LABORATORIO SENSOR DE TEMPERATURA ARDUINO - JAVA

ARQUITECTURA DE SOFTWARE

UNIVERSIDAD CATÓLICA DE COLOMBIA
ARQUITECTURA DE SOFTWARE
INGENIERÍA DE SISTEMAS
BOGOTÁ D.C.
2017

SENSOR DE TEMPERATURA ARDUINO – JAVA

Con el fin de obtener el valor actual de la temperatura ambiente, vamos a utilizar un sensor de temperatura acompañado de un Arduino. Esto nos va a generar un valor en el puerto serial de nuestro equipo, el cual vamos debemos enviar a una aplicación de escritorio en JAVA, para así, representar de manera gráfica el valor de la temperatura.

Herramientas:

- **Arduino**.
  Se trata de un microcontrolador, una placa, un pequeño sistema de procesamiento. Es una poderosa plataforma de hardware libre, ideal para proyectos de electrónica que van desde el encendido de un LED hasta un complejo sistema de domótica para controlar una casa entera.

- **Sensor LM35.**
  Es un sensor de temperatura digital. El LM35 es un integrado con su propio circuito de control, que proporciona una salida de voltaje proporcional a la temperatura.

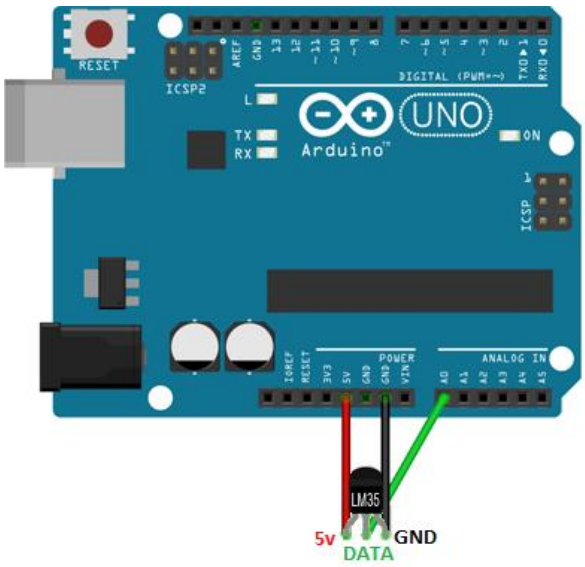
- **Eclipse JAVA.**
  Es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar aplicaciones. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT).

Procedimiento:

1. Montaje eléctrico del sensor de temperatura.
2. Programación en Arduino para obtener el valor por medio del puerto serial.
3. Crear las tablas y atributos en la base de datos donde registraremos los valores obtenidos.
4. Creación de proyecto en JAVA.
5. Importar controladores para utilizar librerías de comunicación serial con Arduino.
6. Importar controladores para realizar la conexión con la base de datos PostgreSQL.
7. Configurar archivo de persistencia.
8. Utilizar las herramientas del Framework JPA para generar las entidades a partir de las tablas creadas en la base de datos.
9. Utilizar la instancia Entity Manager para realizar la inserción de registros en la base de datos.
10. Utilizar la biblioteca grafica de java swing para el diseño de la aplicación de escritorio en la cual vamos a visualizar el valor de la temperatura.
11. Visualización de los valores de la temperatura.

DESARROLLO DEL PROYECTO

1. **Montaje eléctrico del sensor.**
   En el sensor LM35, los pines extremos son para alimentación, mientras que el pin central proporciona la medición en una referencia de tensión, a razón de 10mV/ºC el cual a través de una operación matemática hará referencia al valor de la temperatura. En cuanto al Arduino cuenta una fuente de voltaje de 5v, conexiones de tierra (GND) y adicionalmente posee varias interfaces de entrada y salida de datos, las cuales vamos a utilizar para poder obtener la información generada por el sensor de temperatura.
   Para nuestro proyecto utilizaremos las interfaces de 5V y GND para proporcionar la energía necesaria al sensor y utilizaremos el pin A0 para la transferencia de datos entre el sensor y el Arduino.



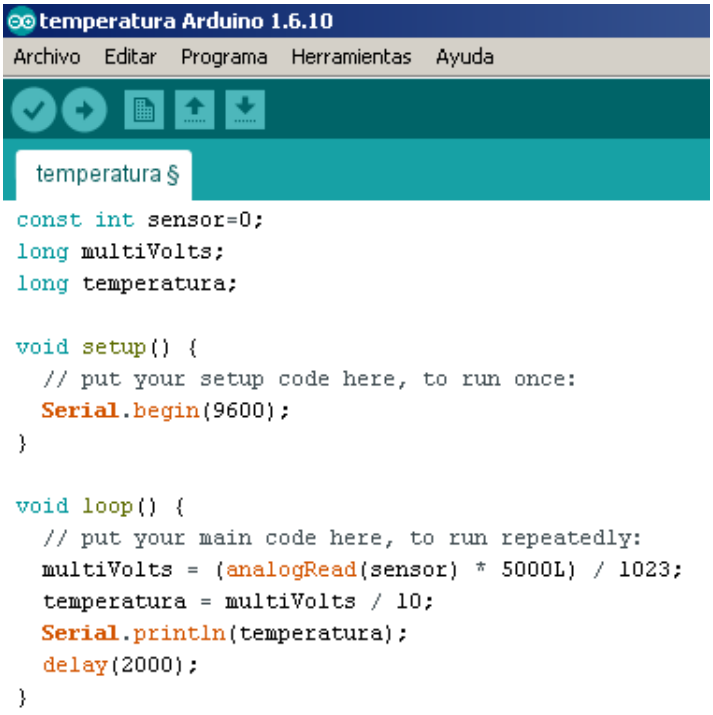
2. **Programación en Arduino.**
   Arduino tiene un IDE propio, el cual soporta el lenguaje de programación C++. Para la elaboración del proyecto se definió el puerto de entrada de datos A0 para la lectura de la temperatura y a través de la función analogRead obtenemos el valor desde el sensor. Sin embargo, el valor emitido por el sensor LM35 está dado en milivoltios por lo cual es necesario realizar la siguiente conversión:

   Entrada de energia de Arduino: 5000mV
   Rango de valores de AnalogRead: 0 – 1023
   X = Valor de la temperatura en mV

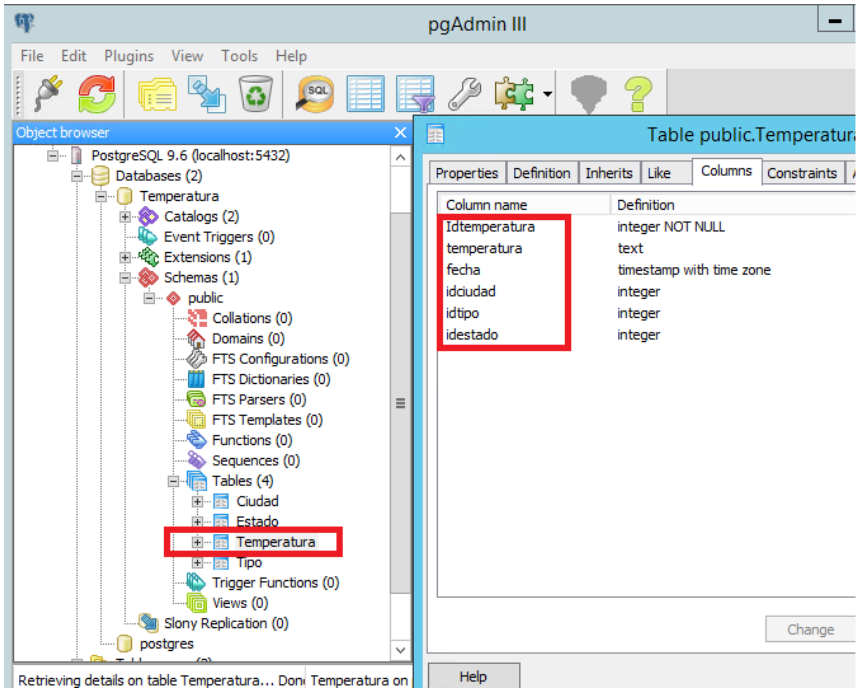
   x= (analogRead * 5000) / 1023

   Adicional a esto, el sensor nos entrega el valor real en milivoltios multiplicado por 10, es decir que 270mV corresponden a 27ª. Por lo cual el valor de la temperatura es:

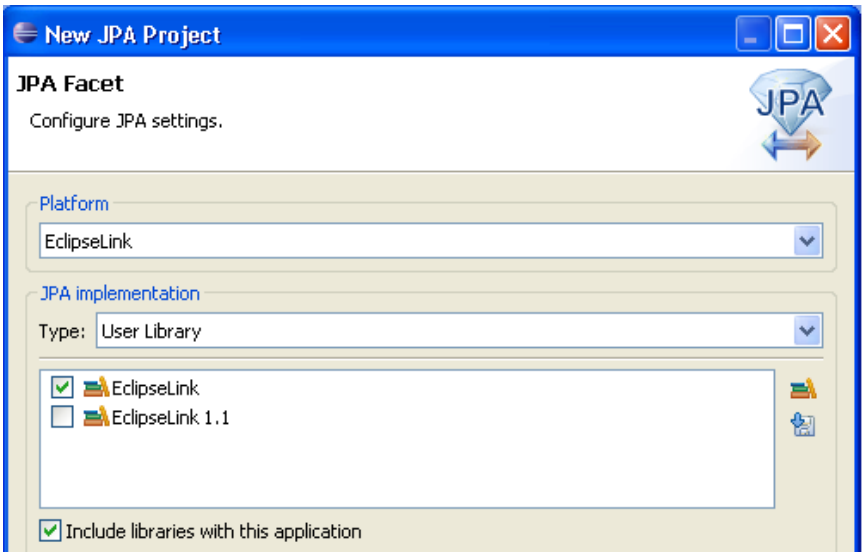
   temperatura = x/10



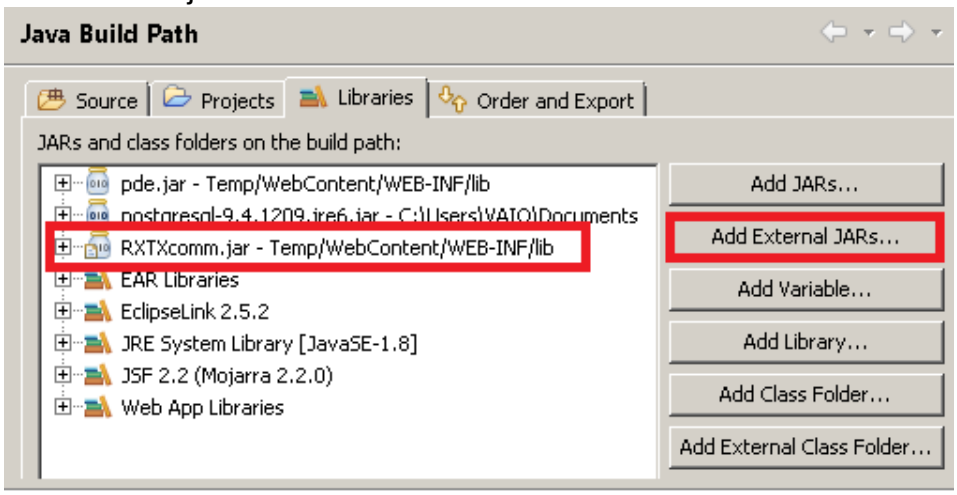
3. **Configuración base de datos.**
   Para nuestro proyecto utilizaremos una base de datos PostgreSQL alojada en un servidor remoto en Amazon Web Services. Allí vamos a realizar la creación de las tablas y atributos necesarios para registrar los datos del Arduino de acuerdo a los requerimientos funcionales establecidos (Temperatura, fecha, hora, ciudad).



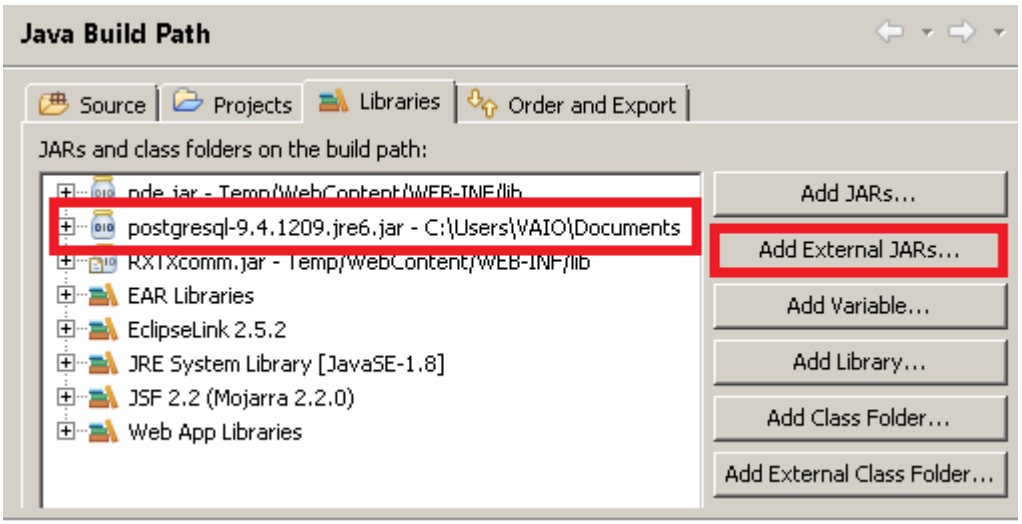
4. **Crear proyecto en JAVA.**
   En el Navegador o en el Explorador de proyectos, seleccionamos **Archivo> Nuevo> Proyecto** y como tipo de proyecto elegimos **JPA** (Java Persistence API) que es la API de persistencia que utiliza JAVA, este framework maneja los datos relaciones para interactuar con las bases de datos. Luego, el asistente nos preguntara sobre la plataforma que vamos a utilizar para la implementación de JPA, en este caso utilizaremos **EclipseLink** que viene de manera predeterminada en Eclipse. Cuando finalicemos el asistente ya tendremos nuestro proyecto creado con los directorios y configuración por defecto.



5. **Importar controlador para Arduino.**
   Para poder utilizar Java y Arduino necesitamos utilizar una librería especial en Java llamada RXTX. Esta librería es requerida por Java para enviar y recibir información a través del puerto serie. Una vez tengamos el archivo descargado, presionamos clic derecho sobre nuestro proyecto y seleccionamos la configuración de **JAVA Build Path.** Allí presionamos la opción **Add External JARs…** y seleccionamos el archivo RXTXcomm.jar desde la ruta en la cual tenemos el archivo.



6. **Importar controlador para PostgreSQL.**
   Al igual que en el paso anterior, es necesario dirigirnos a la página oficial de PostgreSQL y descargar el controlador más reciente, el cual es necesario para establecer la comunicación entre JAVA y nuestra base de datos.



7. **Configurar archivo de persistencia.**
   Las unidades de persistencia se definen en el archivo persistence.xml, que está situado en el directorio META – INF. Un archivo persistence.xml incluye la definición de la comunicación con la base de datos (ubicación, usuario, contraseña y controlador). De esta manera es posible utilizar las instancias Entity Manager del Framework JPA.

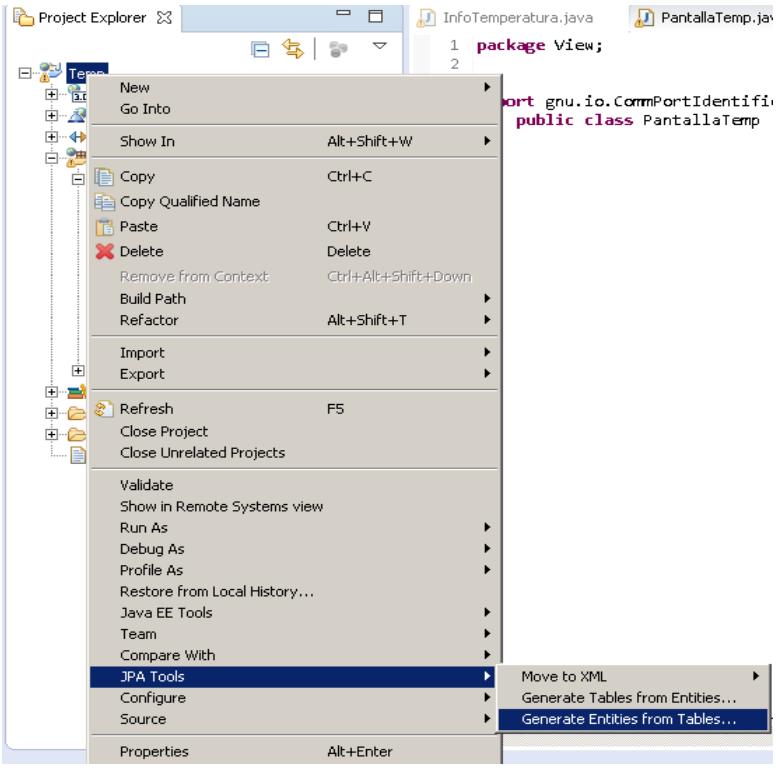
   En el siguiente grafico podemos evidenciar como se realiza la configuración del archivo **persistence.xml:**

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">

    <persistence-unit name="Temp" transaction-type="RESOURCE_LOCAL">
        <class>Model.Temperatura</class>
        <properties>
            <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://52.43.14.33:5432/Temperatura"/>
            <property name="javax.persistence.jdbc.user" value="postgres"/>
            <property name="javax.persistence.jdbc.password" value="123456"/>
            <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver"/>
        </properties>
    </persistence-unit>
</persistence>
```

   Es importante definir el nombre de la unidad de persistencia, en este caso “Temp”, ya que será el identificador que utilizaremos para indicarle Entity Manager Factory la conexión con la base de datos.

8. **Generar entidades JPA.**

Una vez tengamos configurado nuestro archivo de persistencia y la comunicación con nuestra base de datos sea exitosa, podemos generar las entidades de manera automática con la información correspondiente a la base de datos. Esto lo podemos realizar gracias a las herramientas que nos ofrece JPA. Para ello presionamos clic derecho sobre la carpeta raíz de nuestro proyecto, desplegamos la opción **JPA tools** y seleccionamos **Generate Entities from Tables**.



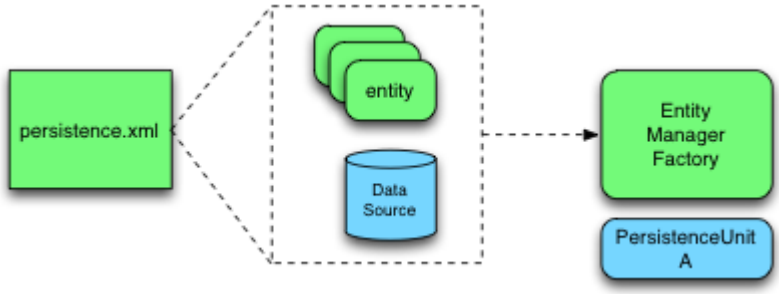
Se desplegará un asistente en donde se encuentran las tablas que tenemos creadas en la base de datos. De esta manera, podemos seleccionar las entidades que se requieren.

Una vez se seleccionen las tablas necesarias y finalicemos el asistente, se generaran las entidades en nuestro proyecto. Tal como se muestra a continuación, las entidades contienen los mismos atributos de las tablas en la base de datos y adicionalmente, ya se encuentran creados los **Getters & Setters** para cada uno de ellos.

```
package Model;

import java.io.Serializable;

@Entity
@Table(name="\"Temperatura\"")
@NamedQuery(name="Temperatura.findAll", query="SELECT t FROM Temperatura t")
public class Temperatura implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @Column(name="\"Idtemperatura\"")
    private Integer idtemperatura;
    private Timestamp fecha;
    private Integer idciudad;
    private Integer idestado;
    private Integer idtipo;
    private String temperatura;

    public Temperatura() {
    }

    public Integer getIdtemperatura() {
        return this.idtemperatura;
    }

    public void setIdtemperatura(Integer idtemperatura) {
        this.idtemperatura = idtemperatura;
    }
```

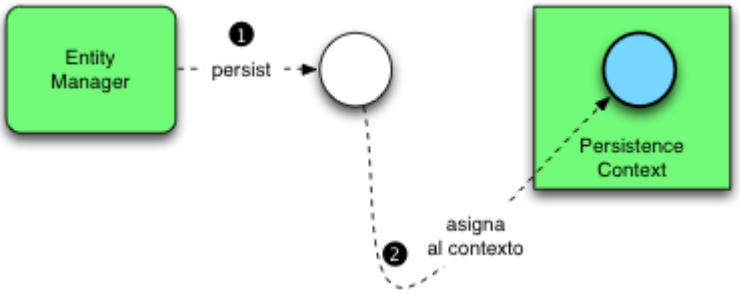
9. Entity Manager Factory y Entity Manger.

En un primer lugar un EntityManagerFactory es único y es con el que nosotros gestionamos todas las entidades.



Una vez disponemos de un EntityManagerFactory este será capaz de construir un objeto de tipo EntityManager que como su nombre indica gestiona un conjunto de entidades u objetos. En principio estas entidades son objetos POJO (Plain Old Java Object) normales con los cuales estamos trabajando en nuestro programa Java .El EntityManager será el encargado de guardarlos, eliminarlos y demas en la base de

datos. Para ello define otro concepto adicional “PersistenceContext”. Este concepto hace referencia a los objetos que han sido manipulados por el EntityManager y se encuentran controlados por él. Para conseguir que alguno de nuestros objetos pase a ubicarse dentro del PersistenceContext bastará con invocar a alguno de los métodos típicos del EntityManager y asi realizaremos la consulta o insercion de los datos.



Para nuestro proyecto creamos la clase ***InfoTemperatura*** en la cual se realizó la instanciación del Entity Manager y en Entity Manager Factory, indicando la unidad de persistencia que se configuro en el archivo persistence.xml, en este caso es **Temp.**

```
public class InfoTemperatura
{
    EntityManager entityManager = null;
    EntityManagerFactory emf =Persistence.createEntityManagerFactory("Temp");
    public Boolean registro( String tempvalue)
```

En esta misma clase definimos la función registro, el cual nos servirá para realizar la inserción de la información en la base de datos. En la firma de la función vamos a definir una variable para el valor de la temperatura que se tomara desde la clase donde realizamos la comunicación con el Arduino. Inicialmente debemos crear una instancia del Entity Manager e iniciar la transacción con **getTransaction().begin()** e instanciar la entidad que generamos automáticamente con la herramienta de JPA, que para nuestro proyecto es “***temperatura”***. Luego traeremos el valor del id desde la base de datos y le sumaremos uno con el fin de continuar con el consecutivo que se tiene, también capturamos el valor de la fecha y hora del momento en el que se está realizando la operación e ingresamos los demás datos que se necesitan para nuestro registro. Para finalizar la transacción utilizamos **persist.(temperatura)** y confirmamos la inserción de datos con **getTransaccion().commit().**

```
public Boolean registro( String tempvalue){
    System.out.println("Temperatura ambiente...");
    int Maxid=0;
    try{
        entityManager = emf.createEntityManager();
        entityManager.getEntityManagerFactory().getCache().evictAll();
        entityManager.getTransaction().begin();
        TypedQuery<Temperatura> query = entityManager.createQuery("select t from Temperatura t", Temperatura.class);
        List<Temperatura> Temprtr = query.getResultList();
        for(Temperatura t :Temprtr){ Maxid = t.getIdtemperatura();  }
        Temperatura temperatura = new Temperatura();
        Maxid += 1;

        temperatura.setIdtemperatura(Maxid);
        temperatura.setIdtipo(1);
        temperatura.setIdestado(1);
        temperatura.setIdciudad(1);
        temperatura.setTemperatura(tempvalue);
        fecha = new java.sql.Timestamp(Calendar.getInstance().getTime().getTime());
        temperatura.setFecha(fecha);

        entityManager.persist(temperatura);
        entityManager.getTransaction().commit();//

        }catch(NoResultException e){ System.out.println(e);
            entityManager.close();
        }
    entityManager.close();
    return true;
}
```

10. Diseño visual de la aplicación de escritorio.

Para el diseño visual de nuestra aplicación de escritorio utilizaremos **Swing**, la cual es una biblioteca gráfica para Java que incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas. JFrame será la base para la aplicación principal, la utilizaremos para la elaboración de las ventanas y haremos uso de algunos componentes adicionales como JButton (botones), Jpanel (contenedores) y Jlabel (cuadros de texto). Adicionalmente, haremos uso de hilos (Threads) para establecer la comunicación serial y realizar la lectura de datos desde Arduino.

```
public PantallaTemp()
{
    setTitle("Sensor de Temperatura");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 636, 365);
    contentPane = new JPanel();
    contentPane.setBackground(Color.WHITE);
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);
    timer = new Thread(new ImplementoRunnable());
    timer.start();
    timer.interrupt();
```

En nuestro proyecto definimos un botón llamado conectar, el cual tiene la acción de establecer la comunicación serial con el Arduino. Aquí debemos abrir la conexión con el puerto y definir los parámetros de tasa de transferencia de datos, bits de comunicación, bits de parada y paridad.

```
Conectar = new JButton("Conectar");
Conectar.addActionListener(new ActionListener()
{   public void actionPerformed(ActionEvent e)
    {   while (portEnum.hasMoreElements())
        {   portId = (CommPortIdentifier) portEnum.nextElement();
            for (String portName : PORT_NAMES)
            {   try
                    {       serialPort = (SerialPort) portId.open("puerto serial", TIME_OUT);
                            serialPort.setSerialPortParams(DATA_RATE, SerialPort.DATABITS_8,
                                    SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
                            serialPort.setDTR(true);
                             out = serialPort.getOutputStream();//Salida de java
                             Input = serialPort.getInputStream(); //Entrada de java
                             Salir.setEnabled(true);
                             Conectar.setEnabled(false);
                             timer.resume(); //Esta variable inici el metodo ImplementoRunnable
                    }
                    catch (PortInUseException e1)
                    {
                        e1.printStackTrace();
                    }
                    break;
            }}}};
```

En el metodo ***ImplementoRunnable,*** inicialmente instanciaremos la comunicación con la clase que contiene el EntityManager y realizaremos la lectura e impresión de los datos desde arduino. Para ello vamos a validar si hay datos disponibles y en caso de que existan, los almacenaremos en el arreglo chunk para posteriormente convertirlos al String tempaux que sera el valor final de la temperatura. Este dato lo imprimimos en la ventana (**ventanatemp.setText(tempaux)**)para que se visualice en la aplicación y también en la consola utilizando el **System.out.print(tempaux)**. Para que el valor quede registrado en la base de datos utilizaremos la funcion ***registro*** enviando el parámetro de la temperatura, tal como se explicó en el numeral **9** del presente documento.

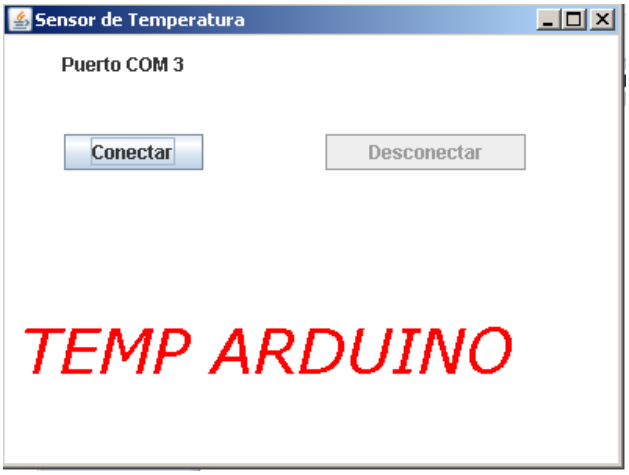
```
private class ImplementoRunnable implements Runnable
{   InfoTemperatura it = new InfoTemperatura();
    public void run()
    {   while(true)
        {   try
            {   out.write('T'); // enviamos el dato al puerto
                Thread.sleep(100);
                int available = Input.available();  // Cuantos datos hay disponibles
                byte chunk[] = new byte[available]; // Crea el arreglo para el numero de datos
                Input.read(chunk, 0, available); // Lee la informacion del puerto serial y la guarda en el arreglo
                String tempaux = new String(chunk); // Convierte los valores ASCII de chunk a String
                ventanatemp.setText(tempaux);
                System.out.print(tempaux);
                it.registro(tempaux);
             repaint();
            } catch (Exception e1) {
 } } } }
```

Finalmente, crearemos la clase ***main*** en donde instanciaremos la clase ***PantallaTemp*** y definiremos el tamaño de la ventana de nuestra aplicación.

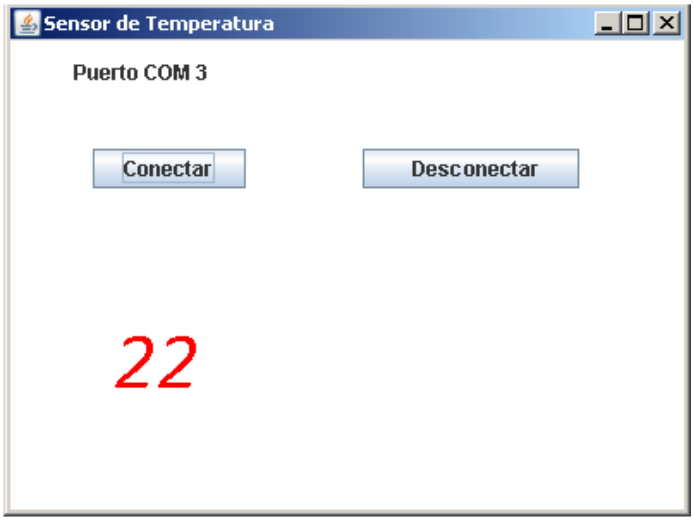
```
public class main {

    public static void main(String[] args) {
       PantallaTemp v= new PantallaTemp();
        v.setVisible(true);
        v.setSize(400,300);
    }

}
```

11. Visualización de la temperatura.

Cuando realicemos la compilación del código, se ejecutara una aplicación de JAVA la cual nos permitirá conectarnos al puerto de comunicación serial.



Cuando realicemos la conexión presionando el boton “Conectar”, el programa empezara a capturar los datos desde el puerto y se mostraran en nuestra aplicación de escritorio y tambien a traves de la consola de compilacion de JAVA.



BIBLIOGRAFÍA

Interface EntityManagerFactory, javax.persistence.
http://docs.oracle.com/javaee/7/api/javax/persistence/EntityManagerFactory.html

Tutorialspoint, JPA – Administradores de la Entidad.
http://www.tutorialspoint.com/es/jpa/jpa_entity_managers.htm

JavaTPoint, Java Swing Tutorial. http://www.javatpoint.com/java-swing