1. Transacción: define una secuencia de operaciones que se realiza por el servidor y se garantiza por el mismo que es atómica, ya sea en presencia de múltiples usuarios e incluso de caídas del servidor. Los servidores deben garantizar que se realiza completamente y que los resultados se almacenan en memoria permanente o no.
2. Interbloqueo fantasma: Un interbloqueo que se «detecta» pero que realmente no lo es se conoce como interbloqueo fantasma.
3. Propiedad de aislamiento de las transacciones: Cada transacción debe ser realizada sin interferencia de otras transacciones, es decir, los efectos intermedios de una transacción no deben ser visibles para los demás.
4. Registro histórico: registro que contiene el historial de todas las transacciones realizadas por el servidor.

**Propiedades de las transacciones:**

1. [Atomic](https://es.wikipedia.org/wiki/Atomicidad)idad
2. Consistencia
3. Aislamiento
4. Permanencia(Durabilidad)

**Operaciones dentro del protocolo de consumación de dos fases:**

1. puedeConsumar?(trans)
2. Consuma(trans)
3. Aborta(trans)

**Pasos en los algoritmos de caza de arcos:**

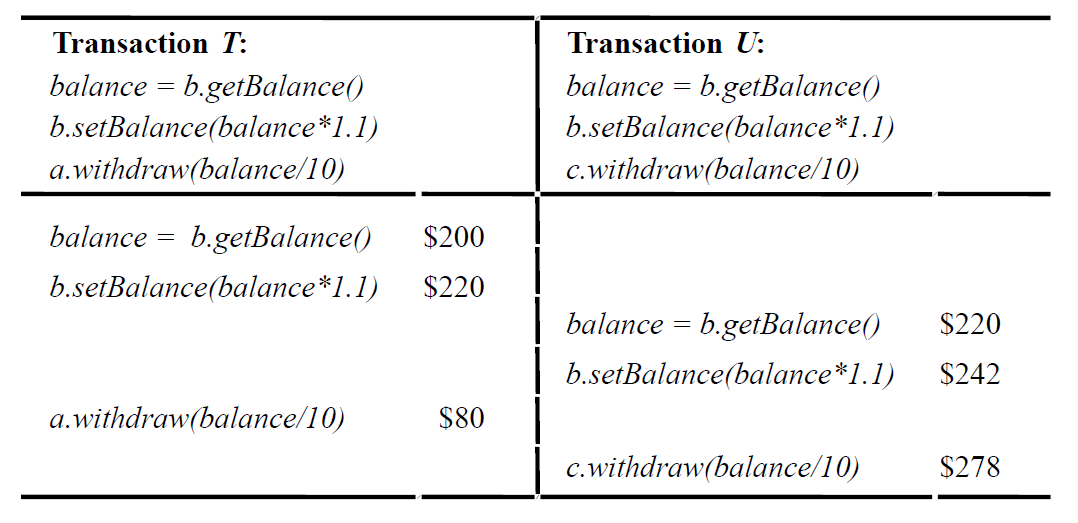
1. iniciación
2. detección
3. resolución

**Complete:**

1. Cuando decimos que un par de operaciones en transacciones tienen conflictos queremos decir que su efecto combinado depende del **orden en el que se ejecutan**
2. Cuando una sub transacción finaliza, hace una decisión independiente sobre si consumarse provisionalmente o abortar. Su decisión de **abortar** es final
3. Si una transacción T ha realizado ya una operación de **lectura** en un objeto particular, entonces una transacción concurrente U no debe **escribir** ese objeto hasta la consumación de T, o que aborte.
4. Un método para resolver bloqueos indefinidos es el uso de **los timeouts de bloqueo**

**Resuelva:**

1. **Escriba 2 transacciones cada una con operaciones sobre los objetos a, b y c, posteriormente re-escriba dichas transacciones para un solapamiento secuencial equivalentes. OBS: Graficar cada transacción en una columna, tal como fue analizada en clase.**



La figura muestra dicho solapamiento en el que las operaciones que afectan a la cuenta compartida B, son realmente secuenciales, puesto que la transacción T realiza todas las operaciones antes que las realice la transacción U. Otro solapamiento de T y U que tenga esta propiedad es uno en el que la transacción U finaliza sus operaciones en la cuenta B antes de que comience la transacción T.

1. **Explique el método de versiones provisionales**

Las operaciones de actualización durante una transacción se hacen sobre versiones provisionales de los objetos en memoria volátil, estas almacenan los valores de los objetos en el propio conjunto privado de la transacción si es posible o fallan

Las versiones provisionales son transferidas a los objetos sólo cuando la transacción se consuma entonces quedarán registradas en la memoria permanente, si una transacción aborta sus versiones provisionales se borran.

1. **Explique la fase 1 del protocolo de consumación en dos fases referente a transacciones distribuidas**

1- El coordinador envía una petición puedeConsumar? a cada participante en la transacción

2- Cuando un participante recibe una petición puedeConsumar?, responde al coordinador con su voto (Si o no). Antes de votar si, se prepara para consumar, guardando los objetos en un dispositivo de almacenamiento. Si el voto es No, el participante aborta inmediatamente.

1. **En la figura 1, explique las transacciones wait y notify dentro del contexto del método put**

Los métodos wait (espera) y notify (notifica) de Java permiten que los hilos se comuniquen con los otros de una manera que resuelva los problemas anteriores.

Un hilo llama a wait en un objeto para suspenderse él mismo y permitir a otro hilo ejecutar un método en ese objeto.

Un hilo llama a notify para informar que cualquier hilo que esté esperando en el objeto que ha cambiado algunos de sus datos.

El acceso a un objeto es todavía atómico cuando un hilo espera por otro: un hilo que llama a wait activa su bloqueo y se suspende como una acción atómica, cuando el hilo es activado después de ser notificado y adquiere un nuevo bloqueo en el objeto y recupera la ejecución del wait.

**wait es para esperar a que alguien realice una acción sobre ese objeto y notify es para avisar que se realizó alguna acción o modificación**

Dentro del método put, mientras la cola esté al límite de su capacidad, con **wait()** el hilo espera a que se realice alguna modificación sobre la cola (o sea, el mismo objeto instanciado **this**). Luego de salir del mientras, se agrega un objeto en la cola y con **notify()** se notifica al primer hilo en la cola de espera, que se realizó una modificación sobre la cola (o sea, el mismo objeto instanciado **this**) y finaliza la ejecución del método.

**Tema 1: Defina / Responda**

1. WSDL: (Web Services Description Language) es un lenguaje para describir servicios Web. Es un lenguaje de interfaz pública para los servicios Web de los requisitos funcionales necesarios para establecer una comunicación con los servicios Web.
2. Control optimista de concurrencia: se basa en la observación de que, en la mayoría de las aplicaciones, la similitud entre las transacciones de dos clientes que acceden al mismo objeto es baja.
3. Equivalencia secuencial: Para cualquier par de transacciones es posible determinar un orden de operaciones conflictivas sobre objetos accedidos por ambas. La equivalencia secuencial se utiliza como un criterio para la obtención de protocolos de control de concurrencia.
4. ¿Qué es y cómo se produce un interbloqueo fantasma? : Un interbloqueo que se «detecta» pero que realmente no lo es se conoce como interbloqueo fantasma. En la detección de interbloqueos distribuidos, la información acerca de las relaciones espera por entre las transacciones se transmite de un servidor a otro. Si existe un interbloqueo, la información necesaria se recogerá, al final, en algún lugar y se detectará el ciclo. Ya que este procedimiento tarda un cierto tiempo, existe la posibilidad de tanto, una de las transacciones que mantiene un bloqueo se haya liberado, en cuyo caso ya no existe.

**Tema 3:**

b) Escriba, en el lenguaje Java, un WebService que contenga 2 métodos web. Además escriba la representación XML (una aproximación) del ***request*** y ***response*** para la invocación de uno de sus métodos. (14p).

import javax.jws.WebService;

import javax.jws.WebMethod;

import javax.jws.WebParam;

@WebService(serviceName = "WsSuma")

public class WsSuma {

/\*\*

\* This is a sample web service operation

\*/

@WebMethod(operationName = "hello")

public String hello(@WebParam(name = "name") String txt) {

return "Hello " + txt + " !";

}

/\*\*

\* Web service operation

\*/

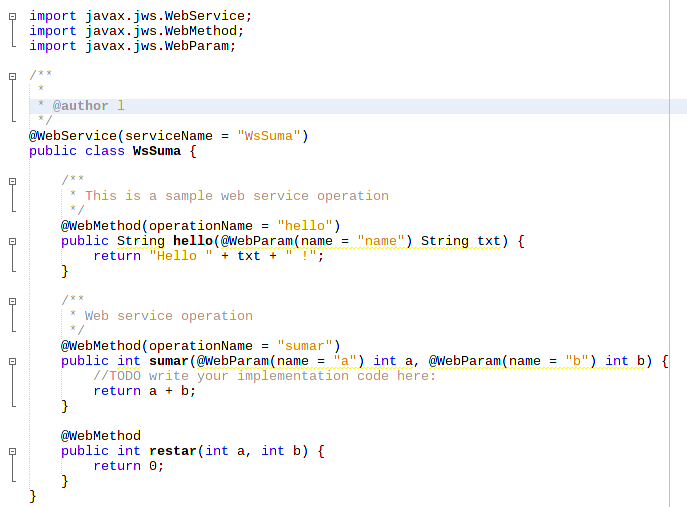
@WebMethod(operationName = "sumar")

public int sumar(@WebParam(name = "a") int a, @WebParam(name = "b") int b) {

return a + b;

}

}



***(El restar era una prueba mia, no lo mires xd, hay que poner como los de arriba)***

***No es necesario que operationName, y los name sean iguales a las variables y al metodo, pueden ser otra cosa, pero va a cambiar el request y response de acuerdo a esos parametros***

Para el método “hello”

Entrada: Melisa

Respuesta: Hello Melisa !

#### **SOAP Request**:

<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">  
 <SOAP-ENV:Header/>  
 <S:Body>  
 <ns2:hello xmlns:ns2="http://pol.py/">  
 <name>Melisa</name>  
 </ns2:hello>  
 </S:Body>  
</S:Envelope>

#### **SOAP Response:**

<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">  
 <SOAP-ENV:Header/>  
 <S:Body>  
 <ns2:helloResponse xmlns:ns2="http://pol.py/">  
 <return>Hello Melisa !</return>  
 </ns2:helloResponse>  
 </S:Body>  
</S:Envelope>

Para el metodo “sumar

Entradas: 1 y 2

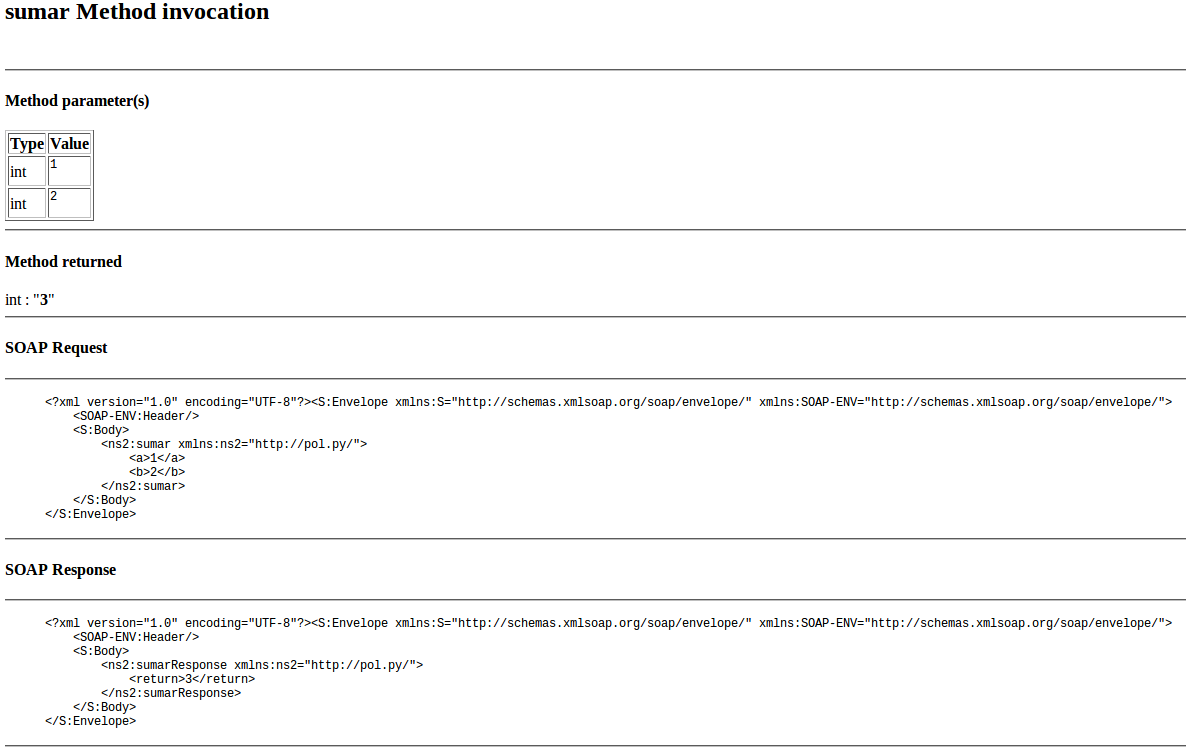
Respuesta: 4

#### **SOAP Request**

<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">  
 <SOAP-ENV:Header/>  
 <S:Body>  
 <ns2:sumar xmlns:ns2="http://pol.py/">  
 <a>1</a>  
 <b>2</b>  
 </ns2:sumar>  
 </S:Body>  
</S:Envelope>

#### **SOAP Response**

<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">  
 <SOAP-ENV:Header/>  
 <S:Body>  
 <ns2:sumarResponse xmlns:ns2="http://pol.py/">  
 <return>3</return>  
 </ns2:sumarResponse>  
 </S:Body>  
</S:Envelope>



e) Explique e indique el nombre de la primera fase del “protocolo de consumación en dos fases”:

Fase 1(fase de votación): 1- El coordinador envía una petición puedeConsumar? a cada participante en la transacción

2- Cuando un participante recibe una petición puedeConsumar?, responde al coordinador con su voto (Si o no). Antes de votar si, se prepara para consumar, guardando los objetos en un dispositivo de almacenamiento. Si el voto es No, el participante aborta inmediatamente.

**Tema 4: Complete con la o las palabras correctas**

1. En JavaRMI, el método **REBIND** del Registry(java.rmi.registry.Registry) es utilizado para publicar/disponibilizar un objeto remoto desde el servidor; y el método **LOOKUP** también del Registry (java.rmi.registry.Registry) es utilizado en el cliente para encontrar el objeto remoto.
2. Uno de los problemas que evita el control de concurrencia es **el problema de las actualizaciones perdidas.**
3. Una forma de detectar un bloqueo indefinido es **utilizado el grafo espera por.**
4. Al implementar mecanismos de bloqueo, si una transacción T ha realizado ya una operación de **LECTURA** en un objeto particular, entonces una transacción concurrente U no debe **ESCRIBIR** ese objeto hasta la consumación de T, o que aborte.