

# Universidad Nacional de San Antonio Abad del Cusco

## Departamento Académico de Informática

### ALGORITMOS PARALELOS Y DISTRIBUIDOS

## Práctica N° 9

### RMI

### (Remote Method Invocation)

#### 1. OBJETIVOS.

- Comprender la noción de programación cliente / servidor distribuida.
- Comprender la arquitectura de RMI.
- Utilizar RMI para implementar aplicaciones cliente / servidor

#### 2. INTRODUCCION

RMI (*Remote Method Invocation*) permite a objetos Java llamar a métodos de otros objetos que están ejecutándose en otras máquinas como si fueran llamadas a objetos definidos localmente por la aplicación. Esta funcionalidad se aporta a través de las clases `java.rmi.*` y `java.rmi.server.*`. El proceso por el que un objeto puede invocar métodos en un objeto remoto se divide en dos partes: la obtención de una referencia al objeto remoto y la invocación propiamente dicha.

#### 3. DESARROLLO DE LA PRACTICA

El siguiente ejercicio establece un objeto remoto que realiza las operaciones aritméticas en el servidor, los clientes invocan remotamente a los métodos `sumar()`, `restar()`, `multiplicar()`, `dividir()` de este objeto pasando como parámetros los operandos, el servidor responde enviando el resultado obtenido. Los pasos a seguir son los siguientes:

##### 3.1 Crear las siguientes carpetas:

D:\rmi  
D:\rmi\cliente  
D:\rmi\servidor

##### 3.2 Definir la interfaz remota

Para indicar que un objeto es remoto, el objeto define una interfaz el cual tiene las siguientes características:

1. La interfaz debe ser pública.
2. La interfaz debe extender de `java.rmi.Remote`.
3. Cada método en la interfase debe declarar que lanza una excepción `java.rmi.RemoteException`.

```
import java.rmi.Remote;
import java.rmi.RemoteException;

/*
Declarar firma de métodos que serán sobrescritos
*/
public interface Interfaz extends Remote
{
```

```

float sumar(float numero1, float numero2) throws RemoteException;
float restar(float numero1, float numero2) throws RemoteException;
float multiplicar(float numero1, float numero2) throws RemoteException;
float dividir(float numero1, float numero2) throws RemoteException;
}

```

Nombre este archivo como *Interfaz.java* y guárdelo en la carpeta del cliente y del servidor

### 3.3 Programación del Servidor

Ahora veamos el servidor. Este se encarga de exportar el objeto, sobrescribir funciones de la interfaz y escuchar peticiones del cliente.

```

import java.rmi.AlreadyBoundException;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
public class Servidor
{
    private static final int PUERTO = 1100; //Si cambias aquí el puerto,
                                           //recuerda cambiarlo en el cliente
    public static void main(String[] args) throws RemoteException, AlreadyBoundException
    {
        Remote remote = UnicastRemoteObject.exportObject(new Interfaz())
        {
            /*
                Sobrescribir opcionalmente los métodos que escribimos en la interfaz
            */
            @Override
            public float sumar(float numero1, float numero2) throws RemoteException {
                return numero1 + numero2;
            };

            @Override
            public float restar(float numero1, float numero2) throws RemoteException {
                return numero1 - numero2;
            };

            @Override
            public float multiplicar(float numero1, float numero2) throws RemoteException {
                return numero1 * numero2;
            };

            @Override
            public float dividir(float numero1, float numero2) throws RemoteException {
                return numero1 / numero2;
            };
        }, 0);

        Registry registry = LocateRegistry.createRegistry(PUERTO);
        System.out.println("Servidor escuchando en el puerto " + String.valueOf(PUERTO));
        registry.bind("Calculadora", remote); // Registrar calculadora
    }
}

```

Nombre este archivo como *Servidor.java* y guárdelo en la carpeta del servidor.

### 3.4 Programación del Cliente.

Para invocar a un método remoto, el programa cliente debe disponer de la siguiente información:

1. El nombre o la dirección IP de la máquina en la que se ha registrado al objeto remoto.

2. El puerto por el cual el objeto remoto está escuchando, por defecto es el puerto 1099
3. El nombre local del objeto remoto dentro del Object Registry.

```
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;
public class Cliente
{
    private static final String IP = "localhost"; //puede ser otro ejem. 192.168.1.15
    private static final int PUERTO = 1100; //Si cambias aquí el puerto,
                                           //recuerda cambiarlo en el servidor

    public static void main(String[] args) throws RemoteException, NotBoundException
    {
        Registry registry = LocateRegistry.getRegistry(IP, PUERTO);
        Interfaz interfaz = (Interfaz) registry.lookup("Calculadora"); //Buscar en el
                                                                    //registro...

        Scanner sc = new Scanner(System.in);
        int eleccion;
        float numero1, numero2, resultado = 0;
        String menu = "\n\n-----\n\n[-1] => Salir\n[0] => Sumar\n[1] =>
                        Restar\n[2] => Multiplicar\n[3] => Dividir\nElige: ";
        do {
            System.out.println(menu);

            try
            {
                eleccion = Integer.parseInt(sc.nextLine());
            } catch (NumberFormatException e) {
                eleccion = -1;
            }

            if(eleccion != -1){

                System.out.println("Ingresa el número 1: ");
                try{
                    numero1 = Float.parseFloat(sc.nextLine());
                }catch(NumberFormatException e){
                    numero1 = 0;
                }

                System.out.println("Ingresa el número 2: ");
                try{
                    numero2 = Float.parseFloat(sc.nextLine());
                }catch(NumberFormatException e){
                    numero2 = 0;
                }

                switch (eleccion) {
                    case 0:
                        resultado = interfaz.sumar(numero1, numero2);
                        break;
                    case 1:
                        resultado = interfaz.restar(numero1, numero2);
                        break;
                    case 2:
                        resultado = interfaz.multiplicar(numero1, numero2);
                        break;
                    case 3:
                        resultado = interfaz.dividir(numero1, numero2);
                        break;
                }
            }
        } while (eleccion != -1);
    }
}
```

```

    }
    System.out.println("Resultado => " + String.valueOf(resultado));
    System.out.println("Presiona ENTER para continuar");
    sc.nextLine();
    }
    } while (eleccion != -1);
}

```

Nombre este archivo como *Cliente.java* y guárdelo en la carpeta del cliente.

### 3.5 Compilar el Servidor.

Para compilar el servidor abrimos una ventana de comandos e ingresamos hasta la carpeta D:\rmi\servidor y ejecutamos la siguiente instrucción:

```
javac Servidor.java Interfaz.java
```

### 3.6 Compilar el Cliente.

Para compilar el cliente abrimos una ventana de comandos e ingresamos a la carpeta D:\rmi\cliente y ejecutamos la siguiente instrucción:

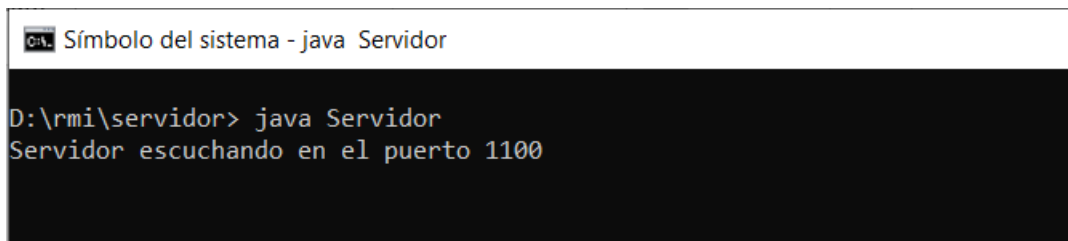
```
Javac Cliente.java Interfaz.java
```

### 3.7 Arrancar el Servidor.

En la ventana de comandos del servidor ejecutamos la siguiente instrucción:

```
java Servidor
```

y obtendremos el siguiente resultado.



```

C:\A. Símbolo del sistema - java Servidor

D:\rmi\servidor> java Servidor
Servidor escuchando en el puerto 1100

```

### 3.8 Ejecutar los clientes .

En la ventana del cliente simplemente digitamos la siguiente instrucción:

```
java Cliente
```

y obtendremos un menú para realizar las operaciones aritméticas como se muestra en la siguiente figura:

CS\ Símbolo del sistema - java Cliente

```
D:\rmi\cliente>java Cliente
```

```
-----  
[-1] => Salir  
[0] => Sumar  
[1] => Restar  
[2] => Multiplicar  
[3] => Dividir  
Elige:
```

#### 4. TRABAJO

- Realizar un programa basado en RMI para simular las operaciones en un banco. Los clientes tienen registrado una cuenta en una estructura de datos a la que pueden acceder a través de un código para depositar dinero o para retirar dinero o para realizar consultas de su saldo.

#### 5. BIBLIOGRAFIA

- Deitel y Deitel “JAVA How to Program” 3ra. Edición Prentice Hall 1999.
- Ken Arnold “El Lenguaje de Programación JAVA”. 3ra. Edición. James Goslin Addison Wesley 2001  
David Holmes
- <http://java.sun.com/> Sun Microsystems.