Complemento a actividad: calculadora distribuida

Parte 1:

1. Debe completar la actividad referida como **calculadora distribuida**, donde deberá ejecutar las pruebas de rigor realizando peticiones al servidor desde un cliente para verificar el funcionamiento correcto de cada operación.
2. En la clase **CalculatorI** debe agregar retardos en las operaciones para simular el tiempo de cómputo necesario para efectuar una operación compleja. Por ejemplo, el método add() lo he modificado de la siguiente forma:

```
@Override
public double add(double x, double y, Current current) {
    try{
        Thread.sleep(5000);
    }catch(Exception e){
    //TODO: handle exception
    }
    return x + y;
}
```

   Nota: poner retardos a todos los métodos

3. Una vez hecha la modificación deben ejecutar tres clientes al mismo tiempo y analizar el comportamiento de las respuestas. ¿Las respuestas se dan de forma simultanea o es necesario esperar que termine un llamado para iniciar el otro?.

Parte 2:

4. En el **servidor** realizar la siguiente modificación para permitir que la configuración se haga mediante un archivo de texto.

   Deben crear la carpeta **resources** en la ubicación:
   `mkdir server/src/main/resources/`
   En dicha carpeta deben crear el archivo **server.config** y poner el siguiente contenido:

```
services.Endpoints=tcp -h localhost -p 10000
```

   En la clase Server.java, al momento de crear el objeto Communicator deben enviar en los argumentos de Initialize() el nombre del archivo:

```
import MathCalc.*;
import com.zeroc.Ice.*;
public class Server {
    public static void main(String[] args) {
        try (Communicator communicator = Util.initialize(args,"server.config")) {
            ObjectAdapter adapter = communicator.createObjectAdapter("services");
```

   Aplicar de nuevo **./gradlew build** y **./gradlew :server:build**
   Verificar que todo funcione correctamente, ejecutando el servidor y el cliente.

5. Para usar ThreadPool es necesario indicarle a ICE cuantos hilos queremos iniciar en el pool, es decir, cuantos clientes pueden estar solicitando tareas de forma simultánea. Podemos agregar la siguiente línea al archivo de configuración server.config:
   `Ice.ThreadPool.Server.Size=5`
6. Aplicar de nuevo **./gradlew build** y **./gradlew :server:build**
   Una vez terminadas las modificaciones, aplicar de nuevo la prueba efectuada en el punto 3. ¿Que diferencias puede notar?

7. **Del lado del cliente** puede realizar modificaciones similares para que la configuración se haga mediante un archivo. Le llamaremos **client.config**
   Debe estar guardado en la carpeta **client/src/main/resources/** y debe tener el siguiente contenido:

   ```
   math.proxy= CalcServant:tcp -h localhost -p 10000
   ```

   Debe tener en cuenta que en el momento de agregar el objeto servant (instancia de CalculatorI) al adapter le dimos un nombre o una identidad, en este caso se le llama CalcServant y debe coincidir con el nombre puesto en el archivo .config.
8. La clase Client.java debe modificarla de la siguiente forma para indicar que debe cargar el archivo y asignar la configuración al proxy:

```java
import MathCalc.*;
import com.zeroc.Ice.*;
import java.util.concurrent.CompletableFuture;


public class Client {
    public static void main(String[] args) throws java.lang.Exception{
        try (Communicator communicator = Util.initialize(args, "client.config")) {
            ObjectPrx base = communicator.propertyToProxy("server.proxy");
```

9. El llamado al **servant** puede realizarla de la siguiente forma:

```java
double x = 10;
double y = 5;
double r;

// Realizar algunas operaciones de cálculo
System.out.println("Llamado a suma... ");
r = calculator.add(x, y)
System.out.println("Resultado: " + r );
```

Parte 3:

10. Tomando como referencia las conclusiones de su análisis al realizar las pruebas de los puntos 3 y 6. Realizar la siguiente consulta:
    Cual es la diferencia entre llamados remotos síncronos y llamados remotos asíncronos?
    En que situaciones se pueden usar cada uno de estos llamados?

11. En la clase Client.java Realizar la siguiente modificación del llamado al método add(). En su lugar usar addAsync()

```
// Realizar algunas operaciones de cálculo
System.out.println("Llamado a suma asincrona... ");
CompletableFuture<Double> response = calculator.addAsync(x, y);

r =response.get();
System.out.println("Resultado : " + r);
```

En el encabezado debe importar el paquete :

```
import java.util.concurrent.CompletableFuture;
```

12. **./gradlew build** y **./gradlew :client:build**
13. Ejecutar de nuevo las pruebas con 2 clientes realizando llamados de forma simultánea. Que diferencias observa con respecto a las pruebas 3 y 6?.