Linkia FP

Formación Profesional Oficial a Distancia

Actividad

SERVIDOR BANCARIO RMI Y CHAT

M09 SOCKET Y SERVICIOS

Santiago Fernández González

INDICE

CHAT

Cliente

Creamos un chat que nos permita mantener una comunicación entre un cliente y un servidor de forma asíncrona. Para ello utilizaremos la comunicación por socket y multihilo, que nos permitirán enviar varios mensajes seguidos en ambas direcciones.

Las dos aplicaciones, cliente y servidor, se ejecutaran en la misma maquina por lo que utilizaremos localhost. Se ha definido el puerto de escucha 5050.

```
public static void main(String[] args) {
    String ipServidor = "localhost";
    String recibida="";
    Scanner sc = new Scanner(System.in);
    String mensaje="";

    try {
        Socket conexion = new Socket(ipServidor,5050);
        BufferedReader entrada = new BufferedReader(new InputStreamReader(conexion.getInputStream()));
        PrintStream salida = new PrintStream(conexion.getOutputStream());
        while(!mensaje.equals("--SALIR--")) {
```

He creado una clase llamada hilo qu eextiende la clase Thread, en la que estarán los mensajes recibidos desde el servidor y serán ejecutados en multihilo.

```
class Hilo extends Thread{
    String recibida;
    BufferedReader entrada;

    Hilo(BufferedReader entrada){

        this.entrada=entrada;
    }
    public void run() {
        try {
            recibida = entrada.readLine();
            System.out.println("El servidor dice:" + recibida);
        }catch(IOException e) {

        }
    }
}
```

En la clase cliente estará el main.

```
public class Cliente {

    public static void main(String[] args) {
        String ipServidor = "localhost";
        String recibida="";
        Scanner sc = new Scanner(System.in);
        String mensaje="";

        try {
            Socket conexion = new Socket(ipServidor,5050);
            BufferedReader entrada = new BufferedReader(new InputStreamReader(conexion.getInputStream()));
            PrintStream salida = new PrintStream(conexion.getOutputStream());
            while(!mensaje.equals("--SALIR--")) {
                Hilo hilo = new Hilo(entrada);
                hilo.start();
                System.out.print("Mensaje a enviar: ");
                mensaje = sc.nextLine();
                salida.println(mensaje);


            }
            entrada.close();
            salida.close();
            conexion.close();
        }catch(IOException e) {

        }
    }
}
```

Servidor

En el caso del servidor hemos definido como puerto de escucha para el cliente el 5050.

```
try {
    ServerSocket conexion = new ServerSocket(5050);
    Socket conectado = conexion.accept();
    BufferedReader entrada = new BufferedReader(new InputStreamReader(conectado.getInputStream()));
    PrintStream salida = new PrintStream(conectado.getOutputStream());
```

También he extendido la clase Thread para utilizar mulhilo en la recepción de los mensajes desde el cliente.

```
    class HiloServidor extends Thread{
        String recibida;
        BufferedReader entrada;

        HiloServidor(BufferedReader entrada){

            this.entrada=entrada;
        }
        public void run() {
            try {
                recibida = entrada.readLine();
                System.out.println("El cliente dice: " + recibida);
            }catch(IOException e) {

            }
        }
    }
```

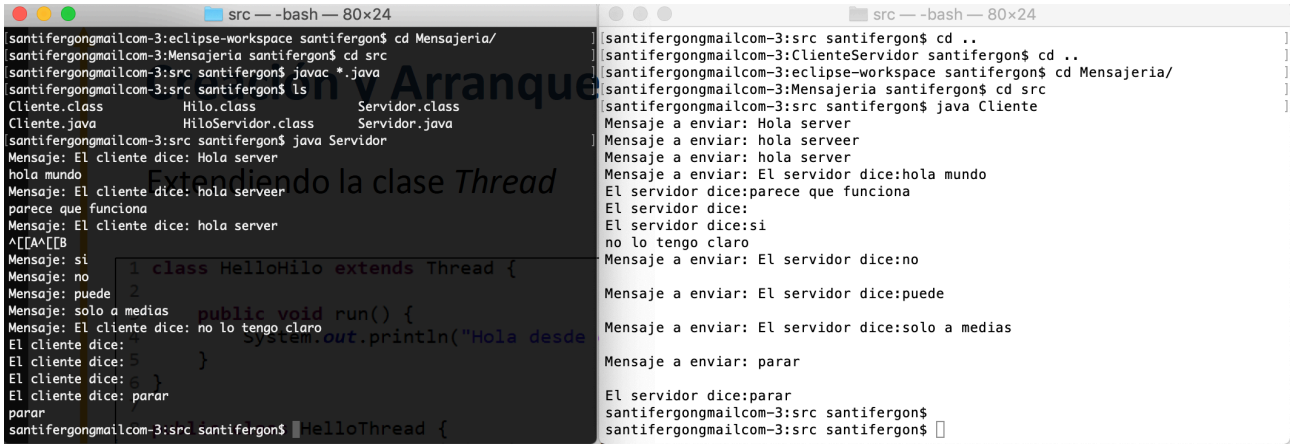
En la clase servidor tenemos el resto del código.

```
public class Servidor {

    public static void main(String[] args) {
        String recibida="";
        Scanner sc = new Scanner(System.in);
        String mensaje="";

        try {
            ServerSocket conexion = new ServerSocket(5050);
            Socket conectado = conexion.accept();
            BufferedReader entrada = new BufferedReader(new InputStreamReader(conectado.getInputStream()));
            PrintStream salida = new PrintStream(conectado.getOutputStream());
            while(!mensaje.equals("--SALIR--")) {
                HiloServidor hilo = new HiloServidor(entrada);
                hilo.start();
                System.out.print("Mensaje: ");
                mensaje = sc.nextLine();
                salida.println(mensaje);
            }
            salida.close();
            entrada.close();
            conectado.close();
        }catch(IOException e) {
            System.out.println("ERROR: " + e.getMessage());
        }
    }
}
```

El resultado de la ejecución quedaría como se muestra en la imagen. La ventana de la izquierda ejecutara el código de servidor y la de la derecha el código de cliente.



SERVIDOR BANCARIO

Definicion

Creare un servidor bancario que permita convertir euros a diferentes monedas y calculo de interés simple y compuesto. Se ha creado en 4 archivos diferentes, el primero llamado Banco contiene la definición de los métodos.

```java
import java.rmi.Remote;

public interface Banco extends Remote {

    public double convertirDolares(double cantidad) throws RemoteException;
    public double convertirYenes(double cantidad) throws RemoteException;
    public double convertirPesos(double cantidad) throws RemoteException;
    public double convertirLibras(double cantidad) throws RemoteException;
    public double interesSimple(double cantidad, double tasa, int tiempo) throws RemoteException;
    public double interesCompuesto(double cantidad, double tasa, int tiempo) throws RemoteException;

}
```

Implementación

En el segundo llamado BancoImpl se han implementado los métodos definidos en el anterior archivo.

```java
import java.rmi.*;

public class BancoImpl extends UnicastRemoteObject implements Banco {

        public BancoImpl() throws RemoteException {
            super();
        }

        public double convertirDolares(double cantidad) throws RemoteException {
            cantidad  = cantidad * 1.11;
            return cantidad;
        }

        public double convertirYenes(double cantidad) throws RemoteException {
            cantidad = cantidad * 124.75;
            return cantidad;
        }

        public double convertirPesos(double cantidad) throws RemoteException {
            cantidad = cantidad * 21.15;
            return cantidad;
        }

        public double convertirLibras(double cantidad) throws RemoteException {
            cantidad = cantidad * 0.86;
            return cantidad;
        }

        public double interesSimple(double cantidad, double tasa, int tiempo) throws RemoteException {
            cantidad = cantidad*(tasa/100)*tiempo;
            return cantidad;
        }

        public double interesCompuesto(double cantidad, double tasa, int tiempo) throws RemoteException {
            cantidad = cantidad*(1+(tasa/100))*tiempo;
            return cantidad;
        }

}
```

Servidor Bancario

En el archivo BancoServer se ha creado el servidor que estará escuchando en el puerto 5555.

```
import java.rmi.registry.Registry;

public class BancoServer {

    public static void main(String[] args) throws RemoteException {
        try {
            Registry reg = LocateRegistry.createRegistry(5555);
            Banco banco = new BancoImpl();
            reg.rebind("Banco", banco);
            System.out.println("Servidor en marcha.....");
        }catch(Exception e) {
            System.err.println("Error: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Cliente

El archivo ClienteBanco contiene el cliente que conectara con el servidor y llamara a los métodos creados anteriormente.

```
Registry reg = LocateRegistry.getRegistry("localhost", 5555);
Banco b = (Banco)reg.lookup("Banco");
System.out.println("Cliente conectado...");
System.out.println("Pulse una tecla...");
```

Para la llamada a los diferentes métodos se ha creado un menú.

```
System.out.println("Opciones");
System.out.println("--------");
System.out.println("D -> Convertir a dolares");
System.out.println("Y -> Convertir a yenes");
System.out.println("P -> convertir a pesos");
System.out.println("L -> Convertir a libras");
System.out.println("I -> Interes Simple");
System.out.println("C -> Interes Compuesto");
System.out.print("Opcion: ");
```

Mediante switch se gestionan las llamadas a los métodos definidos.

```
switch(opcion) {
    case "D":
        resultado = b.convertirDolares(cantidad);
        System.out.println("La cantidad de " + cantidad + " € son " + resultado + " $");
        break;
    case "Y":
        resultado = b.convertirYenes(cantidad);
        System.out.println("La cantidad de " + cantidad + " € son " + resultado + " ¥");
        break;
    case "P":
        resultado = b.convertirPesos(cantidad);
        System.out.println("La cantidad de " + cantidad + " € son " + resultado + " $ (Pesos)");
        break;
    case "L":
        resultado = b.convertirLibras(cantidad);
        System.out.println("La cantidad de " + cantidad + " € son " + resultado + " £");
        break;
    case "I":
        System.out.print("Tasa: ");
        tasa = sc.nextDouble();
        System.out.print("Tiempo: ");
        tiempo=sc.nextInt();
        resultado = b.interesSimple(cantidad,tasa,tiempo);
        System.out.println("El interes simple es: " + resultado);
        break;
    case "C":
        System.out.print("Tasa: ");
        tasa = sc.nextDouble();
        System.out.print("Tiempo: ");
        tiempo=sc.nextInt();
        resultado = b.interesCompuesto(cantidad,tasa,tiempo);
        System.out.println("El interes simple es: " + resultado);
        break;
    default:
        System.out.println("No es una opcion valida");
}
```