Tarea Recuperación Tema 3. Museo Cliente-Servidor

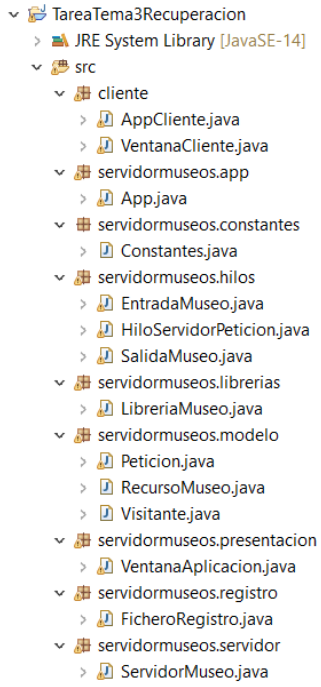
Contenido

Entrega

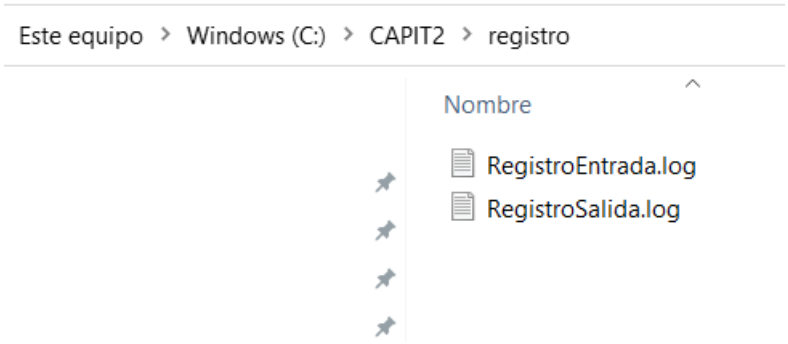
Se **entregará el proyecto Tarea3Recuperacion** en un **fichero comprimido .zip apellidosNombreTarea3Recuperacion.zip**

Tarea

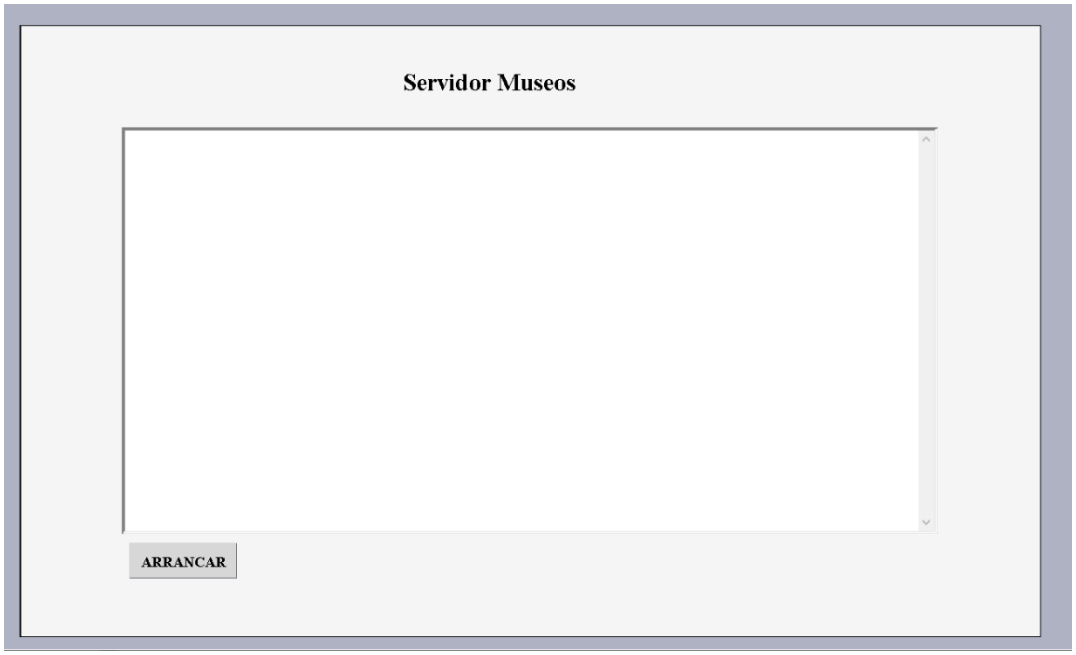
En la **tarea vamos a realizar mediante hilos la simulación de la entrada y salida** en un museo que tiene un aforo máximo. Los **Visitantes registraraban su salida y su entrada** en un fichero y **se quedarán esperando en caso de que el museo este lleno**. En esta tarea vamos a complementar la anterior con un modelo Cliente-Servidor. **El cliente se encargará de realizar las peticiones al servidor** para **entrar y salir** del museo. El **servidor se encargará de realizar los tramites** al estilo de la practica anterior y de **hacer esperar a los clientes** si el **museo esta lleno**.



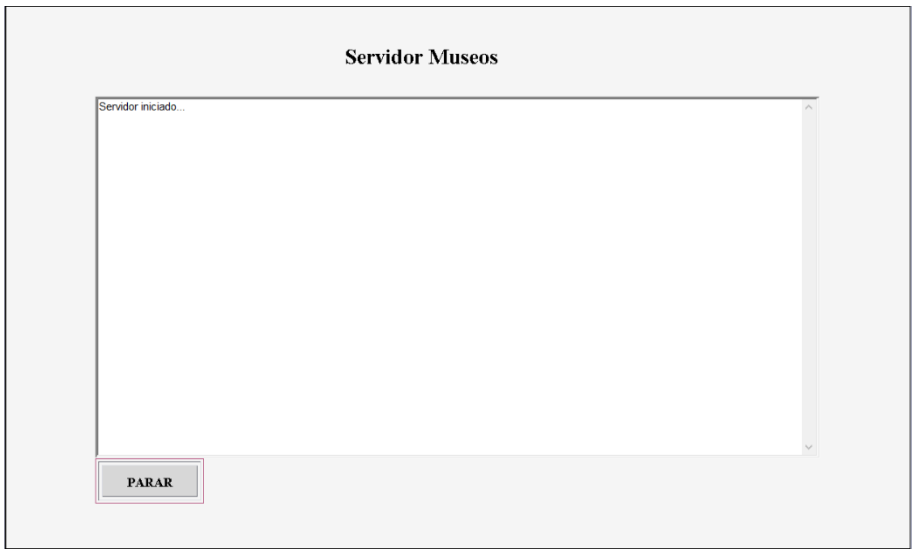
Mantenemos los **dos ficheros de registro uno de entrada y otro de salida.**

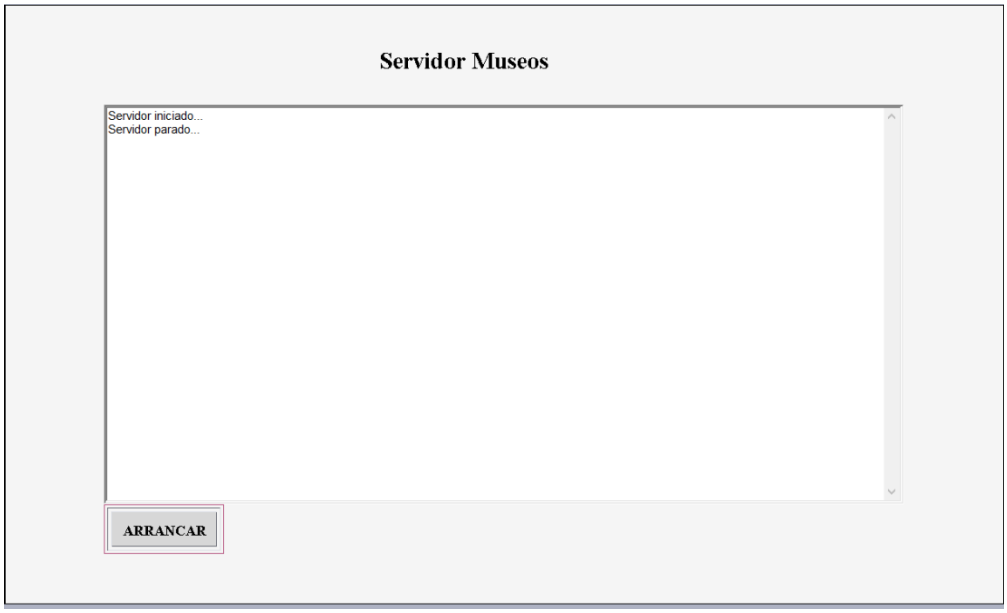


La **ventana de servidor** permite:

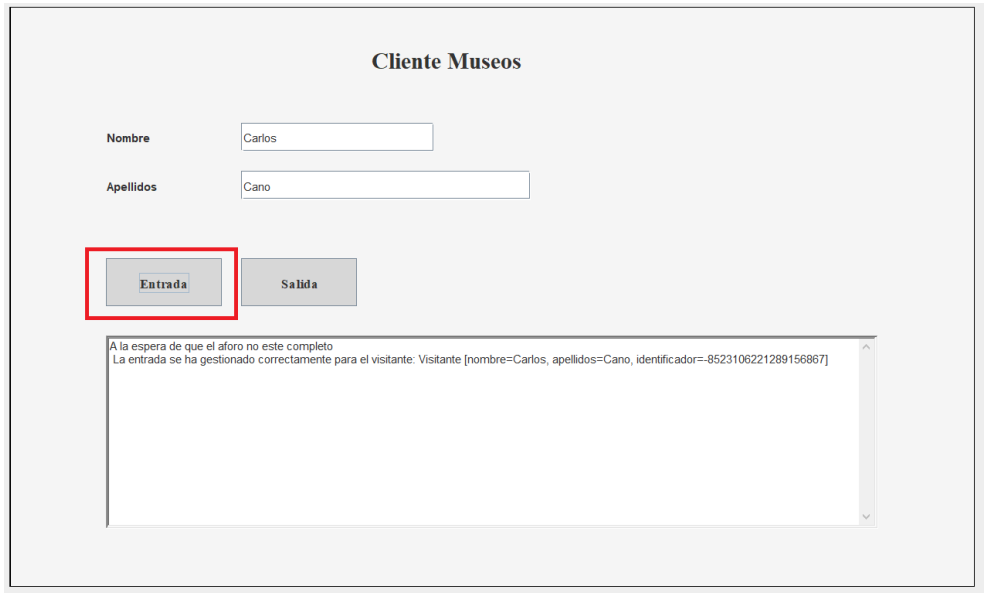


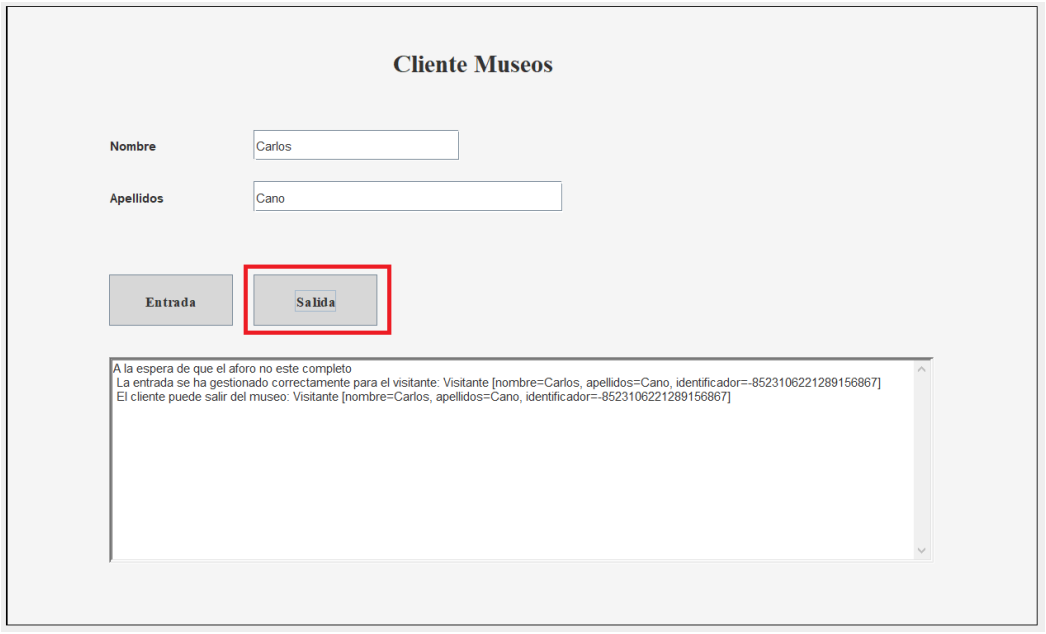
Arrancar y parar el servidor



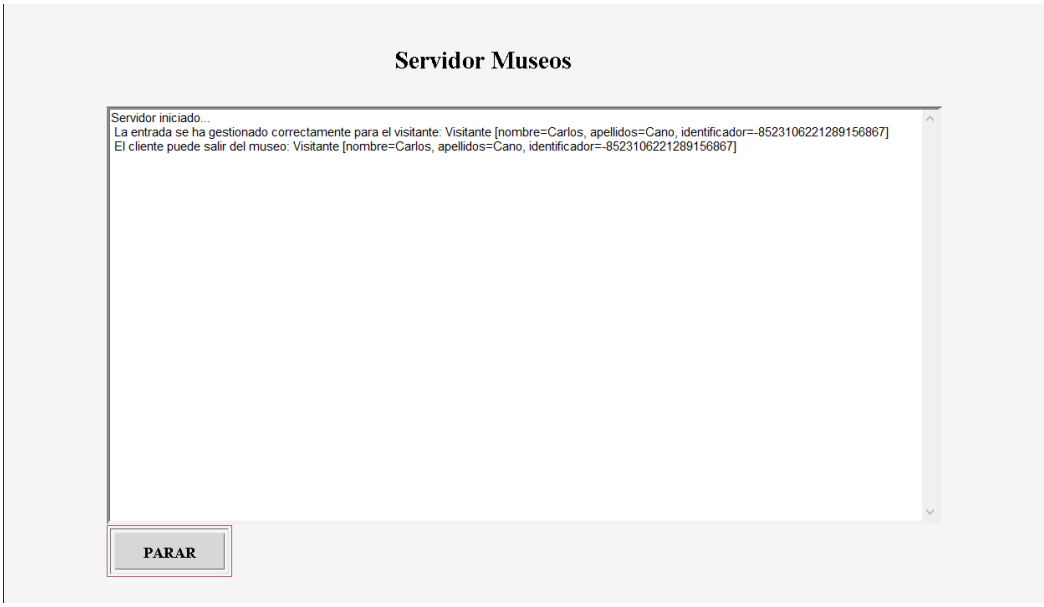


La **ventana cliente** realiza las **peticiones de entrada y salida** al museo.

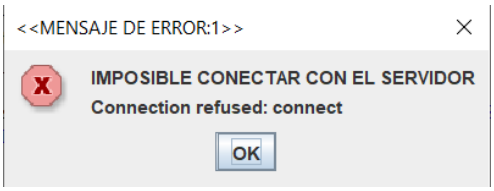




Que **se verá reflejado en la ventana** de **Servidor:**



Si el servidor no esta activo el programa mandará un mensaje de error:



Recursos

La clase Peticion

```
package servidormuseos.modelo;

import java.io.Serializable;


public class Peticion implements Serializable{


	private Visitante visitante ;
	private int tipoPeticion;

	public static final int ENTRADA=1;
	public static final int SALIDA=2;


	/**
	 *
	 */
	private static final long serialVersionUID = -2775003184799206666L;

	public Peticion() {


	}

	public Peticion(Visitante visitante, int tipoPeticion) {


		this.visitante= visitante;
		this.tipoPeticion=tipoPeticion;

	}

	public Visitante getVisitante() {
		return visitante;
	}

	public void setVisitante(Visitante visitante) {
		this.visitante = visitante;
	}

	public int getTipoPeticion() {
		return tipoPeticion;
	}

	public void setTipoPeticion(int tipoPeticion) {
```

```
            this.tipoPeticion = tipoPeticion;
        }



}
```

La clase Constantes

```
    public static final String PARAR = "PARAR";
    public static final String ARRANCAR = "ARRANCAR";
    public static final int TIEMPO = 2;
    public static final int SEGUNDO=1000;

    public static final String INICIADO= "Servidor iniciado..." ;

    public  static final String PARADO= "Servidor parado...";

    public static final String APLICACION_NOMBRE = " Aplicación Servidor Museos";
    public static final String FONT = "Serif";
```

Se debe implementar

Se debe implementar las comunicaciones de vuelta en la **clase principal AppCliente.java. Sólo tendréis que establecer las comunicaciones.** La **VentanaCliente** mandará un objeto **de tipo Peticion**, con dos opciones**: Entrada y Salida,** de manera que **la AppCliente se quedará a espera de las dos mensajes de texto de respuesta**, con **la entrada y la salida**. Las **respuestas** deben tener su **reflejo en VentanaAplicacion**.

AppCliente

```
public class AppCliente {

    private static Socket socket;

    private static final int PUERTO =7700;
    private static final String Host ="localhost";
    private static final JFrame ventana= null;

      /**
   * @param args the command line arguments
   */
  public static void main(String[] args) {


      try {
```

```
                socket = new Socket(Host, PUERTO);

                VentanaCliente.createVentanaCliente(socket);

                // Realizar los necesario para recibir las dos respuestas
                // de texto



            }  catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();



            }


    }

}
```

La ventana Cliente

```
public class VentanaCliente {


       public static final String ENTRAR = "Entrada";
       public static final String SALIR = "Salida";
       public static final int TIEMPO = 2;
       public static final int SEGUNDO=1000;

       public static final String APLICACION_NOMBRE = " Aplicación Servidor Museos";
       public static final String FONT = "Serif";




       private static boolean servidorParado=false;
       private static JFrame frame ;
       private static TextArea txtArea;
       private static Socket socket;
       private static Visitante visitante;

       private static JLabel lblNombre;
       private static JTextField txtNombre;
       private static JLabel lblApellidos;
       private static JTextField txtApellidos;
       private static ObjectOutputStream obs =null;


// Construye  una peticion de entrada
       private static Peticion crearPeticionEntrada() {
```

```
	}


	private static Peticion crearPeticionSalida() {



	}

	// Escribe un mensaje en la TextArea de ventana Cliente
	public static void addText(String mensaje) {




	}

// crea la ventana principal
// atentos a los listener de los botones que deben mandar las peticiones al servidor
	  public static JFrame createVentanaCliente (Socket socket) {

	}
```

La clase VentanaAplicacion

Desde la **ventana servidor se puede parar o arrancar el Servidor**. Se verán **reflejados** tanto los **mensajes de Entrada** del visitante (Cliente) como **los de Salida.**

```
public class VentanaAplicacion {

	private static JFrame frame;
	private static TextArea txtArea;

	public static void addText(String mensaje) {

		txtArea.append(mensaje + "\n");

	}

	public static JFrame createVentanaAplicacion() {



	}



	}
```

La clase ServidorMuseo

Debeis **controlar el arranque y la parada de servidor y que funcione correctamente.** Estas acciones **se realizarán en la VentanaAplicacion.**

```
public class ServidorMuseo extends Thread {


	private static boolean servidorParado=true;

	private static final int PUERTO =7700;
	private static ServerSocket servidor;

private static final String INICIADO= "Servidor iniciado..." ;

	private static final String PARADO= "Servidor parado...";

//Comprueba si el servidor está parado
	public static boolean estaServidorParado() {



}


	//Arranca el servidor
	public static void arrancarServidor() {




	}
	//Para el servidor
	// Cuando el servidor este parado debéis controlar en el método run
	// que se paran todos sus hilos y se cierra el Socket
	public static void pararServidor() {







	}


	public void run () {

		ServerSocket servidor;

		try {
		servidor = new ServerSocket(PUERTO);
		System.out.println(Constantes.INICIADO);
```

```
            RecursoMuseo recurso= new RecursoMuseo();

//Gestionar las peticiones con HiloPeticionServidor
//y un bucle para que el servidor no pare
// gestionar el cierre de hilos y recursos


}


}
```

HiloServidorPeticion

Gestiona las peticiones de entrada y salida. Si la **petición es de entrada lanzará** el hilo **EntradaMuseo**. En la **petición de salida** lanzará el **hilo SalidaMuseo**. El **Hilo debe parar cuando el servidor pare**.

```
package servidormuseos.hilos;


public class HiloServidorPeticion extends Thread {

        Socket clienteSocket = null;
        RecursoMuseo recurso;

        public HiloServidorPeticion(Socket clienteSocket, RecursoMuseo recurso) {

                this.clienteSocket = clienteSocket;
                this.recurso = recurso;

        }

        //Gestiona las peticiones tanto de entrada como de salida
        // Gestionará las dos peticiones de entrada y de salida.
        public void run() {

        }

}
```

Modificaciones

Ahora EntradaMuseo no debe lanzar a SalidaMuseo, lo realizará HiloPeticion. Se **encargarán** de **contestar al cliente que la entrada o salida se ha realizado correctamente**

con un mensaje de texto. **Pararán si el servidor ha sido parado**.

Los nuevos constructores para EntradaMuseo y SalidaMuseo son:

```
public EntradaMuseo(DataOutputStream dos,Visitante visitante, RecursoMuseo museo)

public SalidaMuseo(DataOutputStream dos, Visitante visitante, RecursoMuseo museo)
```

Pool de hilos

Para lanzar los hilos **debéis usar el commonPool de ForkJoinPool**.

Opcional

Integrar el sistema de borrado y estadísticas de la tarea 1.