

# Threads

Programmation avancée – Java

F5 – ISIMA 2020/2021

**ISIMA** 

**openium**  
créateur d'applications

Olivier Goutet  
o.goutet@openium.fr

17 novembre 2020

# Plan

---

Retours TP6

Threads

## Intérêt de la base ?

---

```
public static void main(String[] args) throws Exception {
    Meteo maMeteo = new Meteo();
    Bdd db = new Bdd("database.sqlite");
    String ville = args.length == 1 ? args[0] : "Clermont-Ferrand";
    try {
        Measure maMeasure = maMeteo.request(ville);
        // Measure maMeasure = maMeteo.request("uneVilleImaginaire");
        db.insertMeasure(maMeasure);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    ArrayList<Measure> measureList = db.getAllMeasures();

    for(Measure m : measureList) {
        System.out.println(m.toString());
    }
}
```

# Gestion des try/catch

---

```
1 try {
2     conn = myDB.newDB(DBName, user, passwd);
3 }
4 catch (final Exception e){
5     System.out.println(e.getClass().getCanonicalName());
6 }
7
8 try {
9     myDB.dropTable(conn, "meteo"); // On s'assure qu'elle est vide
10 }
11 catch (Exception e){
12     e.printStackTrace();
13 }
14
15 try {
16     myDB.createNewTable(conn); // Creation d'une table de tests
17 }
18 catch (final Exception e){
19     e.printStackTrace();
20 }
21
22 try {
23     myDB.loadValues(conn, wt); // Ajout d'une ligne a jour
24 }
25 catch (final Exception e){
26     e.printStackTrace();
27 }
```

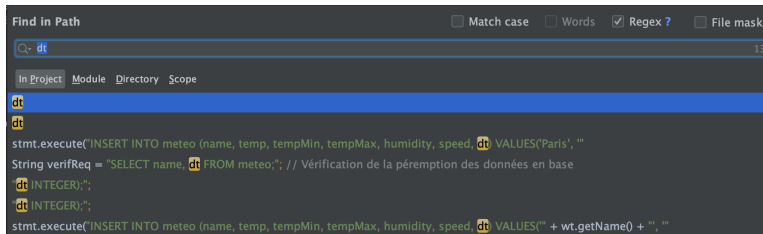
# Si on a un moteur de base, on s'en sert !

---

```
1 public static void updateDB(Connection conn, Weather wt) {
2     Calendar today = Calendar.getInstance();
3     Date date = today.getTime();
4     SimpleDateFormat formattedDate = new SimpleDateFormat("yyyyMMdd"); // Recuperation de la date
5
6     String verifReq = "SELECT name, dt FROM meteo;"; // Verification de la peremption des donn
7     try (final Statement stmt = conn.createStatement()){
8         ResultSet resSetVerif = stmt.executeQuery(verifReq);
9         String cityName = "";
10        int lastUpdate = 0;
11        while(resSetVerif.next()) {
12            cityName = resSetVerif.getString(1);
13            lastUpdate = resSetVerif.getInt(2);
14            if(!resSetVerif.isNull()) {
15                if(lastUpdate != Integer.parseInt(formattedDate.format(date).toString())) { //
16                    System.out.println("UPDATE : CLEAR OLD WEATHER");
17                    stmt.execute("DELETE FROM meteo WHERE name = '" + cityName + "';");
18                }
19            }
20        }
21    }
22    catch (final Exception e){
23        System.out.println(e.getClass().getName());
24        e.printStackTrace();
25    }
26 }
```

# Utilisation de constantes

---



The screenshot shows the 'Find in Path' search results in an IDE. The search term is 'dt'. The results are listed under the 'In Project' tab. The first two results are 'dt' in a file named 'dt'. The third result is 'dt' in a file named 'stmt.execute("INSERT INTO meteo (name, temp, tempMin, tempMax, humidity, speed, dt VALUES('Paris', "''. The fourth result is 'dt' in a file named 'String verifReq = "SELECT name, dt FROM meteo;"'. The fifth result is 'dt' in a file named 'Integer);'. The sixth result is 'dt' in a file named 'Integer);'. The seventh result is 'dt' in a file named 'stmt.execute("INSERT INTO meteo (name, temp, tempMin, tempMax, humidity, speed, dt VALUES('' + wt.getName() + ', '')

```
Find in Path ☐ Match case ☐ Words ☒ Regex ? ☐ File mask
```

Search:  13

☒ In Project ☐ Module ☐ Directory ☐ Scope

dt

dt

stmt.execute("INSERT INTO meteo (name, temp, tempMin, tempMax, humidity, speed, dt VALUES('Paris', "

String verifReq = "SELECT name, dt FROM meteo;" // Vérification de la péremption des données en base

dt INTEGER);

dt INTEGER);

stmt.execute("INSERT INTO meteo (name, temp, tempMin, tempMax, humidity, speed, dt VALUES('' + wt.getName() + ', '')

# Utilisation de constantes

---

```
1 public class Constantes {
2     public final static class COLUMNS{
3         public final static String NAME = "name";
4         public final static String TEMP = "temp";
5         public final static String TEMPMIN = "tempMin";
6         public final static String TEMPMAX = "tempMax";
7         public final static String HUMIDITY = "humidity";
8         public final static String SPEED = "speed";
9         public final static String DT = "dt";
10    }
11
12    public final static class TYPES{
13        private final static String TYPE_DOUBLE= "DOUBLE";
14        public final static String NAME_TYPE = "TEXT(128)";
15        public final static String TEMP_TYPE = TYPE_DOUBLE;
16        public final static String TEMPMIN_TYPE = TYPE_DOUBLE;
17        public final static String TEMPMAX_TYPE = TYPE_DOUBLE;
18        public final static String HUMIDITY_TYPE = TYPE_DOUBLE;
19        public final static String SPEED_TYPE = TYPE_DOUBLE;
20        public final static String DT_TYPE = "INTEGER";
21    }
22 }
23 ...
24
25 stmt.execute("DELETE FROM meteo WHERE "+Constantes.COLUMNS.NAME+" = ' " + wt.name + " '");
```

# Plan

---

Retours TP6

Threads



# Threads ?

---

- Flot de contrôle dans un programme
- Sorte de processus
  - Partage des attributs
  - Partage des variables statiques
- Synchronisation ?
  - Intégré dans java

# Threads utilisation

---

- Paralléliser
  - Utiliser tous les processeurs de la machine
  - Utiliser toutes les ressources de la machine
  - Améliorer les performances de traitement
- Gain visible ?
  - Réactivité des interfaces
  - Moins de temps d'attente
  - Plus de bugs ?

# Thread et Runnable

---

- Thread
  - Instance de `java.lang.Thread`
  - Exécute, contrôle et coordone l'exécution
- Interface Runnable
  - Ce qu'exécute le thread
  - `run()`
  - public, pas d'argument/code de retour

# Thread et Runnable exemple 1

---

```
class TraitementImage implements Runnable {
    public void run() {
        while (true) {
            // Recuperation d'une image
            // traitement de l'image
        }
    }
}

class GestionTraitement{
    public void lancerTraitement(){
        Runnable r = new TraitementImage();
        Thread monThread = new Thread(r);
        monThread.start();
    }
}
```

## Thread et Runnable exemple 2

---

```
class TraitementImage implements Runnable {
    Thread monThread;
    public TraitementImage(){
        monThread = new Thread(this);
        monThread.start();
    }

    public void run() {
        while (true){
            // Recuperation d'une image
            // traitement de l'image
        }
    }
}
```

## Thread et Runnable exemple 3

---

```
class TraitementImage extends Thread {  
    public void run() {  
        while (true){  
            // Recuperation d'une image  
            // traitement de l'image  
        }  
    }  
}
```

```
Thread ti = new TraitementImage();  
ti.start();
```

- Vous exposez les méthodes de Thread

# Contrôle des threads

---

- `sleep()`
  - attente non active
  - permet d'endormir le thread pendant une période donnée
- `wait()`, `notify()`
  - synchronisation de plusieurs threads
  - Mutex
- `interrupt()`
  - réveille un thread
  - permet de sortir un Thread de son état d'attente

# Sleep

---

```
public run(){  
    try{  
        // thread courant  
        Thread.sleep(1000);  
        // thread particulier  
        unThread.sleep(500);  
    } catch (InterruptedException e){  
        // interruption du sleep  
    }  
}
```



# Mort d'un thread

---

- Fin de la méthode `run()`
- Exception non gérée
- `stop()` (deprecated)
- Sinon il n'est jamais tué

<http://docs.oracle.com/javase/7/docs/technotes/guides/concurrency/threadPrimitiveDeprecation.html>

# Synchronisation

---

- Basé sur le concept de moniteurs
- Verrou
  - Un seul thread peut accéder en même temps à une ressource
- Il suffit d'indiquer ce qu'il faut verrouiller

# Synchronisation de méthodes

---

```
public class SynthetiseurVocal{  
    synchronized void parler( String mots){  
        //parler  
    }  
}
```

- Verrou sur l'instance
- Plusieurs méthodes peuvent être synchronized

# Synchronisation de bloc

---

```
public class SynthetiseurVocal{  
    public void parler( String mots){  
        // generer le fichier a prononcer  
        synchronized (this){  
            // acceder au peripherique  
        }  
    }  
}
```

- Verrouillage uniquement d'une portion de code
- Plus performant si peu de code à protéger

# Synchronisation de bloc

---

```
public class SynthetiseurVocal{  
    public void parler(String mots){  
        // generer le fichier a prononcer  
        synchronized (SynthetiseurVocal.class){  
            // acceder au peripherique  
        }  
    }  
}
```

- Verrouillage sur toutes les instances de la classe

# Synchronisation d'attributs

---

- La JVM accède aux type de base et aux référence de manière atomique
- double et long pas forcément bien géré sur certaines JVM
  - synchronized sur les getter/setter

## wait / notify

---

- Toujours dans un bloc synchronisé
- `wait()`
  - Relache le verrou
  - Endors le thread
  - `wait(int timeout)`
- `notify()`
  - réveille le premier thread
- `notifyAll()`

# Exemple

---

```
public class ServeurXMPP{
    synchronized void methodeServeur(){
        // traitement
        wait(); // arret, on a besoin d'un message pour continuer
        // on continue traitement
    }
    synchronized void methodeMessenger(){
        // traitement
        notifyAll(); // indique au serveur que l'on a termine
        // on continue traitement
    }
    synchronized void methodeQuiFaitAutreChose(){
        // traitement
    }
}
```