Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Лабораторна робота № 2

Тема: “Відображення інформації про розподілену вимірювальну систему”

Виконав:
ст. гр. ІПм-16-2
Лубковський А. А.
Перевірив:
Лютак І.З.

Івано-Франківськ

2016

Мета: Освоїти відображення інформації про розподілену вимірювальну систему.

Завдання

Створити 20 класів, що повинні відповідати давачам розподіленої вимірювальної системи. Кожен клас повинен виконуватись у окремому потоці (Thread). Клас повинен чекати поки до нього не звернуться за даними. При запиті клас повинен надати дані:

- Температура - випадкове число
  - від -20 до +5 у випадку холодної пори року,
  - від +5 до +40 у випадку теплої пори року. (Пора року задається у класі.)
- Вологість - випадкове число від 20% до 100%
- Координати у форматі GPS
- Номер давача, що проводив вимірювання, наприклад:
  - T20I345 - серійний номер давача температури,
  - B486EE - давача вологості

Створити головну програму, що опитуватиме давачі та записуватиме інформацію у базу даних MySQL. Таблиця БД повинна містити такі поля:

- Номер давача
- Дані вимірювання
- Час
- Координати

Створити веб програму для відображення результатів вимірювання у двох виглядах:

- Таблиця із результатами
- Карта із позначеними точками результатів (Результати на карті позначати кольорами)

Використати Java Thread, JSP. Бажано, але не обов'язково Spring, Spring MVC

Короткі теоретичні відомості

Мова програмування Java містить реалізацію потоків у інтерфейсі Runnable. При його наслідуванні метод run визначає команди, що складають новий потік.

JSP дозволяє виконати команди Java на сервері та відобразити інформацію клієнту. Для запуску потребує веб сервера контейнером сервлетів, такий як Tomcat або Jetty. Від часу випуску у 1999-у році JSP вже пережили свій час розквіту та стали початком для JSTL та JSF.

Код

Клас DataManager

```
import java.util.ArrayList;

/**
 * Manage all data
 * @author Foxart
 * @version 2.1
 * @since 11.14.2016
 */
public class DataManager {

	/**
	 * Delete all data
	 */
	public void deleteData() {
		MeasuredData dataInst = new MeasuredData();
		dataInst.cleanData();
	}

	/**
	 * Create data and inserts it to db and layers
	 */
	public void createData() {
		MeasuredData dataInst = new MeasuredData();
		dataInst.createData();
	}

	/**
	 * Create data and inserts it to db and layers
	 * @param number of threads/samples
	 */
	public void createData(int number) {
		MeasuredData dataInst = new MeasuredData();
		dataInst.createData(number);
	}

	/**
	 * Gets all data from db
	 * @return data
	 */
	public ArrayList<ArrayList<String>> getData() {
		MeasuredData dataInst = new MeasuredData();
		return dataInst.getAllData();
	}

	/**
	 * Gets all data from db with described id
	 * @return data
	 */
	public ArrayList<ArrayList<String>> getData(String id) {
		MeasuredData dataInst = new MeasuredData();
		return dataInst.getAllData(id);
	}
}
```

Клас DB

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
import java.util.ArrayList;
/**
* DB connection and data edit
*
* @ author FoxArt
* @ version 3.0
* @ since 11.27.2016
*/
public class DB {

	private static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
	private static final String DB_URL = "jdbc:mysql://localhost/Measurement";
	private static final String USER = "root";
	private static final String PASS = "1111";
private String sql = "";
	private ArrayList<ArrayList<String>> dataLists = new
ArrayList<ArrayList<String>>();
	private Writer write = new Writer();

	/**
	* Used to connect with db when sql updates data
	*/
	private synchronized void connection() {
		write.logData("Connect to DB");
		Connection conntn = null;
		Statement sttmnt = null;
		try {
			conntn = DriverManager.getConnection(DB_URL, USER, PASS);
			sttmnt = conntn.createStatement();
			sttmnt.executeUpdate(sql);
			sttmnt.close();
			conntn.close();
			write.logData("Connection success with sql: " + sql);
		} catch (SQLException se) {
			se.printStackTrace();
			write.logData("Connection broke with SQLException: " + se);
		} catch (Exception e) {
			e.printStackTrace();
			write.logData("Connection broke with Exception: " + e);
		} finally {
			try {
				if (sttmnt != null) sttmnt.close();
				if (conntn != null) conntn.close();
			} catch (SQLException se) {
				se.printStackTrace();
				write.logData("Connection closing broke with SQLException: "
+ se);
			}
		}
	}
```

```
	/**
	*Add SQL code to insert data to measurement table
	*/
	synchronized protected void insertData(int id, String number, String
measuredData, String latitude, String longitude) {
		sql = "INSERT INTO Measurement (id, number, measured_data, time, latitude,
longitude) VALUES ('";
		sql += String.valueOf(id) + "', '" + number + "', '" +
String.valueOf(measuredData)
				+ "', '" + latitude + "', '" + longitude + "');";
		write.logData("INSERT code formed with id " + Integer.toString(id));
		connection();
	}

	/**
	* Add SQL code to insert data to measurement table
	* whithout id
	*/
	synchronized protected void insertData(String number, String measuredData, String
latitude, String longitude) {
		sql = "INSERT INTO Measurement (number, measured_data, time, latitude,
longitude) VALUES ('";
		java.util.Date date = new java.util.Date();
		sql += number + "', '" + String.valueOf(measuredData) + "', '"
				+ String.valueOf(new Timestamp(date.getTime()))
			+ "', '" + latitude + "', '" + longitude + "');";
		write.logData("INSERT code formed with data: " + number + ", " +
measuredData + ", " + latitude + ", " + longitude);
		connection();
	}

	/**
	* Add MySQL code to get all data from table and launch rest
	* @return data from DB
	*/
	protected ArrayList<ArrayList<String>> getDataFromDB() {
		//dataLists.clear();
		sql = "SELECT * FROM Measurement;";
		write.logData("SELECT ALL code formed");
		getDataFromDBConnection();
		return dataLists;
	}

	/**
	 * Add MySQL code to get all data from table where id is defined and launch rest
	 * @param id
	 * @return data from DB
	 */
	protected ArrayList<ArrayList<String>> getDataFromDB(String id) {
		dataLists.clear();
		sql = "SELECT * FROM Measurement WHERE id=" + id + ";";
		write.logData("SELECT WHERE id=" + id + " formed");
		getDataFromDBConnection();
		return dataLists;
	}

	/**
	* Connects to DB when here is no need to update it
	*/
```

```
	private void getDataFromDBConnection() {
		Connection conntn = null;
		Statement sttmnt = null;
		try {
			conntn = DriverManager.getConnection(DB_URL, USER, PASS);
			sttmnt = conntn.createStatement();
			sttmnt.executeQuery(sql);
			fillData(sttmnt.executeQuery(sql));
			sttmnt.close();
			conntn.close();
			write.logData("Connection success with sql: " + sql);
		} catch (SQLException se) {
			se.printStackTrace();
			write.logData("Connection broke with SQLException: " + se);
		} catch (Exception e) {
			e.printStackTrace();
			write.logData("Connection broke with Exception: " + e);
		} finally {
			try {
				if (sttmnt != null) sttmnt.close();
				if (conntn != null) conntn.close();
			} catch (SQLException se) {
				se.printStackTrace();
				write.logData("Connection closing broke with SQLException: "
+ se);
			}
		}
	}

	/**
	 * Fills dataList with data from DB
	 * @param resSet
	 * @throws SQLException
	 */
	protected void fillData(ResultSet resSet) throws SQLException {
		while (resSet.next()) {
			ArrayList<String> dataList = new ArrayList<String>();
			dataList.add(Integer.toString(resSet.getInt("id")));
			dataList.add(resSet.getString("number"));
			dataList.add(resSet.getString("measured_data"));
			dataList.add(resSet.getString("time"));
			dataList.add(resSet.getString("latitude"));
			dataList.add(resSet.getString("longitude"));
			dataLists.add(dataList);
		}
		resSet.close();
	}

	/**
	 * Delete all data from table measurement
	 */
	protected void deleteAllData() {
		sql = "Delete from measurement;";
		write.logData("DELETE ALL code formed");
		connection();
	}
}
```

Клас MeasuredData

```
import java.util.ArrayList;

/**
 * Lists for all data crossing between DB and user
 * @author Foxart
 * @since 11.14.2016
 * @version 1.0
 */
public class MeasuredData {

	private ArrayList<ArrayList<String>> dataOutput;

	/**
	 * Creates data from 20 samples
	 */
	protected void createData() {
		createData(20);
	}

	/**
	 * Creates data with defined number of samples
	 * @param numberOfSamples
	 */
	protected void createData(int numberOfSamples) {
		for (int beforeIndex = 1; beforeIndex < numberOfSamples+1; beforeIndex++)
{
			@SuppressWarnings("unused")
			ThreadGenerator threadGeneratorInstance = new
ThreadGenerator(beforeIndex);
		}
	}

	/**
	 * Delete all data from maps and db
	 */
	protected void cleanData() {
		DB dbInst = new DB();
		dbInst.deleteAllData();
	}

	/**
	 * Get all data(temperature/humidity, time, latitude, longitude) with defined id
	 * @param id
	 * @return data
	 */
	protected ArrayList<ArrayList<String>> getAllData(String id) {
		fillOutputData(id);
		return dataOutput;
	}

	/**
	 * Get all data(temperature/humidity, time, latitude, longitude) with defined id
	 * @param id
	 * @return data
	 */
	protected ArrayList<ArrayList<String>> getAllData() {
		fillOutputData();
```

```
		return dataOutput;
	}

	/**
	 * Fill data for output with data from DB with described id
	 * @param id
	 */
	protected void fillOutputData(String id) {
		DB dbInst = new DB();
		dataOutput = dbInst.getDataFromDB(id);
	}

	/**
	 * Fill data for output with data from DB
	 */
	protected void fillOutputData() {
		DB dbInst = new DB();
		dataOutput = dbInst.getDataFromDB();
	}
}
```

Клас Sensor

```
import java.math.RoundingMode;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import java.util.Locale;
import java.util.Random;

/**
* Generates data from sensor
*
* @author FoxArt
* @version 2.1
* @since 11.28.2016
*/
public class Sensor {

	private Random randomInstance = new Random();

	/**
	*Generates temperature in the range from -20~+5/+5~+40
	*
	*@param cold Define cold weather and lower temperature
	*/
	protected int getTemperature (boolean cold) {
		return cold?(randomInstance.nextInt(25) - 20):(randomInstance.nextInt(35)
- 5);
	}

	/**
	*Generates delay in the range 1~10
	*/
	protected int getDelay () {
		return (randomInstance.nextInt(9) + 1);
	}

	/**
```

```
	*Generates humidity in the range 20~100
	*/
	protected int getHumidity () {
		return (randomInstance.nextInt(80) + 20);
	}

	private DecimalFormat decForm;

	/**
	*Defines decimal format to generate latitude and longitude
	*/
	private void defineDecFormat() {
		//locale need to separate decimals by dots
		DecimalFormatSymbols otherSymbols = new
DecimalFormatSymbols(Locale.ENGLISH);
		decForm = new DecimalFormat("#.######", otherSymbols);
		decForm.setRoundingMode(RoundingMode.HALF_UP);
	}

	/**
	*Generates latitude with defined decimal format and the range -85~85
	*/
	protected String getLatitude () {
		defineDecFormat();
		return decForm.format((randomInstance.nextDouble() * 170) - 85);
	}

	/**
	*Generates longitude with defined decimal format and the range -170~170
	*/
	protected String getLongitude () {
		defineDecFormat();
		return decForm.format((randomInstance.nextDouble() * 340) - 170);
	}
}
```

Клас ThreadWriter

```
/**
* Generate thread
*
* @author FoxArt
* @version 2.1
* @since 11.15.2016
*/
public class ThreadGenerator implements Runnable {

	private String threadName;

	/**
	* Defines new thread instance
	*
	* @param sensorIndex Thread name and his number
	*/
	ThreadGenerator (int sensorIndex) {
		threadName = String.valueOf(sensorIndex);
		Thread threadInstance = new Thread(this, threadName);
		threadInstance.start();
```

```
	}

	/**
	* Redefine thread commands to execute
	*/
	public void run() {
		Sensor sensorInstance = new Sensor();
		try {
			Thread.sleep(sensorInstance.getDelay());
		} catch (InterruptedException e) {
			Writer write = new Writer();
			write.logData("Thread " + threadName + " interrupted with exception:
" + e);
		}
		Writer writerInst = new Writer();
		writerInst.insertDataToDB();
	}
}
```

Висновок: Використання розділювальних систем пришвидшує роботу, якщо для цього є відповідні умови: підтримка використаних підключень(як бібліотек, так і сторонніх класів) без значного сповільнення; швидка синхронізація даних на виході розділювальної системи, якщо такий є; націленість архітектури проекту на розподілену систему та відповідні рішення при її розробці тощо. В умовах, коли переваги розділювальної системи не використовується, або ж використовується не за призначенням, коли підключені модулі лише сповільнюють роботу системи без відчутних переваг, чи занадто громіздкі для використання, - відбувається сповільнення розробки/підтримки, розростання без доцільності на те і її розвиток може стати непотрібним.