidor Tomcat es **necesario** definir las siguientes variables de entorno:

```
export CATALINA_HOME=aquí va la ruta del directorio de Tomcat 8
```

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

2. Iniciar la ejecución de Tomcat ejecutando el siguiente comando:

```
sh $CATALINA_HOME/bin/catalina.sh start
```

3. Para detener la ejecución de Tomcat se deberá ejecutar el siguiente comando:

```
sh $CATALINA_HOME/bin/catalina.sh stop
```

Notar que Tomcat se ejecuta sin permisos de administrador (no se usa "sudo"), lo cual es muy importante para prevenir que algún atacante pueda entrar a nuestro sistema con permisos de super-usuario.

### Instalación de MySQL

1. Actualizar los paquetes en la máquina virtual ejecutando el siguiente comando:

```
sudo apt update
```

2. Instalar el paquete default de MySQL:

```
sudo apt install mysql-server
```

3. Ejecutar el script de seguridad:

```
sudo mysql_secure_installation
```

Press y|Y for Yes, any other key for No: **N**

New password: *contraseña-de-root-en-mysql*

Re-enter new password: *contraseña-de-root-en-mysql*

Remove anonymous users? (Press y|Y for Yes, any other key for No) : **Y**

Disallow root login remotely? (Press y|Y for Yes, any other key for No) : **Y**

Remove test database and access to it? (Press y|Y for Yes, any other key for No) : **Y**

Reload privilege tables now? (Press y|Y for Yes, any other key for No) : **Y**

4. Ejecutar el monitor de MySQL:

```
sudo mysql
```

5. Ejecutar el siguiente comando SQL para modificar la contraseña de root:

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'contraseña-de-root-en-mysql';
```

6. Actualizar los privilegios:

```
FLUSH PRIVILEGES;
```

7. Ejecutar el siguiente comando para salir del monitor de MySQL:

```
quit
```

### Crear un usuario en MySQL

1. Ejecutar el monitor de MySQL:

```
mysql -u root -p
```

2. Crea el usuario "hugo":

```
create user hugo@localhost identified by 'contraseña-del-usuario-hugo';
```

3. Otorgar todos los permisos al usuario "hugo" sobre la base de datos "servicio\_web":

```
grant all on servicio_web.* to hugo@localhost;
```

4. Ejecutar el siguiente comando para salir del monitor de MySQL:

```
quit
```

### Crear la base de datos

1. Ejecutar el monitor de MySQL (notar que ahora se utiliza el usuario "hugo"):

```
mysql -u hugo -p
```

2. Crear la base de datos "servicio\_web":

```
create database servicio_web;
```

3. Conectar a la base de datos creada anteriormente:

```
use servicio_web;
```

4. Crear las tablas "usuarios" y "fotos\_usuarios", así mismo, se crea una regla de integridad referencial y un índice único:

```
create table usuarios
(
    id_usuario integer auto_increment primary key,
    email varchar(256) not null,
    nombre varchar(100) not null,
    apellido_paterno varchar(100) not null,
    apellido_materno varchar(100),
    fecha_nacimiento date not null,
    telefono varchar(20),
    genero char(1)
);
create table fotos_usuarios
(
    id_foto integer auto_increment primary key,
    foto longblob,
    id_usuario integer not null
);
alter table fotos_usuarios add foreign key (id_usuario) references usuarios(id_usuario);
create unique index usuarios_1 on usuarios(email);
```

5. Salir del monitor de MySQL:

```
quit
```

### Compilar, empacar y desplegar el servicio web

1. Descargar de la plataforma y desempacar el archivo [Servicio.zip](#).

2. Definir la variable de ambiente CATALINA\_HOME:

```
export CATALINA_HOME=aquí va la ruta completa del directorio de Tomcat 8
```

3. Cambiar al directorio dónde se desempacó el archivo [Servicio.zip](#) (en ese directorio se encuentra el directorio "negocio").

4. Compilar la clase Servicio.java:

```
javac -cp $CATALINA_HOME/lib/javax.ws.rs-api-2.0.1.jar:$CATALINA_HOME/lib/gson-2.3.1.jar:.  
negocio/Servicio.java
```

5. Editar el archivo "context.xml" que está en el directorio "META-INF" y definir el username de la base de datos y el password correspondiente. El usuario "hugo" fue creado en el paso 2 de la sección **Crear un usuario en MySQL**.

6. Ejecutar los siguientes comandos para crear el servicio web para Tomcat (notar que los servicios web para Tomcat son archivos JAR con la extensión .war):

```
rm WEB-INF/classes/negocio/*
```

```
cp negocio/*.class WEB-INF/classes/negocio/.
```

```
jar cvf Servicio.war WEB-INF META-INF
```

7. Para desplegar (*deploy*) el servicio web, copiar el archivo **Servicio.war** al directorio "webapps" de Tomcat. Notar que Tomcat desempaca automáticamente los archivos con extensión .war que se encuentran en el directorio webapps de Tomcat.

Para eliminar el servicio web se deberá eliminar el archivo "Servicio.war" y el directorio "Servicio", en éste orden.

Cada vez que se modifique el archivo Servicio.java se deberá compilar, generar el archivo Servicio.war, borrar el archivo Servicio.war y el directorio Servicio del directorio webapps de Tomcat, y copiar el archivo Servicio.war al directorio webapps de Tomcat.

### Probar el servicio web utilizando HTML-Javascript

1. Copiar el archivo [usuario\\_sin\\_foto.png](#) al subdirectorio webapps/ROOT de Tomcat.

Notar que todos los archivos que se encuentran en el directorio webapps/ROOT de Tomcat son accesibles públicamente.

Para probar que Tomcat esté en línea y el puerto 8080 esté abierto, ingresar la siguiente URL en un navegador:

```
http://ip-de-la-máquina-virtual:8080/usuario_sin_foto.png
```

2. Copiar el archivo [WSClient.js](#) al directorio webapps/ROOT de Tomcat.

3. Copiar el archivo [prueba.html](#) al directorio webapps/ROOT de Tomcat.

4. Ingresar la siguiente URL en un navegador:

```
http://ip-de-la-máquina-virtual:8080/prueba.html
```

5. Dar clic en el botón "Alta usuario" para dar de alta un nuevo usuario. Capturar los campos y dar clic en el botón "Alta".

6. Intentar dar de alta otro usuario con el mismo email (se deberá mostrar una ventana de error indicando que el email ya existe)

7. Dar clic en el botón "Consulta usuario" para consultar el usuario dado de alta en el paso 5. Capturar el email y dar clic en el botón "Consulta",

8. Modificar algún dato del usuario y dar clic en el botón "Modifica":

9. Recargar la página actual y consultar el usuario modificado, para verificar que la modificación se realizó.
10. Dar clic en el botón "Borra usuario" para borrar el usuario. Capturar el email del usuario a borrar y dar clic en el botón "Consulta".

## Actividades individuales a realizar

Utilizando un teléfono inteligente y/o una tableta, probar el servicio web accediendo a la siguiente URL en un navegador (Chrome, Firefox, Opera, Safari, etc):

<http://ip-de-la-máquina-virtual:8080/prueba.html>

	<a href="#">Servicio.zip</a>	
	<a href="#">prueba.html</a>	
	<a href="#">WSClient.js</a>	
	<a href="#">usuario_sin_foto.png</a>	
	<a href="#">Tarea 7. Implementación de un servicio web estilo REST</a>	

Cada alumno ejecutará el procedimiento que vimos en clase, dónde instalamos Tomcat, instalamos MySQL, y creamos un servicio web estilo REST.

Se deberá probar el servicio web utilizando la aplicación web [prueba.html](#) tal como se explicó en clase.

Se **deberá** subir a la plataforma el código fuente de los programas y un reporte de la tarea en formato PDF con portada, desarrollo y conclusiones como mínimo.

El reporte PDF deberá incluir las capturas de pantalla de la compilación y ejecución del programa, se deberá incluir la captura de pantalla correspondiente a **cada paso** de la creación de la máquina virtual.

No se admitirá la tarea si no incluye las pantallas correspondientes a cada paso del procedimiento de creación de la máquina virtual.

El nombre de la máquina virtual deberá ser el número de boleta del alumno, si el número de boleta del alumno es 12345678, entonces la máquina virtual deberá llamarse: A12345678. **No se admitirá la tarea** si la máquina virtual no se nombra como se indicó anteriormente.

**La captura de pantallas deberá estar completa**, no se admitirá la tarea si incluye imágenes que sean cortes de las capturas de pantalla.

Recuerden que deben **eliminar la máquina virtual** cuando no la usen, con la finalidad de ahorrar el saldo de sus cuentas de Azure.

No se admitirá la tarea si se envía en formato RAR o en formato WORD.

Valor de la tarea: 20% (1.2 punto de la segunda evaluación parcial)

## Clase del día - 30/04/2021



La clase de hoy explicaremos cómo funciona el servicio estilo REST que creamos la clase anterior.

En la plataforma se publicó los siguientes archivos:

- [Servicio.zip](#) contiene el código Java y archivos de configuración de un servicio web estilo REST.
- [prueba.html](#) contiene una aplicación web que invoca el servicio web mediante Javascript.
- [WSClient.js](#) funciones para invocar el servicio web mediante AJAX



- **usuario\_sin\_foto.png** imagen que se despliega cuando el usuario no tiene foto.

El archivo [Servicio.zip](#) contiene los siguientes directorios:

- META-INF
- negocio
- WEB-INF

El directorio META-INF contiene el archivo **context.xml**, en el cual se configura lo siguiente:

El atributo **name** de la etiqueta **Resource**, define el nombre del datasource, en este caso el datasource se llama "jdbc/datasource\_Servicio".

Un datasource permite configurar las conexiones que realiza el servicio web, sin tener que escribir estos parámetros de configuración en el código (por ejemplo el nombre y contraseña del usuario de la base de datos).

El atributo **url** define el nombre de la base de datos, en este caso la base de datos se llama "servicio\_web", el atributo **username** define el nombre del usuario de la base de datos y el atributo **password** define la contraseña del usuario.

El directorio WEB-INF contiene el directorio **classes** y el archivo **web.xml**.

El archivo **web.xml** configura lo siguiente:

La etiqueta **load-on-startup** igual a 1 indica que el servicio web se debe cargar cuando inicie el servidor de aplicaciones Tomcat.

La etiqueta **url-pattern** indica la ruta del servicio web. La URL del servicio web se explicará más adelante.

La etiqueta **resource-ref**, indica el nombre del datasource y la etiqueta **res-type** indica el tipo de recurso javax.sql.DataSource.

En el directorio **classes** se colocarán las clases compiladas del servicio web (archivos .class). En este caso el directorio incluye un subdirectorio llamado "negocio", debido a que las clases del servicio web se agrupan en un paquete llamado "negocio".

Al mismo nivel de los directorios META-INF y WEB-INF se encuentra un directorio llamado **negocio** dónde se puede encontrar los archivos que contienen el código fuente del servicio web.

Los archivos incluidos en el directorio **negocio** son los siguientes:

#### **AdaptadorGsonBase64.java**

La clase AdaptadorGsonBase64 permite modificar la forma en que GSON serializa los campos con tipo byte[].

Por omisión GSON convierte un campo de tipo byte[] en una lista de números separados por comas, lo cual ocupa mucho espacio (y tiempo en la comunicación).

La clase AdaptadorGsonBase64 permite convertir los campos de tipo byte[] a base 64, lo cual produce un texto más compacto.

Notar que jax-rs reemplaza cada "+" por espacio, sin embargo el decodificador Base64 no reconoce el espacio, por lo que es necesario reemplazar los espacios por "+".

#### **Error.java**

La clase Error va a permitir regresar un mensaje de error dentro de un objeto.

#### **Foto.java**

La clase Foto encapsula un arreglo de bytes y una clave numérica. El servicio web utiliza esta clase para enviar y recibir imágenes.

#### **Usuario.java**

La clase Usuario encapsula los datos del usuario. El servicio web utiliza esta clase para recibir los datos del usuario como un objeto.

Es muy importante notar que la anotación `@FormParam` requiere un método que convierta una String a objeto de tipo Usuario. En este caso se implementa el método `valueOf` para este propósito.

### Servicio.java

La clase Servicio implementa los métodos del servicio web.

### URL del servicio web

La URL del servicio web se compone de cuatro elementos:

1. Dominio y el puerto, por ejemplo **localhost:8080**
2. Nombre del archivo .war, en este caso **Servicio**
3. Ruta definida en la etiqueta url-pattern en el archivo web.xml, en este caso la ruta: **rest**
4. Ruta definida en la anotación `@Path` de la clase Servicio, en este caso se define: **ws**

En este caso la URL completa sería: `http://localhost:8080/Servicio/rest/ws`

### Pool de conexiones

Con frecuencia cuándo un servicio web recibe una petición (requerimiento), abre una conexión a la base de datos. Sin embargo realizar una conexión a la base de datos es lento. Por tanto, es conveniente que los servicio web utilicen un pool de conexiones mediante un datasource.

Como se dijo anteriormente, un datasource permite configurar los parámetros de conexión a la base de datos fuera del código del servicio web. No es buena práctica escribir el nombre de usuario de base de datos y la contraseña en el código del servicio web, ya que habría que editar y recompilar el código cada vez que se cambie la contraseña del usuario.

Así mismo, el uso de datasource y pool de conexiones permite establecer un número de conexiones a la base de datos y mantenerlas abiertas siempre. Cada vez que un método del servicio web requiera conectarse a la base de datos, solicita una conexión disponible en el pool.

Cuándo el método termina deberá liberar la conexión invocando el método "close", no obstante, este método no cierra la conexión sino que la regresa al pool de conexiones, para su posterior re-uso.

Es muy importante considerar que cualquier configuración que el método haga a la conexión (por ejemplo activar o desactivar el autocommit) quedará establecida en la conexión, de manera que si otro método re-usa la conexión, esta conexión tendrá una configuración previa.

Debido a lo anterior, un método que obtiene una conexión del pool de conexiones deberá re-configurar la conexión de acuerdo a sus propósitos, y no suponer que la conexión tiene una configuración determinada.

En el caso de la clase Servicio, el pool de conexiones se crea en la inicialización estática de la clase. Notar que el datasource se identifica con el nombre que se le dio en el archivo context.xml

### El método POST

Como vimos en la clase anterior, un servicio web de estilo REST utiliza los métodos GET, POST, DELETE y PUT para realizar operaciones determinadas. En este caso, el servicio web que implementamos se aparta un poco del estándar ya que TODOS los métodos web utilizan el método POST, por esta razón todos los métodos de la clase Servicio incluyen la anotación `@POST`.

### El servicio web produce JSON

El el código de la clase Servicio puede observarse que cada método incluye una anotación **@Produces**(MediaType.APPLICATION\_JSON). Esto significa que el método regresará los resultados en formato JSON.

Para poder serializar y des-serializar JSON utilizaremos GSON (la implementación de JSON de Google), se declara una variable estática de tipo Gson y se inicializa con una instancia creada mediante el método create de la clase GsonBuilder:

```
static Gson j = new GsonBuilder()
    .registerTypeAdapter(byte[].class, new AdaptadorGsonBase64())
    .setDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS")
    .create();
```

Notar que en este caso se configura la instancia de Gson para utilizar el adaptador AdaptadorGsonBase64 para serializar y des-serializar arreglos de bytes, y se define el formato de fecha como ISO8601.

### El servicio web consume URL encoded

Cada método incluye la anotación **@Consumes**(MediaType.APPLICATION\_FORM\_URLENCODED) la cual indica que los parámetros del método se recibirán codificados como URL. Esta codificación reemplaza algunos caracteres especiales como ampersand &, igual =, espacio, etc. los cuales son caracteres reservados cuando se construye una URL.

### El nombre del método web

Cada método incluye la anotación **@Path**, esta anotación define el nombre del método web el cual puede ser diferente al nombre del método. Por ejemplo, el método "alta\_usuario" se invocará como el método web "alta".

### Los nombres de los parámetros del método web

Para indicar el nombre de cada parámetro del método web se utiliza la anotación **@FormParam**. El nombre del parámetro de método web puede ser diferente al nombre del parámetro del método Java.

### El método web regresa Response

El método web regresa una instancia de la clase Response. Básicamente hay tres tipos de respuesta:

**El método no regresa resultados.** El método termina sin error pero no regresa resultados al cliente (por ejemplo el método "alta\_usuario"). El método deberá regresar al cliente la siguiente instancia de la clase Response: Response.ok().build() esta instancia incluye el código HTTP 200 indicando que no hubo error.

**El método regresa resultados.** El método termina sin error y regresa algún resultado al cliente. Debido a que se definió en la anotación @Produces que el método regresa JSON, el método deberá regresar al cliente la siguiente instancia de la clase Response: Response.ok().entity(j.toJson(r)).build() en este caso r sería un objeto o un arreglo, el cual será serializado mediante GSON. El cliente recibe el código HTTP 200 indicando que no hubo error.

**El método regresa un mensaje de error.** El método termina con error. Para regresar un mensaje de error al cliente se utiliza la clase Error. Debido a que se definió en la anotación @Produces que el método regresa JSON, el método deberá regresar al cliente la siguiente instancia de la clase Response: Response.status(400).entity(j.toJson(new Error("Mensaje de error"))).build() el mensaje de error será de acuerdo al error que se produjo. El cliente recibe el código HTTP 400 indicando que hubo error. Adicionalmente, si así se desea se podría agregar a la clase Error un código numérico.

### El método web debe cerrar la conexión a la base de datos

Un error frecuente de los programadores es que no cierran las conexiones a la base de datos. En general, los DBMS cuentan con un número limitado de conexiones. Si una aplicación no cierra las conexiones que no utiliza, se agotarán las conexiones en el DBMS y el sistema dejará de funcionar.

Para evitar esta situación, es necesario que el programador cierre las conexiones a la base de datos, archivos, sockets y cualquier otro recurso que haya abierto el método web. Esto debe hacerse mediante un cuidadoso diseño del manejo de excepciones mediante try-finally-catch.

#### **El archivo WSClient.js**

Este archivo contiene código Javascript utilizado por prueba.html para consumir el servicio web mediante AJAX (XMLHttpRequest).

La función WSClient() incluye la función "post" la cual recibe los siguientes parámetros: el nombre del método web a invocar, un arreglo con los argumento del método web y una función callback que se invocará cuándo el método web termine.

#### **El archivo prueba.html**

Este archivo contiene una Single-Page Application (SPA) escrita en HTML-Javascript, la cual invoca los métodos web del servicio web "Servicio".

### Actividades individuales a realizar

1. Agregar un método web llamado "borrar\_usuarios" al archivo Servicio.java, este método deberá borrar todos los usuarios de la tabla "usuarios". El método deberá regresar un mensaje de error si no pudo borrar los usuarios.
2. Agregar un botón al archivo prueba.html. Al dar clic a este botón se deberá invocar el método web "borrar\_usuarios".
3. Compilar el archivo Servicio.java
4. Construir el archivo Servicio.war
5. Borrar el archivo Servicio.war y el subdirectorío Servicio los cuales se encuentran en el directorio webapps de Tomcat.
6. Copiar el archivo Servicio.war al directorio webapps de Tomcat.
7. Utilizando un navegador probar el nuevo botón que invoca el método web "borrar\_usuarios".

Clase del día - 04/05/2021



En la clase de hoy vamos a jugar un kahoot en la modalidad de "challenge".

Para jugar el kahoot deberán ingresar al siguiente enlace:

#### **Desarrollo de Sistemas Distribuidos - Servicios Web**

Es necesario que los alumnos y alumnas ingresen su "nickname" como su nombre y apellido (por ejemplo JuanLopez), de manera que sea posible identificar a los ganadores de puntos extra.

La hora límite para jugar este kahoot es 11:00 PM del 4 de mayo.



#### **Tarea 8. Desarrollo de un cliente para un servicio web REST**



Cada alumno deberá desarrollar un programa Java consola (modo carácter) cliente del servicio web que creamos en la tarea 7.

Se deberá realizar las siguientes modificaciones al servicio web que creamos en la tarea 7:

1. Agregar el campo id\_usuario a la clase Usuario.

2. Modificar el método web "alta\_usuario", de manera que al dar de alta un usuario el método web deberá regresar al cliente el id del usuario agregado. Se deberá desplegar el id del usuario dado de alta. El campo id\_usuario es auto\_increment en la base de datos, por tanto se deberá recuperar el ID inmediatamente después de ejecutar la instrucción INSERT.
3. Modificar el método web "consulta\_usuario", ahora la consulta se deberá realizar mediante el id del usuario no el email.
4. Modificar el método web "modifica\_usuario", utilizar el id del usuario en el WHERE de la instrucción UPDATE en lugar del email. No deberá ser posible modificar el id de un usuario ya que se trata de la llave primaria.
5. Modificar el método web "borra\_usuario", utilizando como clave el id del usuario no el email.

El programa cliente deberá desplegar el siguiente menú:

MENU

- a. Alta usuario
- b. Consulta usuario
- c. Borra usuario
- d. Salir

Opción: \_

Las opciones deberán implementar la siguiente funcionalidad:

- La opción "Alta usuario" leerá del teclado el email, el nombre del usuario, el apellido paterno, el apellido materno, la fecha de nacimiento, el teléfono y el género ("M" o "F"). Entonces se invocará el método web "alta\_usuario". Se deberá desplegar el id del usuario dado de alta, o bien, el mensaje de error que regresa el servicio web. Notar que el método web "alta\_usuario" recibe como parámetro una instancia de la clase Usuario, recordemos que esta clase se deberá definir de la siguiente manera:

```
class Usuario
{
    id_usuario int;
    String email;
    String nombre;
    String apellido_paterno;
    String apellido_materno;
    String fecha_nacimiento;
    String telefono;
    String genero;
    byte[] foto;
}
```

Para invocar el método web "alta\_usuario" desde el cliente Java es necesario crear una instancia de la clase Usuario y asignar los valores a los campos (en este caso el campo "foto" será null). Una vez que se tenga el objeto de tipo Usuario se deberá utilizar **GSON** para convertir el objeto a una string JSON, entonces se deberá utilizar esta string como valor del parámetro (ver más adelante cómo se enviarán los parámetros a los métodos web).

- La opción "Consulta usuario" leerá del teclado el id de un usuario previamente dado de alta. Entonces se invocará el método web "consulta\_usuario". Si el usuario existe, se desplegará en pantalla el nombre del usuario, el apellido paterno, el apellido materno, la fecha de nacimiento, el teléfono y el género. Debido a que el programa no es gráfico, la foto del usuario se ignorará. Notar que el método web "consulta\_usuario" regresa una string JSON la cual representa un objeto de tipo Usuario, por tanto será necesario utilizar **GSON** para convertir la string JSON a un objeto Java de tipo Usuario y posteriormente desplegar los campos del objeto (excepto el campo "foto"). Si hubo error, se desplegará el mensaje que regresa el servicio web.
- La opción "Borra usuario" leerá del teclado el id de un usuario previamente dado de alta. Entonces se invocará el método "borra\_usuario" del servicio web. Se deberá desplegar "El usuario ha sido borrado" si se pudo borrar el usuario, o bien, el mensaje de error que regresa el servicio web.
- La opción "Salir" terminará el programa.

**¿Cómo invocar un método web REST utilizando Java?**

A continuación se muestra un ejemplo de código Java que permite invocar el método web "consulta\_usuario" del servicio web **Servicio.war**

La URL que utilizaremos para acceder el método web "consulta\_usuario" es la siguiente (notar que el nombre del método se escribe al final de la URL):

```
http://ip-de-la-máquina-virtual:8080/Servicio/rest/ws/consulta_usuario
```

Para que el método web pueda recibir los parámetros, cada parámetro se codifica de la siguiente manera:

**nombre=valor**. Los parámetros se deben separar con un caracter **&**

El **valor** debe codificarse utilizando el método `URLEncoder.encode()`. La codificación URL reemplaza los caracteres reservados por una secuencia de escape consistente en un % y un valor hexadecimal, ver:

<https://developers.google.com/maps/documentation/urls/url-encoding>

Por ejemplo, si un método web tiene tres parámetros a, b y c, y los parámetros tienen los valores 10, 20 y 30, entonces se deberá enviar al método web los parámetros de la siguiente manera: "a=10&b=20&c=30", en este caso no es necesario utilizar el método `URLEncoder.encode()` para codificar el valor ya que se trata de un número.

El código que permite invocar el método "consulta\_usuario" del servicio web Servicio.war es el siguiente:

```
URL url = new URL("http://ip-de-la-máquina-virtual:8080/Servicio/rest/ws/consulta_usuario");
```

```
URLConnection conexion = (URLConnection) url.openConnection();
```

```
conexion.setDoOutput(true);
```

```
// en este caso utilizamos el método POST de HTTP
conexion.setRequestMethod("POST");
```

```
// indica que la petición estará codificada como URL
conexion.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
```

```
// el método web "consulta_usuario" recibe como parámetro el id de un usuario, en este caso el id es 10
String parametros = "id=" + URLEncoder.encode("10", "UTF-8");
```

```
OutputStream os = conexion.getOutputStream();
```

```
os.write(parametros.getBytes());
```

```
os.flush();
```

```
// se debe verificar si hubo error
if (conexion.getResponseCode() == 200)
{
    // no hubo error
    BufferedReader br = new BufferedReader(new InputStreamReader((conexion.getInputStream())));

    String respuesta;

    // el método web regresa una string en formato JSON
```

Usted está ingresado como [Jose Joel Perez Federico \(Salir\)](#)

[Página Principal \(home\)](#)

[Español - México \(es\\_mx\)](#)

[English \(en\)](#)

[Español - México \(es\\_mx\)](#)

[Resumen de conservación de datos](#)

[Obtener la App Mobile](#)

```
// capture una excepción para terminar el programa
throw new RuntimeException("Codigo de error HTTP: " + conexion.getResponseCode());
}

conexion.disconnect();
```

Notar que el código anterior se puede usar para desarrollar aplicaciones Android que requieran consumir servicios web REST.

Se deberá entregar un reporte en formato PDF que incluya las capturas de pantalla donde se muestre cada paso, desde la creación de la máquina virtual hasta la compilación y ejecución de cada opción del menú. El reporte deberá tener portada y conclusiones. Se deberá entregar también el todos los archivos utilizados incluyendo el código fuente del servicio web y el cliente.

El nombre de la máquina virtual deberá ser el número de boleta del alumno, si el número de boleta del alumno es 12345678, entonces la máquina virtual deberá llamarse: X12345678. **No se admitirá la tarea** si la máquina virtual no se nombra como se indicó anteriormente.

Las capturas de pantalla deberán ser completas, no se aceptará la tarea si incluye recortes de capturas de pantalla.

Valor de la tarea: 30% (1.8 puntos de la segunda evaluación parcial)

---

## 4. Servicios de nombres, archivos y replicación

---

## 5. Cómputo en la nube

---

© Carlos Pineda Guerrero, 2021. 