

République Algérienne Démocratique et Populaire  
Ministère de la Défense Nationale  
Ecole Militaire Polytechnique



RAPPORT DE TP  
Filière : 2 année Génie Informatique  
Option : Réseaux et Sécurité Informatique

---

# Génie Logiciel

---

*Auteurs :*

EOC. Younes HAMZA  
EOC. Ali IKHLEF

*Encadrant :*

Cne. Bilal MERABTI

Année Universitaire  
2020/2021

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Introduction . . . . .	3
1.2	Java . . . . .	3
1.3	C'est quoi Maven ? . . . . .	3
1.4	Problématique et Origines du besoin d'outils Maven . . . . .	3
<b>2</b>	<b>Patrons de conceptions utilisé</b>	<b>4</b>
2.1	Singleton . . . . .	5
2.2	Observateur . . . . .	5
<b>3</b>	<b>Manipulations</b>	<b>7</b>
3.1	Premier programme . . . . .	8
3.2	Implémentation d'une application modulaire . . . . .	9
3.2.1	Premier logiciel Multi-modules . . . . .	9
3.2.2	Une montre intelligente . . . . .	9
<b>4</b>	<b>Demonstration : Montre Intelligente</b>	<b>11</b>
4.1	L'interface graphique :générale . . . . .	12
4.2	L'interface graphique :Zoom . . . . .	12
4.3	L'interface graphique :Activer/Desactiver . . . . .	12
4.4	La journalisation : . . . . .	13
<b>5</b>	<b>Annexe</b>	<b>14</b>
5.1	les figures . . . . .	15

# Chapter 1

## Introduction

## 1.1 Introduction

Etudier le génie logiciel comme un module théorique n'est pas suffisant pour développer un logiciel concret. Il y a une grande différence entre la conception et le vrai processus de développement. Dans ce TP nous allons manipuler quelques outils qui permettent et facilitent le développement de logiciels.

Dans un premier lieu, nous allons présenter le langage *Java*, puis nous allons créer un premier programme en utilisant *Maven*.

Enfin, nous allons créer un programme multi-modules en utilisant maven.

## 1.2 Java

Java est un langage de programmation et une plate-forme informatique qui ont été créés par Sun Microsystems en 1995. Beaucoup d'applications et de sites Web ne fonctionnent pas si Java n'est pas installé et leur nombre ne cesse de croître chaque jour. Java est rapide, sécurisé et fiable. Des ordinateurs portables aux centres de données, des consoles de jeux aux superordinateurs scientifiques, des téléphones portables à Internet, la technologie Java est présente sur tous les fronts !

## 1.3 C'est quoi Maven ?

Maven est un outil permettant d'automatiser la gestion de projets Java. Il offre entre autres les fonctionnalités suivantes :

- Compilation et déploiement des applications Java (JAR, WAR)
- Gestion des librairies requises par l'application
- Exécution des tests unitaires
- Génération des documentations du projet (site web, pdf, Latex)
- Intégration dans différents IDE (Eclipse, JBuilder)

## 1.4 Problématique et Origines du besoin d'outils Maven

Le déploiement d'applications est devenu aujourd'hui un vrai casse-tête. En effet, à chaque phase du projet, les développeurs doivent gérer un environnement différent : intégration, recette, pré-production, production. Chaque environnement possède ses propres caractéristiques : adresses IP, serveurs de bases de données, etc.

Les applications sont la plupart du temps générées à l'aide de scripts Ant, de shell scripts ou même à la main. Cela revient souvent à faire des multitudes de copier/coller de scripts et à les réadapter à chaque projet.

Maven permet donc de s'affranchir de ces contraintes et d'uniformiser le déploiement des applications.

Initialement, Maven a été créée comme outil d'aide au développement d'un autre projet Jakarta : Turbine. Les développeurs sont en effet partis du constat que les différents projets Jakarta possédaient chacun leurs propres fichiers de compilation Ant. Les développeurs voulaient donc une façon standard de compiler les projets, une façon simple de publier les informations du projet et une façon de partager les jar entre projets.

## Chapter 2

### Patrons de conceptions utilisé

## 2.1 Singleton

Singleton est un patron de conception de création qui garantit que l'instance d'une classe n'existe qu'en un seul exemplaire, tout en fournissant un point d'accès global à cette instance.

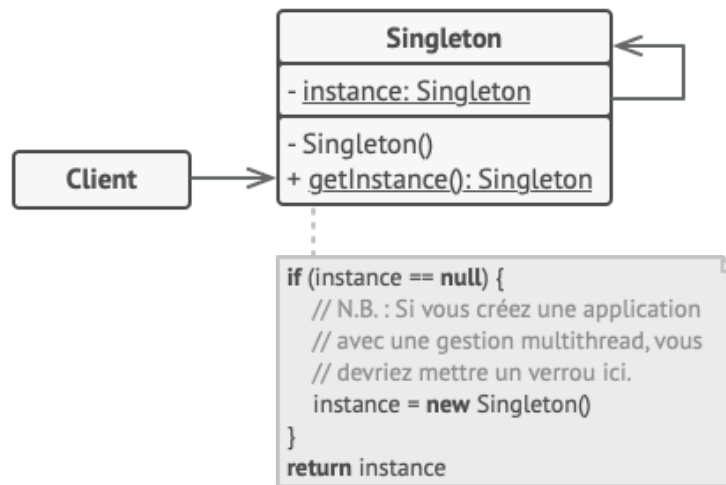


Figure 2.1: Structure d'un singleton

La classe Singleton déclare la méthode statique `getInstance` qui retourne la même instance de sa propre classe.

Le code client ne doit pas avoir de visibilité sur le constructeur du singleton. Seule la méthode `getInstance` doit permettre l'accès à l'objet du singleton.

Nous allons utiliser ce patron pour créer *Lookup*

## 2.2 Observateur

L'Observateur est un patron de conception comportemental qui permet de mettre en place un mécanisme de souscription pour envoyer des notifications à plusieurs objets, au sujet d'événements concernant les objets qu'ils observent.

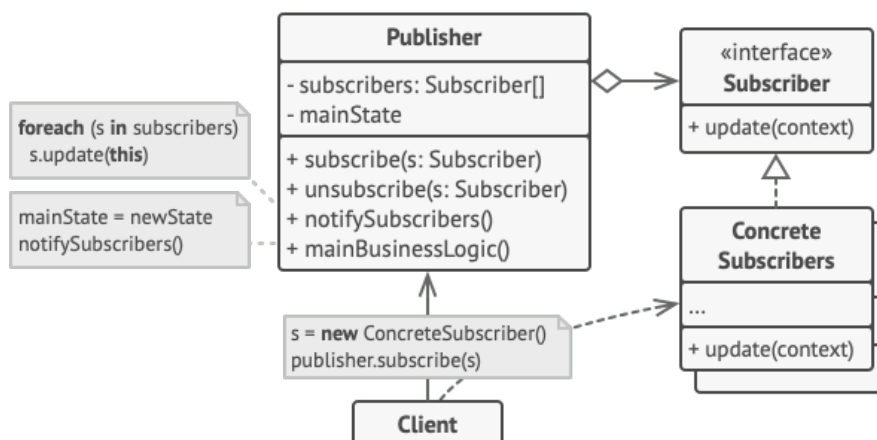


Figure 2.2: Structure d'un observateur

- Le Diffuseur (*Publisher*) envoie des événements intéressants à d'autres objets.

- Quand un nouvel événement survient, le diffuseur parcourt la liste d'inscriptions et appelle la méthode de notification déclarée dans l'interface des souscripteurs sur chaque objet souscripteur.
- L'interface Souscripteur déclare les méthodes de notification.
- Les Souscripteurs Concrets exécutent certaines actions en réponse aux notifications envoyées par le diffuseur.
- En général, les souscripteurs ont besoin de détails à propos du contexte afin d'exécuter correctement la mise à jour.
- Le Client crée des objets diffuseur et Souscripteur séparément et inscrit les souscripteurs aux mises à jour du diffuseur.

Nous allons utiliser ce patron pour gérer la *Journalisation*

# Chapter 3

## Manipulations



## 3.1 Premier programme

Pour créer un nouveau projet on exécute la commande suivante:

```
mvn archetype:generate \  
-DgroupId=org.emp.gl \  
-DartifactId=firsttp \  
-Dversion=0.0.1 \  
-Dpackage=org.emp.gl.firsttp \  
-DarchetypeArtifactId=maven-archetype-quickstart \  
-DarchetypeVersion=1.4 \  
-DinteractiveMode=false
```

Pour compiler le code :

```
mvn compile
```

Pour créer l'exécutable :

```
mvn package
```

Pour exécuter le programme :

```
java -jar target/first-tp-1.0-SNAPSHOT.jar
```

Pour partager l'archive

```
mvn install
```

Pour exécuter l'exécutable généré une configuration est indispensable car nous n'avons pas encore déterminé d'où commencer l'exécution. Cela doit être spécifié dans le fichier *pom.xml*, nous devons remplacer

```
<plugin>  
  <artifactId>maven-jar-plugin</artifactId>  
  <version>3.0.2</version>  
</plugin>
```

par

```
<plugin>  
  <artifactId>maven-jar-plugin</artifactId>  
  <version>3.0.2</version>  
  <configuration>  
    <archive>  
      <manifest>  
        <addClasspath>>true</addClasspath>  
        <classpathPrefix>lib/</classpathPrefix>  
        <mainClass>org.emp.gl.firsttp.App</mainClass>  
      </manifest>  
    </archive>  
  </configuration>  
</plugin>
```

## 3.2 Implémentation d'une application modulaire

### 3.2.1 Premier logiciel Multi-modules

Nous pouvez regrouper plusieurs modules pour créer un seul projet grâce à *Maven*, où chaque module à son propre fichier de configuration *pom.xml*.

Dans un premier temps, on crée le projet racine (*parent-project*).

```
mvn archetype:generate \  
-DgroupId=org.emp.gl \  
-DartifactId=parentdemo \  
-Dversion=0.0.1 \  
-Dpackage=org.emp.gl.parentdemo \  
-DarchetypeArtifactId=maven-archetype-quickstart \  
-DarchetypeVersion=1.4 \  
-DinteractiveMode=false  
  
cd parentdemo && rm -r src
```

Dans le fichier *pom.xml* il faut changer le packaging en pom.

```
<packaging>pom</packaging>
```

Ensuite on crée le module *childdemo*

```
mvn archetype:generate \  
-DgroupId=org.emp.gl \  
-DartifactId=childdemo \  
-Dversion=0.0.1 \  
-Dpackage=org.emp.gl.childdemo \  
-DarchetypeArtifactId=maven-archetype-quickstart \  
-DarchetypeVersion=1.4 \  
-DinteractiveMode=false
```

### 3.2.2 Une montre intelligente

Avant d'implémenter une application modulaire, il faut d'abord faire une conception correcte en réduisant au maximum la dépendance entre les modules ainsi que la dépendance de modules externes.

Dans cette partie nous avons opté pour utiliser le diagramme de classe similaire à celle vue en tp.

(figure 3.1).

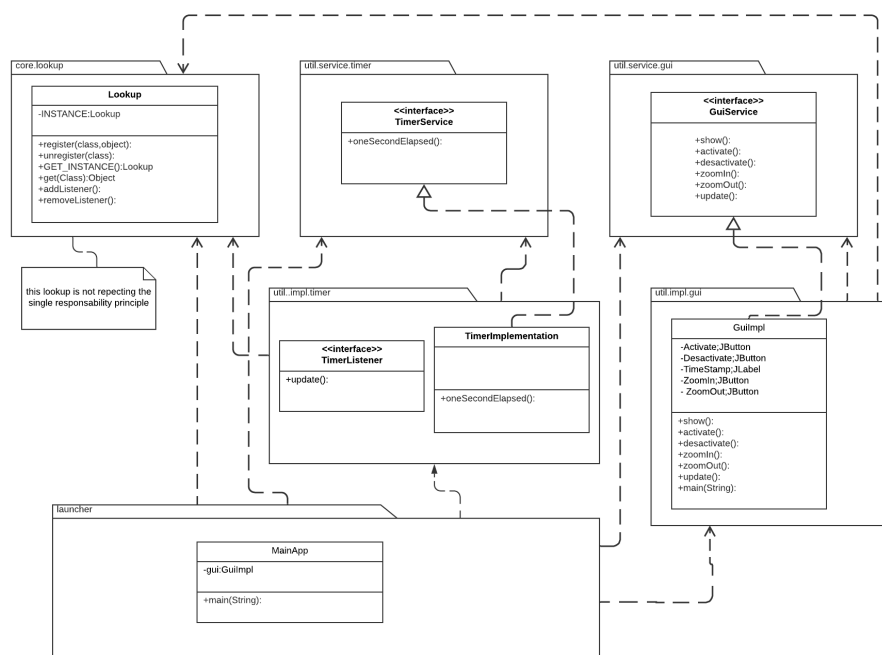


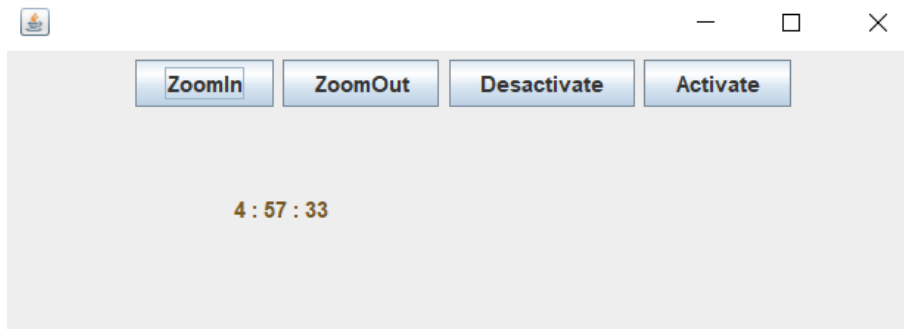
Figure 3.1: Diagramme UML représentant la conception d'une montre intelligente

## Chapter 4

### Demonstration : Montre Intelligente

## 4.1 L'interface graphique :générale

Nous avons conçu une interface graphique simple comme le montre la figure suivante :



## 4.2 L'interface graphique :Zoom

L'interface graphique est dotée d'un mécanisme de zoom.



## 4.3 L'interface graphique :Activer/Desactiver

On peut désactiver l'affichage comme il est montré ci-après:



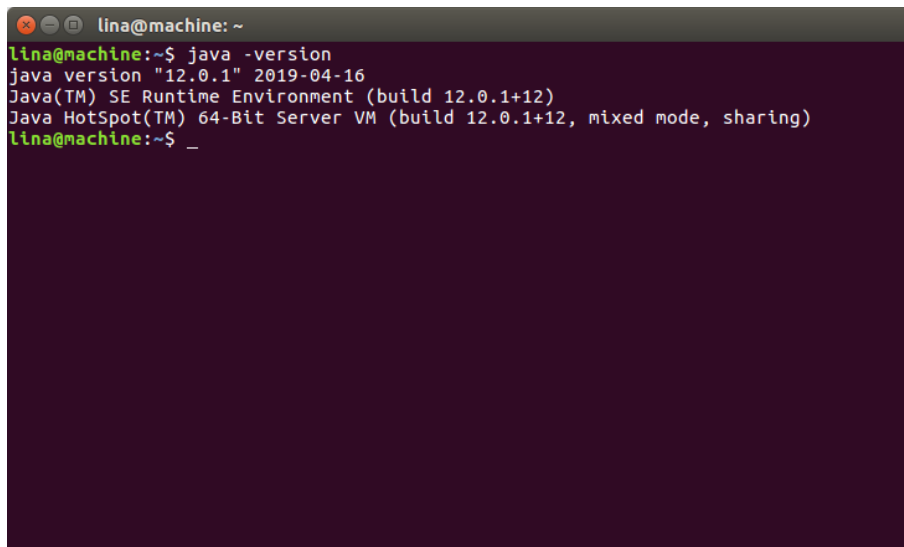
La journalisation est effectuée en utilisant le logger comme observateur sur le lookup,comme vous le voyez ci-dessous:



# Chapter 5

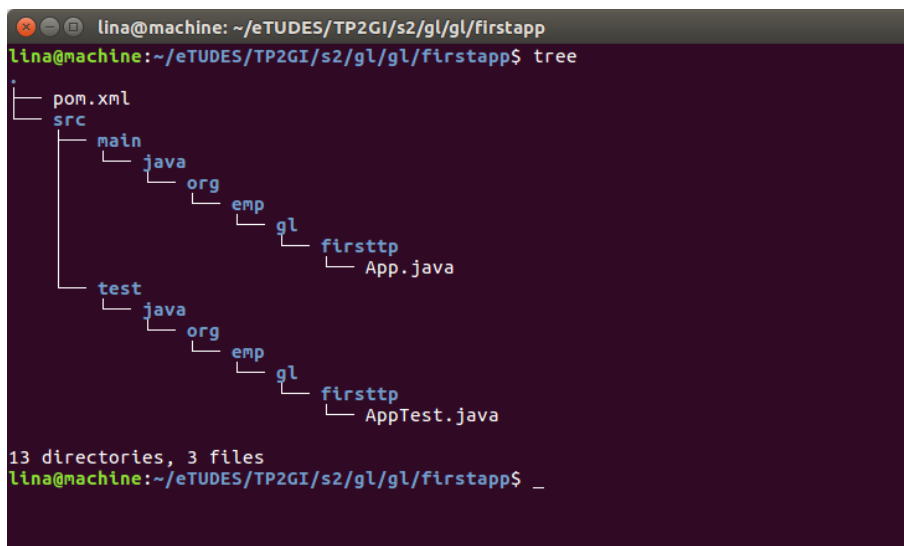
## Annexe

## 5.1 les figures



```
lina@machine: ~  
lina@machine:~$ java -version  
java version "12.0.1" 2019-04-16  
Java(TM) SE Runtime Environment (build 12.0.1+12)  
Java HotSpot(TM) 64-Bit Server VM (build 12.0.1+12, mixed mode, sharing)  
lina@machine:~$ _
```

Figure 5.1: version Java utilisée



```
lina@machine: ~/eTUDES/TP2GI/s2/gl/gl/firstapp  
lina@machine:~/eTUDES/TP2GI/s2/gl/gl/firstapp$ tree  
.  
├── pom.xml  
└── src  
    ├── main  
    │   ├── java  
    │   │   ├── org  
    │   │   │   ├── emp  
    │   │   │   │   ├── gl  
    │   │   │   │   │   ├── firsttp  
    │   │   │   │   │   │   App.java  
    │   └── test  
    │       ├── java  
    │       │   ├── org  
    │       │   │   ├── emp  
    │       │   │   │   ├── gl  
    │       │   │   │   │   ├── firsttp  
    │       │   │   │   │   │   AppTest.java  
    └── 13 directories, 3 files  
lina@machine:~/eTUDES/TP2GI/s2/gl/gl/firstapp$ _
```

Figure 5.2: structure de notre premier programme



```
lina@machine: ~/eTUDES/TP2GI/s2/gl/gl/firstapp
├── pom.xml
├── src
│   ├── main
│   │   ├── java
│   │   │   ├── org
│   │   │   │   ├── emp
│   │   │   │   │   ├── gl
│   │   │   │   │   │   ├── firsttp
│   │   │   │   │   │   │   App.java
│   │   └── test
│   │   ├── java
│   │   │   ├── org
│   │   │   │   ├── emp
│   │   │   │   │   ├── gl
│   │   │   │   │   │   ├── firsttp
│   │   │   │   │   │   │   AppTest.java
│   └── target
│       ├── classes
│       │   ├── org
│       │   │   ├── emp
│       │   │   │   ├── gl
│       │   │   │   │   ├── firsttp
│       │   │   │   │   │   App.class
│       ├── generated-sources
│       │   ├── annotations
│       ├── maven-status
│       ├── maven-compiler-plugin
│       │   ├── compile
│       │   │   ├── default-compile
│       │   │   │   ├── createdFiles.lst
│       │   │   │   └── inputFiles.lst
└── 25 directories, 6 files
lina@machine:~/eTUDES/TP2GI/s2/gl/gl/firstapp$
```

Figure 5.3: compilation du code

```
lina@machine: ~/eTUDES/TP2GI/s2/gl/gl/firstapp
/3.1.1/maven-archiver-3.1.1.jar (24 KB at 144.4 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/tukaani/xz/1.5/xz-1.5.jar
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-io/2.7.1/plexus-io-2.7.1.jar (84 KB at 253.2 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/3.4/plexus-archiver-3.4.jar (183 KB at 384.4 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/3.0.1/maven-shared-utils-3.0.1.jar (151 KB at 312.9 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/tukaani/xz/1.5/xz-1.5.jar (98 KB at 199.2 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/apache/commons/commons-compress/1.11/commons-compress-1.11.jar (416 KB at 725.4 KB/sec)
[INFO] Building jar: /home/lina/eTUDES/TP2GI/s2/gl/gl/firstapp/target/firstapp-0.0.1.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 13.619 s
[INFO] Finished at: 2021-04-11T22:15:21+01:00
[INFO] Final Memory: 17M/64M
[INFO] -----
lina@machine:~/eTUDES/TP2GI/s2/gl/gl/firstapp$
```

Figure 5.4: lors de la création de l'exécutable(.jar)

```
lina@machine: ~/eTUDES/TP2GI/s2/gl/gl/firstapp
lina@machine:~/eTUDES/TP2GI/s2/gl/gl/firstapp$ tree
.
├── pom.xml
├── src
│   ├── main
│   │   └── java
│   │       ├── org
│   │       │   ├── emp
│   │       │   │   └── gl
│   │       │   │       └── firstttp
│   │       │   │           └── App.java
│   └── test
│       ├── java
│       │   ├── org
│       │   │   ├── emp
│       │   │   │   └── gl
│       │   │   │       └── firstttp
│       │   │   │           └── AppTest.java
├── target
│   ├── classes
│   │   ├── org
│   │   │   ├── emp
│   │   │   │   └── gl
│   │   │   │       └── firstttp
│   │   │   │           └── App.class
│   ├── firstapp-0.0.1.jar
│   ├── generated-sources
│   │   └── annotations
│   └── generated-test-sources
```

Figure 5.5: après la création de l'exécutable(.jar)