TECSUP

GUÍA DE LABORATORIO 5

“Java: Operaciones de entrada/salida y sockets”

LABORATORIO

Objetivos:

- Crear canales de datos con Java.
- Usar canales de caracteres y bytes.
- Escribir programas que naveguen por el sistema de archivos.
- Leer y escribir en archivos.
- Serializar objetos.
- Implementar Sockets

Requisitos:

- PC con Sistema Operativo con soporte a Java.
- IDE para Java.

Introducción:

Los Thread son aplicaciones en Java que nos permiten realizar subprocesos en forma paralela. Esa característica nos permite entender como una aplicación puede atender simultáneamente a varias aplicaciones, el cual es el principio básico de los servidores web.

Seguridad:

- Ubicar maletines y/o mochilas en el gabinete al final de aula de laboratorio.
- No ingresar con líquidos ni comida al aula de laboratorio.
- Al culminar la sesión de laboratorio, apagar correctamente la computadora y el monitor.

Preparación:

Durante el desarrollo de los temas de clase se tendrán ejercicios explicativos en cada uno de los puntos, ello le dará a la sesión una interacción de la teoría y la parte práctica, ya que en todo el momento el alumno podrá comprobar en su propia PC, todos los ítems del manual.

Procedimiento y Resultados:

Ejercicio: Book S.A.

La empresa Book SA se dedica a la venta de libros y para ello cuenta con un selecto grupo de vendedores. Cada vendedor cuenta con una cartera de clientes a quienes debe atender adecuadamente. Un cliente puede ser una corporación o una persona. Para los clientes corporativos es necesario registrar el nombre de un contacto, quien es la persona encargada de realizar los pedidos de los libros. Además, se debe registrar la dirección de la sede principal de la compañía y el lugar (almacén) donde se entregarán los libros. Los clientes corporativos poseen una calificación y un límite de crédito para realizar sus compras. Los clientes personales sólo pueden realizar sus compras al contado o con tarjeta de crédito. Si el pago es realizado con una tarjeta de crédito el número debe ser registrado para compras posteriores.
Book SA posee personal administrativo y vendedores para poder realizar sus operaciones. Los vendedores reciben un sueldo básico mensual y un porcentaje de comisión por el total de las ventas efectuadas durante el mes. El personal administrativo recibe un sueldo mensual acorde con el cargo administrativo que posee y, además, cuenta con un seguro tepagado por la compañía.

TEMA 1: ENTRADA/ SALIDA

1. Escribir un programa que compruebe si un archivo existe. Si existe, entonces, mostrará su nombre, ruta, ruta absoluta, si se puede escribir en él, si se puede leer en él y cuantos bytes ocupa el archivo.

```
package laboratorio6;

import java.io.File;

public class CompruebaArchivo {

   public static void main(String args[]){

      File f = new File("C:\\temario.txt");
      System.out.println("Nombre: " + f.getName());
      System.out.println("Ruta : " + f.getPath());
      System.out.println("Ruta Absoluta: " + f.getAbsolutePath());

      if(f.exists()){
         System.out.println("Archivo sí existe!");
         System.out.println((f.canRead()) ? "Sí se puede leer" : "");
         System.out.println((f.canWrite()) ? "Sí se puede escribir" : "");
         System.out.println("La longitud del archivo es de " + f.length() + " bytes");
      } else {
         System.out.println("El archivo no existe");
      }
   }
}
```

2. Escribir un programa que liste el contenido de un directorio.

```
package laboratorio6;

import java.io.*;

public class ListarDirectorio {

   public static void main(String args[]){
      File directorio = new File("C:\\ ");
      if ( (directorio.exists()) && (directorio.isDirectory())){
         String[] lista = directorio.list();
         for(int i=0; i<lista.length; i++){
            System.out.println(lista[i]);
         }
      } else {
         System.out.println("El directorio no existe.");
      }
   }
}
```

CANALES DE ALTO NIVEL

```
package laboratorio6;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class EscribirArchivo {

	public static void main(String[] args) {
		FileWriter fw = null;
		BufferedWriter bw = null;
		try {
			fw = new FileWriter("/home/alumno/test");
			bw = new BufferedWriter(fw);
			bw.write("Es una linea");
			bw.newLine();
			bw.write("Es otra linea");
			bw.close();
			fw.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

```
package laboratorio6;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class LeerArchivo {

	public static void main(String[] args) {
		FileReader fr = null;
		BufferedReader br = null;
		String sInput = null;
		try {
			fr = new FileReader("/home/alumno/test");
			br = new BufferedReader(fr);
			while ((sInput = br.readLine()) != null) {
				System.out.println(sInput);
			}
			br.close();
			fr.close();
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

Ejercicio

Escribir un programa que agregue el contenido a un archivo. Primero deberá leer lo que existe en el archivo, luego procederá a escribir en el archivo.

CANALES DE OBJETOS

```
package laboratorio6;

import java.io.Serializable;

public class Persona implements Serializable {

   private String nombre;
   private int edad;

   public Persona(String nombre, int edad){
      this.nombre = nombre;
      this.edad = edad;
   }

   public String toString(){
      return "Nombre: " + nombre + "; Edad: " + edad;
   }
}
```

```
package laboratorio6;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class SerializarPersona {

   public static void main(String args[]){
      FileOutputStream archivo = null;
      ObjectOutputStream salida = null;

      Persona p1 = new Persona("Alonso", 23);

      try {
         archivo = new FileOutputStream("/home/alumno/Persona.ser");
         salida  = new ObjectOutputStream(archivo);
         salida.writeObject(p1);
         salida.flush();
      } catch (IOException e){
         System.out.println("Imposible crear el archivo o escribir en él");
      } finally {
         try {
            salida.close();
         } catch (IOException e){
            System.out.println("No pudo cerrarse el canal");
         }
      }
   }
}
```

```
package laboratorio6;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class DeserializarPersona {

   public static void main(String args[]){

      File archivo = new File("/home/alumno/Persona.ser ");
      try {
         FileInputStream fis = new FileInputStream(archivo);
         ObjectInputStream ois = new ObjectInputStream(fis);
         Persona p1 = (Persona)ois.readObject();
         ois.close();
         fis.close();

         System.out.println("Persona recuperada: " + p1.toString());
      } catch (IOException e){
         System.err.println(e.toString());
      } catch (ClassNotFoundException e){
         System.err.println(e.toString());
      }
   }
}
```

TEMA 2: SOCKETS

EJERCICIO 1

Implementa la siguiente comunicación basado en sockets.

SERVIDOR: Bienvenido, ¿Cómo te llamas?
CLIENTE: Mi nombre es David Rodríguez

```
package laboratorio3.sockets;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class Servidor {
	private int port;

	public Servidor(int port) {
		this.port = port;
	}

	public void ejecutar() {
		try {
			// Creamos un servidor de Socket
			ServerSocket server = new ServerSocket(port);

			System.out.println("Servidor iniciado...");
			Socket cliente = server.accept();

			// Crear los canales de lectura y escritura
			PrintWriter out = new PrintWriter(cliente.getOutputStream(), true);
			BufferedReader in = new BufferedReader(new
InputStreamReader(cliente.getInputStream()));

			// COMUNICACION 1
			out.println("SERVIDOR: Bienvenido, ¿Cómo te llamas? ");

			// COMUNICACION 2
			String resp2 = in.readLine();
			System.out.println(resp2);

			/* Cerramos el canal */
			in.close();
			out.close();
			cliente.close();

		} catch (Exception e) {
			System.out.println("Error : " + e.getMessage());
			e.printStackTrace();
		}
```

```
        }
}
```

```java
package laboratorio3.sockets;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class Cliente {

	private String host;
	private int port;

	public Cliente(String host, int port) {
		this.host = host;
		this.port = port;
	}

	public void doConectar() {
		try {
			Socket cliente = new Socket(host, port);

			// Crear los canales de lectura y escritura
			BufferedReader in = new BufferedReader(new
InputStreamReader(cliente.getInputStream()));
			PrintWriter out = new PrintWriter(cliente.getOutputStream(), true);

			// COMUNICACION 1
			String msg = in.readLine();
			System.out.println(msg);

			// COMUNICACION 2
			out.println("CLIENTE: David Rodríguez");

			// Cerrar canales
			in.close();
			out.close();
			cliente.close();
		} catch (Exception e) {
			System.out.println("Error Cliente : " + e.getMessage());
		}
	}
}
```

```java
package laboratorio3.sockets;

public class ServidorMain {

	public static void main(String[] args) {
		Servidor s = new Servidor(2365);
		s.ejecutar();
	}

}
```

```java
package laboratorio3.sockets;

public class ClienteMain {

	public static void main(String[] args) {
		Cliente c = new Cliente("localhost", 2365);
		c.doConectar();
	}
}
```

EJERCICIO 2

Modificar el servidor para que esté siempre ejecutándose y atendiendo a muchos clientes uno por uno.

```
package laboratorio3.sockets;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class Servidor {
    private int port;

    public Servidor(int port) {
        this.port = port;
    }

    public void ejecutar() {

        try {
                // Creamos un servidor de Socket
                ServerSocket server = new ServerSocket(port);

                System.out.println("Servidor iniciado...");

                while(true){
                        Socket cliente = server.accept();

                        // Crear los canales de lectura y escritura
                        PrintWriter out = new PrintWriter(cliente.getOutputStream(), true);
                        BufferedReader in = new BufferedReader(new InputStreamReader(
                                        cliente.getInputStream()));

                        // COMUNICACION 1
                        out.println("SERVIDOR: Bienvenido, ¿Cómo te llamas? ");

                        // COMUNICACION 2
                        String resp2 = in.readLine();
                        System.out.println(resp2);

                        /* Cerramos el canal */
                        in.close();
                        out.close();
                        cliente.close();
                }

        } catch (Exception e) {
                System.out.println("Error : " + e.getMessage());
                e.printStackTrace();
```

```
        }

    }
}
```

EJERCICIO 3

Modificar el servidor para que esté siempre ejecutándose y atendiendo a muchos clientes en simultáneo.

```
package laboratorio3.sockets;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class ServidorHilo extends Thread {

	private Socket cliente;

	public ServidorHilo(Socket cliente) {
		this.cliente = cliente;
	}

	public void run() {
		try {
			// Crear los canales de lectura y escritura
			PrintWriter out = new PrintWriter(cliente.getOutputStream(), true);
			BufferedReader in = new BufferedReader(new InputStreamReader(
					cliente.getInputStream()));

			// COMUNICACION 1
			out.println("SERVIDOR: Bienvenido, ¿Cómo te llamas? ");

			// COMUNICACION 2
			String resp2 = in.readLine();
			System.out.println(resp2);

			/* Cerramos el canal */
			in.close();
			out.close();
			cliente.close();

		} catch (IOException e) {
			e.printStackTrace();
		} finally {
			try {
				if (cliente != null)
					cliente.close();
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
	}
}
```

```
package laboratorio3.sockets;

import java.net.ServerSocket;
import java.net.Socket;

public class Servidor {
	private int port;

	public Servidor(int port) {
		this.port = port;
	}

	public void ejecutar() {

		try {
			// Creamos un servidor de Socket
			ServerSocket server = new ServerSocket(port);

			System.out.println("Servidor iniciado...");

			while(true){
				System.out.println("Esperando ...");
				Socket cliente = server.accept();

				ServidorHilo ts = new ServidorHilo(cliente);
				ts.start();

			}

		} catch (Exception e) {
			System.out.println("Error : " + e.getMessage());
			e.printStackTrace();
		}
	}
}
```

EJERCICIO PROPUESTO 1

Escribir un servidor basado en sockets que permita realizar las operaciones SUMA, RESTA, MULTIPLICACIÓN o DIVISIÓN.

El servidor de Operaciones debe estar siempre operativo y debe atender a múltiples clientes al mismo tiempo.

Los datos enviados por el cliente deben contar con algún algoritmo de encriptación.

EJERCICIO PROPUESTO 2

El club deportivo “Sport Elegante”, necesita controlar la venta de las entradas a los partidos de futbol, ya que todos los asientos son numerados y es importante no generar duplicidad en la venta. Para ello, se le encarga a usted la implementación de un servidor basado en sockets que permita manejar las siguientes opciones:

1. Ingreso
2. Separación/Venta de Entradas
3. Relación de asientos disponibles
4. Relación de asientos ocupados
5. Salir

- En caso de Ingreso, el usuario debe ingresar: número de asiento, que tiene una estructura de Tribuna-Fila-Columna, (ejemplo OCC-F08-C09). el servidor registrará el asiento, su estado (desocupado) y emitirá un mensaje “Asiento REGISTRADO”.
- En caso de Separación/Venta, el usuario debe ingresar el número de asiento, el servidor ubicará en el arreglo al asiento. Si el asiento está desocupado, le cambiará el estado y emitirá un mensaje “Asiento xxxx VENDIDO”, caso contrario emitirá un mensaje “El Asiento xxxx está OCUPADO.
- En caso de Relación de asientos disponibles, el servidor emitirá la relación de asientos disponibles.
- En caso de Relación de asientos ocupados, el servidor emitirá la relación de asientos ocupados.
- En caso de Salir, el servidor culminará su proceso.