# *Génie logiciel pour la conception d'un Système d'Information*

## CSC4521

**Voie d'Approfondissement**
**Intégration et Déploiement de Systèmes d'Information**
**( VAP DSI )**

## Distributed versus Centralised

**http://jpaulgibson.synology.me/~jpaulgibson/TSP/Teaching/CSC4521/**

**.../CSC4521/CSC4521-DistribuedVersusCentralised.pdf**

**Problem** - maximum temperature in a water tank

We are going to look at 2 different implementations/simulations of a system of temperature sensors in an environment (water tank) where some parts of the tank can be very cold whilst other parts can be hot.

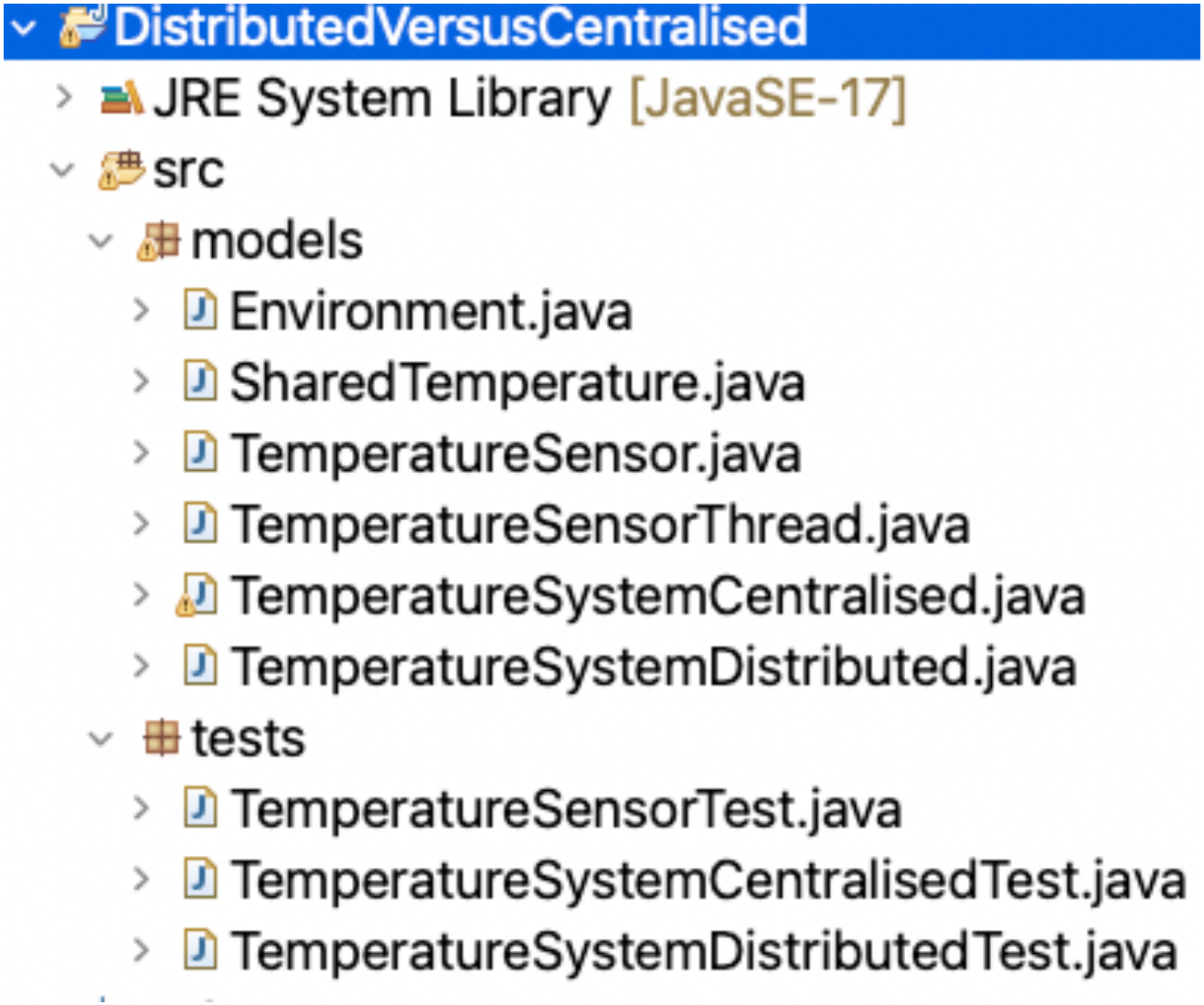
The centralised system uses a single thread of control to read all sensor values (iteratively) in order to find the maximum, when required

The distributed system models the sensors each running their own thread in order to update the maximum value.

There are delays in sensing between the sensors and the pool, and in reading sensor values.

We should see that the distributed architecture is faster.

# Download the Java archive (from the web)



```java
package models;

import java.util.Random;

public class Environment {

  public final float MINIMUM_TEMP;
  public final float MAXIMUM_TEMP;
  public final float RANGE_TEMP;

  public final Random RNG;

  public Environment (Random RNG, float minimum, float maximum) {

  this.RNG = RNG;
  MINIMUM_TEMP = minimum;
  MAXIMUM_TEMP = maximum;
  RANGE_TEMP = maximum -minimum;

  }

  public float getTemperature() {

    return RNG.nextFloat() * (MAXIMUM_TEMP - MINIMUM_TEMP) + MINIMUM_TEMP;

  }

}
```

```
package models;

public class SharedTemperature {

  float temperature;

  public SharedTemperature (float temperature) {

    this.temperature = temperature;
  }

  public float getTemperature() {
    return temperature;
  }

  public void setTemperature(float temperature) {
    this.temperature = temperature;
  }

}
```

```java
public class TemperatureSensor {

  public final int SENSOR_ID;
  public final float MINIMUM_TEMP = -50;
  public final float MAXIMUM_TEMP = 150;
  private final Environment environment;
  private final int GET_TEMPERATURE_DELAY =100;
  private final int UPDATE_TEMPERATURE_DELAY =10;
  public float temperature;

  public TemperatureSensor(Environment environment, int sensorID) {
    this.environment = environment;
    SENSOR_ID =  sensorID;
    updateTemperature(); }

  public float getTemperature() {
    try {
      Thread.sleep(GET_TEMPERATURE_DELAY);
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    return temperature;
  }

  public void updateTemperature() {
    try {
      Thread.sleep(UPDATE_TEMPERATURE_DELAY);
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    temperature  = environment.getTemperature();
  }
…
```

```java
package models;

public class TemperatureSensorThread extends Thread{

  TemperatureSensor temperatureSensor;
  SharedTemperature sharedMaximum;

  public TemperatureSensorThread(TemperatureSensor temperatureSensor,
SharedTemperature sharedMaximum) {
    this.temperatureSensor = temperatureSensor;
    this.sharedMaximum = sharedMaximum;
  }

  public void run() {
    if (temperatureSensor.getTemperature()>sharedMaximum.getTemperature())
      sharedMaximum.setTemperature(temperatureSensor.getTemperature());
    }
}
```

```
public class TemperatureSystemCentralised {

  public final int NUMBER_OF_SENSORS;
  protected final Environment environment;
  protected TemperatureSensor [] sensors;

  public TemperatureSystemCentralised(int numberOfSensors, Environment
environment) {
    NUMBER_OF_SENSORS = numberOfSensors;
    this.environment = environment;
    sensors = new TemperatureSensor [numberOfSensors];
    for (int sensorCount=0; sensorCount<numberOfSensors; sensorCount++ )
      sensors[sensorCount] = new TemperatureSensor(environment,sensorCount);}

  public float maximumSensorValue() {

    float resultMax = sensors[0].getTemperature();
    for (int sensorCount=1; sensorCount<NUMBER_OF_SENSORS; sensorCount++ )
      if (resultMax < sensors[sensorCount].getTemperature())
          resultMax = sensors[sensorCount].getTemperature();
    return resultMax;}

  public void updateSensors() {
    for (int sensorCount=1; sensorCount<NUMBER_OF_SENSORS; sensorCount++ )
      sensors[sensorCount].updateTemperature();}
…
```

```
public class TemperatureSystemDistributed extends
TemperatureSystemCentralised{

  private SharedTemperature resultMax;

  public TemperatureSystemDistributed(int numberOfSensors, Environment
environment) {
    super(numberOfSensors, environment);

    resultMax = new SharedTemperature(0);

  }

  @Override
  public float maximumSensorValue() {

    resultMax.setTemperature(sensors[0].getTemperature());

    for (int sensorCount=1; sensorCount<NUMBER_OF_SENSORS; sensorCount++ )
      new TemperatureSensorThread(sensors[sensorCount],resultMax).start();

    return resultMax.getTemperature();
  }


}
```

```
public class TemperatureSensorTest {

  public static void main (String[] args) {

    int  NUMBER_OF_UPDATES =10;

    System.out.println("**Testing Temperature Sensor**\n");

    Random RNG = new Random();
    Environment environment = new Environment(RNG,0,100);

    TemperatureSensor ts1 = new TemperatureSensor(environment,1);

    for (int updateCount = 0; updateCount <NUMBER_OF_UPDATES;
updateCount++ ) {
        System.out.println(ts1);
        ts1.updateTemperature();
    }

  }

}
```

```
**Testing Temperature Sensor**

TemperatureSensor [SENSOR_ID=1, temperature=48.148655]
TemperatureSensor [SENSOR_ID=1, temperature=71.80335]
TemperatureSensor [SENSOR_ID=1, temperature=55.482418]
TemperatureSensor [SENSOR_ID=1, temperature=17.748434]
TemperatureSensor [SENSOR_ID=1, temperature=76.76605]
TemperatureSensor [SENSOR_ID=1, temperature=4.487604]
TemperatureSensor [SENSOR_ID=1, temperature=84.48372]
TemperatureSensor [SENSOR_ID=1, temperature=25.382715]
TemperatureSensor [SENSOR_ID=1, temperature=63.025753]
TemperatureSensor [SENSOR_ID=1, temperature=49.740852]
```

```java
public class TemperatureSystemCentralisedTest {

  public static void main(String[] args) {

    long startTime = System.currentTimeMillis();

    int  NUMBER_OF_SENSORS =10;

    System.out.println("**Testing Temperature System Centralised **\n");

    Random RNG = new Random();
    Environment environment = new Environment(RNG,0,100);

    TemperatureSystemCentralised tempSysCent = new
             TemperatureSystemCentralised(NUMBER_OF_SENSORS,environment);

    tempSysCent.updateSensors();
    System.out.println(tempSysCent);

    System.out.println("The maximum sensor value is "+tempSysCent.maximumSensorValue());

    long endTime = System.currentTimeMillis();

    System.out.println("\nExecution time was " + (endTime - startTime) + "
         milliseconds");
    }
}
```

```
**Testing Temperature System Centralised **

TemperatureSystemCentralised [NUMBER_OF_SENSORS=10,
sensors=[TemperatureSensor [SENSOR_ID=0, temperature=16.64077],
TemperatureSensor [SENSOR_ID=1, temperature=7.5184226],
TemperatureSensor [SENSOR_ID=2, temperature=62.09985],
TemperatureSensor [SENSOR_ID=3, temperature=95.276794],
 TemperatureSensor [SENSOR_ID=4, temperature=61.801292],
TemperatureSensor [SENSOR_ID=5, temperature=12.464124],
TemperatureSensor [SENSOR_ID=6, temperature=96.3085],
TemperatureSensor [SENSOR_ID=7, temperature=52.032066],
TemperatureSensor [SENSOR_ID=8, temperature=22.23801],
TemperatureSensor [SENSOR_ID=9, temperature=80.38498]]]
The maximum sensor value is 96.3085

Execution time was 1592 milliseconds
```

```
public class TemperatureSystemDistributedTest {

  public static void main(String[] args) {

    long startTime = System.currentTimeMillis();

    int  NUMBER_OF_SENSORS =10;

    System.out.println("**Testing Temperature System Distributed **\n");

    Random RNG = new Random();
    Environment environment = new Environment(RNG,0,100);

    TemperatureSystemDistributed tempSysDist = new
TemperatureSystemDistributed(NUMBER_OF_SENSORS,environment);

    tempSysDist.updateSensors();
    System.out.println(tempSysDist);

    System.out.println("The maximum sensor value is "+tempSysDist.maximumSensorValue());

    long endTime = System.currentTimeMillis();

    System.out.println("\nExecution time was " + (endTime - startTime) + " milliseconds");
    }


}
```

```
**Testing Temperature System Distributed **

TemperatureSystemDistributed [NUMBER_OF_SENSORS=10,
sensors=[TemperatureSensor [SENSOR_ID=0, temperature=63.848335],
TemperatureSensor [SENSOR_ID=1, temperature=36.68211],
TemperatureSensor [SENSOR_ID=2, temperature=55.69596],
TemperatureSensor [SENSOR_ID=3, temperature=21.707075],
 TemperatureSensor [SENSOR_ID=4, temperature=69.62482],
TemperatureSensor [SENSOR_ID=5, temperature=85.9699],
 TemperatureSensor [SENSOR_ID=6, temperature=76.74236],
 TemperatureSensor [SENSOR_ID=7, temperature=66.521164],
TemperatureSensor [SENSOR_ID=8, temperature=9.0417385],
TemperatureSensor [SENSOR_ID=9, temperature=48.475792]]
The maximum sensor value is 63.848335

Execution time was 368 milliseconds
```

QUESTION: Do you see the error? What can we do to fix the code?

# TO DO (in Java)

- Fix the error in the distributed system code.
- Compare the system architecture performance with different delay values
- Analyse the system architecture performance with different numbers of sensors
- Add  delay for writing/reading to the shared temperature value, and check the impact on performance
- The client wants to calculate 2 more properties of the pool-
    A. Range of values *max-min*
    B. Average (*mean*) of values

  Implement, and test, centralised (single thread) and distributed (multi-thread) architectures for both of these.

**WITHOUT CODING**

Design a distributed solution to the problem of finding the median (middle) value.