

CICLE FORMATIU: CFGS Desenvolupament Aplicacions Multiplataforma DAM

MÒDUL PROFESSIONAL: MP09 Programació de serveis i processos

UNITAT FORMATIVA: UF3. Sòcols i serveis

ACTIVITAT: Invocació de mètodes remots

Pràctica 3. Invocació de mètodes remots.

En aquesta pràctica crearem una aplicació client/servidor que farà servir mètodes remots del servidor, mitjançant RMI en java.

- Busca al moodle l'enllaç al codi del exemple de la calculadora remota.
- L'objectiu és implementar els mètodes sumar, restar, multiplicar i dividir en el costat del servidor, i poder accedir a ells directament des del programa client.
- Et pot ajudar també el codi de l'enllaç "calculadora simple RMI" del moodle.
- Quan ja et funcionin els quatre mètodes anteriors, afegeix un nou mètode "potencia" que calculi un número elevat a un altre.

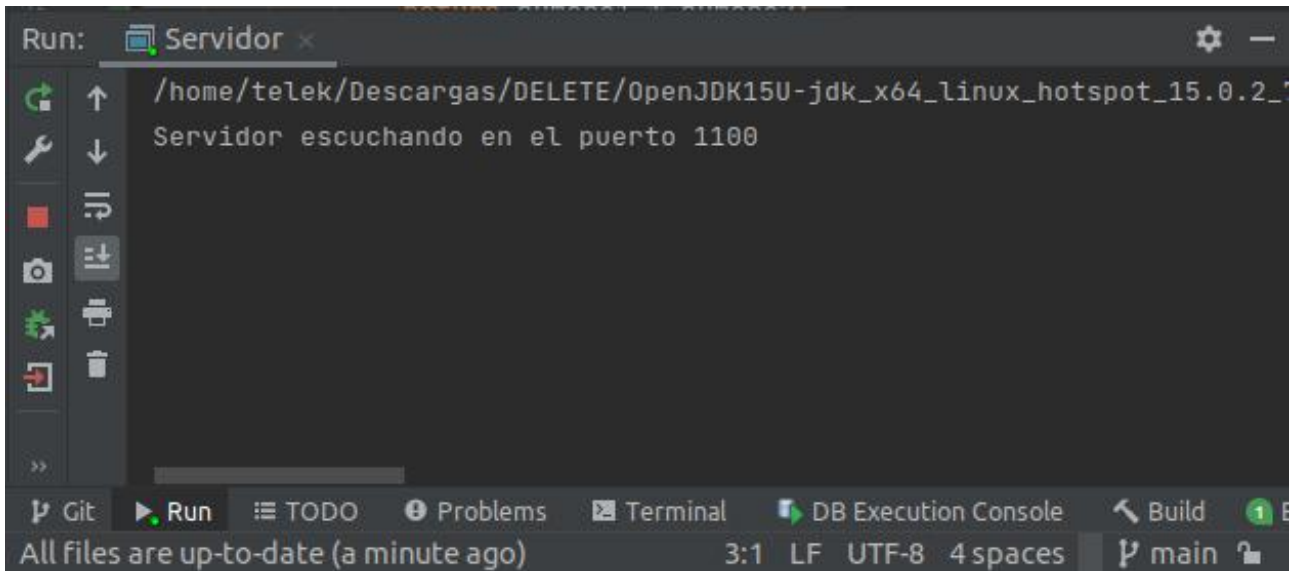
Pràctica

Entrega un document amb els programes client i servidor, i una breu explicació del que fa cada un d'ells. Codi font comentat. Fes captures d'execucions.

El programa client primer de tot prepara la connexió al servidor, i a executa un petit programa per consola per preguntar al usuari per la operació a realitzar en el servidor, i un cop el usuari ha escollit, i introduït els valors a calcular, envia la sol·licitut al servidor i mostra per pantalla el resultat retornat per el servidor.

Per el contrari, el programa servidor té emmagatzemades i preparades les

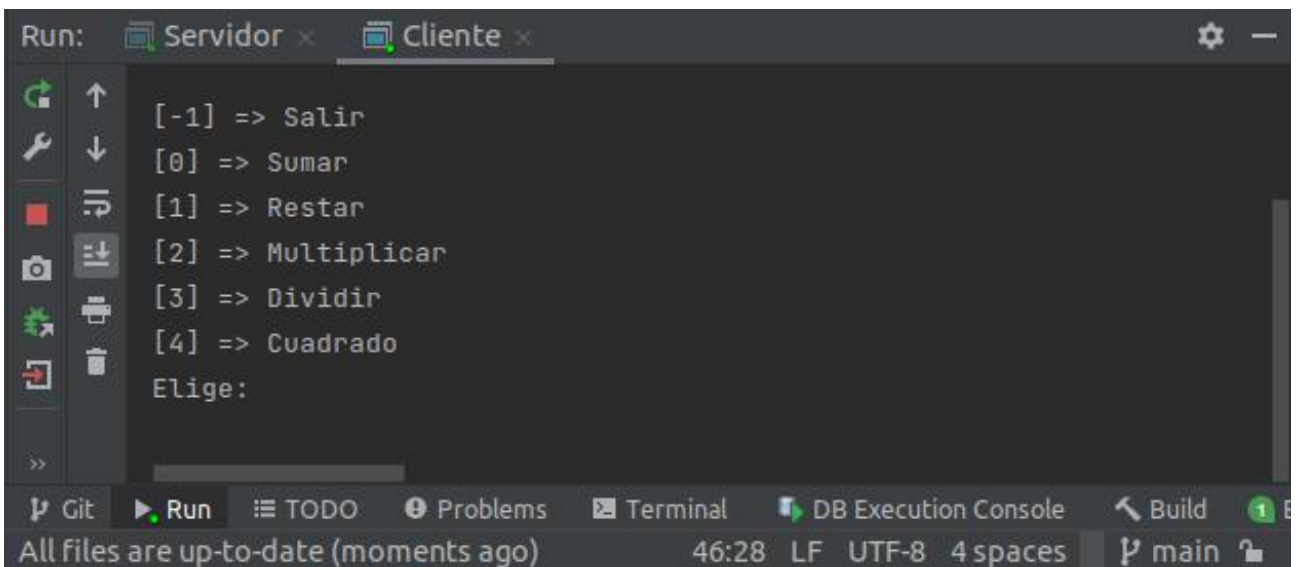
operacions a realitzar, i en el moment que rep una sol·licitut, depenent d'aquesta, s'executarà una funció o una altra, i retornarà el resultat de l'operació indicada amb els números proporcionats.



The screenshot shows an IDE terminal window with the title 'Run: Servidor x'. The terminal output displays the file path `/home/telek/Descargas/DELETE/OpenJDK15U-jdk_x64_linux_hotspot_15.0.2_...` and the message 'Servidor escuchando en el puerto 1100'. The IDE interface includes a sidebar with icons for Run, Debug, Test, and other tools. The bottom status bar indicates 'All files are up-to-date (a minute ago)', '3:1 LF UTF-8 4 spaces', and 'main'.

```
Run: Servidor x
/home/telek/Descargas/DELETE/OpenJDK15U-jdk_x64_linux_hotspot_15.0.2_...
Servidor escuchando en el puerto 1100
```

Programa servidor un cop inicialitzat.



The screenshot shows an IDE terminal window with the title 'Run: Servidor x' and 'Cliente x'. The terminal output displays a menu of operations: `[-1] => Salir`, `[0] => Sumar`, `[1] => Restar`, `[2] => Multiplicar`, `[3] => Dividir`, `[4] => Cuadrado`, and 'Elige:'. The IDE interface includes a sidebar with icons for Run, Debug, Test, and other tools. The bottom status bar indicates 'All files are up-to-date (moments ago)', '46:28 LF UTF-8 4 spaces', and 'main'.

```
Run: Servidor x
Cliente x
[-1] => Salir
[0] => Sumar
[1] => Restar
[2] => Multiplicar
[3] => Dividir
[4] => Cuadrado
Elige:
```

Programa client demanant la operació a realitzar al usuari.

```
[-1] => Salir
[0] => Sumar
[1] => Restar
[2] => Multiplicar
[3] => Dividir
[4] => Cuadrado
Elige:
2
Ingresa el número 1:
5
Ingresa el número 2:
6
Resultado => 30.0
Presiona ENTER para continuar
```

Programa client demanant les dades a l'usuari i mostrant la resposta del servidor per pantalla.

Explicació del codi:

Client:

```
1  import ...
2  public class Cliente {
3      private static final String IP = "localhost"; // Puedes cambiar a localhost
4      private static final int PUERTO = 1100; //Si cambias aqui el puerto, recuerda cambiarlo en el servidor
5
6      public static void main(String[] args) throws RemoteException, NotBoundException {
7          Registry registry = LocateRegistry.getRegistry(IP, PUERTO);
8          Interfaz interfaz = (Interfaz) registry.lookup( name: "Calculadora"); //Buscar en el registro...
9          Scanner sc = new Scanner(System.in);
10         int eleccion;
11         float numero1, numero2 = 0, resultado = 0;
12         String menu = "\n\n-----\n\n[-1] => Salir\n[0] => Sumar\n[1] => Restar\n[2] => Multiplicar\n[3] => Dividir\n[4] => Cuadrado\nElige:
13         do {
14             System.out.println(menu);
15
16             try {
17                 eleccion = Integer.parseInt(sc.nextLine());
18             } catch (NumberFormatException e) {
19                 eleccion = -1;
20             }
21         }
22     }
23 }
```

Primer de tot, iniciem la classe Cliente, declarem la ip a connectar-nos ("localhost" en aquest cas). Després, declarem també el port (1100), i un cop tenim aquestes variables declarades, comencem a programar la funció main.

La funció main, primer de tot crea un nou objecte registry amb la ip i el port declarats previament. En aquest moment s'estableix la connexió, pero encara no s'executa res.

Després, creem un objecte interfaz, mitjançant una interface amb les classes que s'executaràn en remot.

Un cop tenim això, podem programar de la mateixa manera que ho fariem normalment, fent crides a les funcions en remot. (en el nostre cas, es tracta d'un bucle on sempre demanem al usuari una operació i diversos valors per operar amb aquests. Només es surt d'aquest bucle en seleccionar l'opció -1 en el menú (sortir).

```
25
26     if(eleccion != -1){
27
28         System.out.println("Ingresa el número 1: ");
29         try{
30             numero1 = Float.parseFloat(sc.nextLine());
31         }catch(NumberFormatException e){
32             numero1 = 0;
33         }
34         if(eleccion != 4) {
35             System.out.println("Ingresa el número 2: ");
36             try{
37                 numero2 = Float.parseFloat(sc.nextLine());
38             }catch(NumberFormatException e){
39                 numero2 = 0;
40             }
41         }
42
43         switch (eleccion) {
44             case 0 -> resultado = interfaz.sumar(numero1, numero2);
45             case 1 -> resultado = interfaz.restar(numero1, numero2);
46             case 2 -> resultado = interfaz.multiplicar(numero1, numero2);
47             case 3 -> resultado = interfaz.dividir(numero1, numero2);
48             case 4 -> resultado = interfaz.potencia(numero1);
49         }
50     }
```

Final del codi de client. Com es pot visualitzar, les crides a les funcions en remot es realitzen de la mateixa manera que es realitzaria amb funcions en local.

Servidor:

```
1 import ...
2
3 public class Servidor {
4     private static final int PUERTO = 1100; //Si cambias aqui el puerto, recuerda cambiarlo en el cliente
5     public static void main(String[] args) throws RemoteException, AlreadyBoundException {
6         Remote remote = UnicastRemoteObject.exportObject(new Interfaz() {
7             /*
8              * Sobrescribir opcionalmente los métodos que escribimos en la interfaz
9              */
10            @Override
11            public float sumar(float numero1, float numero2) throws RemoteException {
12                return numero1 + numero2;
13            }
14
15            @Override
16            public float restar(float numero1, float numero2) throws RemoteException {
17                return numero1 - numero2;
18            }
19
20            @Override
21            public float multiplicar(float numero1, float numero2) throws RemoteException {
22                return numero1 * numero2;
23            }
24
25            @Override
26            public float dividir(float numero1, float numero2) throws RemoteException {
27                return numero1 / numero2;
28            }
29
30            @Override
31            public float potencia(float numero1) throws RemoteException {
32                return numero1*numero1;
33            }
34        }, port: 0);
35        Registry registry = LocateRegistry.createRegistry(PUERTO);
36        System.out.println("Servidor escuchando en el puerto " + PUERTO);
37        registry.bind(name: "Calculadora", remote); // Registrar calculadora
38    }
39 }
```

El codi de servidor, es relativament senzill. Al igual que amb el client, primer de tot declara el port on s'obrirà el servidor, i en aquest cas no cal definir la IP, ja que aquesta es la direcció de la propia màquina. Un cop fet això, al main es declara un objecte remote que incorpora un override a cadascuna de les funcions de la interface.

Després d'això, es crea un objecte registry similar al del client però només declarant el port (en aquest moment el servidor ja es troba "obert"), fem un print per poder controlar per terminal que l'execució es exitosa, i finalment fem un registry.bind per fer públic el nostre Interface (a RMI es important que la interface comuna entre servidor i client sigui pública, ja que el client comprova que la seva interface i la del servidor tinguin les mateixes funcions).

Interface:

```
1  import java.rmi.Remote;
2  import java.rmi.RemoteException;
3
4  /**
5   * Declarar firma de métodos que serán sobrescritos
6   */
7  public interface Interfaz extends Remote {
8      float sumar(float numero1, float numero2) throws RemoteException;
9      float restar(float numero1, float numero2) throws RemoteException;
10     float multiplicar(float numero1, float numero2) throws RemoteException;
11     float dividir(float numero1, float numero2) throws RemoteException;
12     float potencia(float numero1) throws RemoteException;
13
14 }
```

Finalment, la interface ha d'extendre Remote, i consisteix en una serie de funcions, les quals poden llençar una excepció del tipus RemoteException (sempre pot fallar en qualsevol moment la connexió). Com que es tracta d'una interface, no definim les funcions, ja que aixó es fa a la classe de servidor.