

# IUT de Lens - BUT Informatique

## 2023/2024

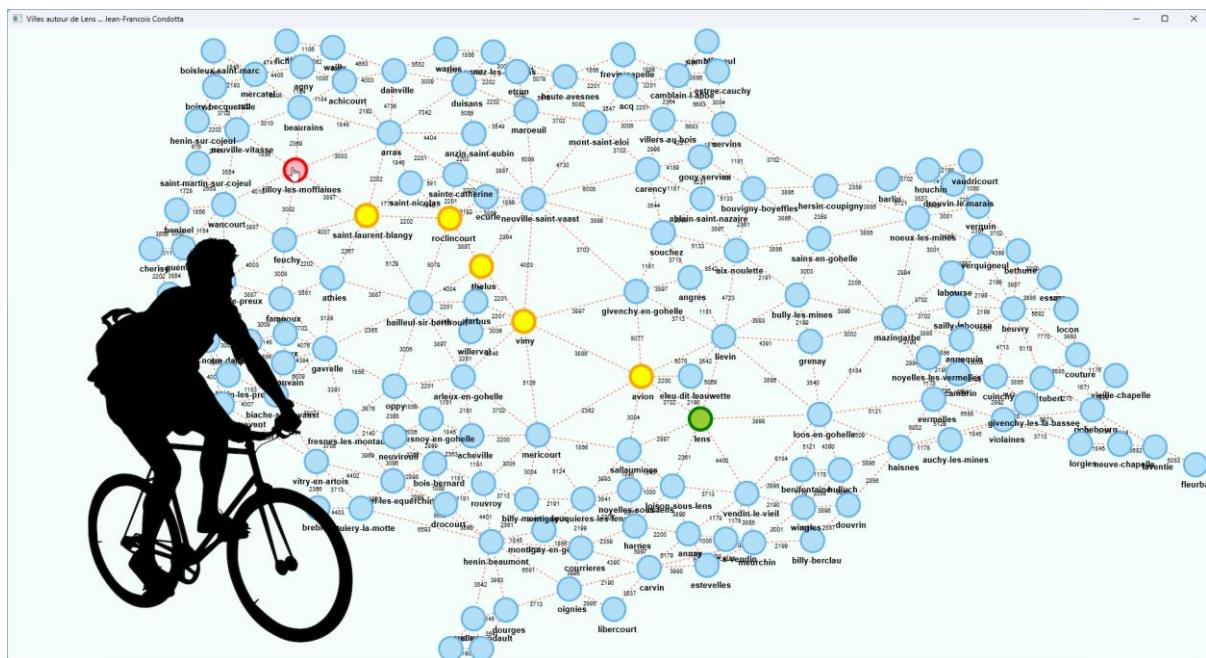
### SAÉ S2.02

## Exploration algorithmique d'un problème

# Autour des graphes : L'algorithme de Dijkstra pour le calcul d'un plus court chemin entre deux villes

R2.01 Dév. Objets & R2.07 Graphes

Mercredi 5 juin 2024 (8 h)



## Contexte et Mission

Pour donner suite à une demande incroyable de la part d'étudiants pour se rendre à l'IUT de Lens en vélo, nous souhaitons réaliser une application en Java (l'application **DijkstraVille**) permettant de calculer un plus court chemin entre deux villes (en particulier Lens et une ville set trouvant aux alentours). L'application est quasiment achevée, il ne reste plus qu'à planter une méthode de calcul de plus court chemin. L'algorithme de Dijkstra sera utilisé pour cela.

Une vidéo rappelant l'algorithme de Dijkstra et utilisant en partie l'application à réaliser est accessible à travers ce lien : <https://youtu.be/fLuBwVx-o4k>.

Pour réaliser cette SAE vous vous mettrez par groupe de 3. Des groupes de 3 ont été constitués pour travailler sur cette SAE (voir sous Moodle). **Vous devez signaler aux enseignants tout problème de groupes.** **Vous mettrez vos noms en commentaire en première ligne du fichier Sommet.java (que vous récupérerez et complèterez dans la suite).**

## Livrables

- Vous remettrez sous Moodle les fichiers sources des deux classes **Sommet.java** et **CalculateurPlusCourtChemin.java** que vous allez devoir compléter, la remise se fera **au plus tard à 12h**. Un seul étudiant par groupe remettra les livrables. Plus tôt vous remettrez ce livrable, meilleure sera votre évaluation.
- Vous remettrez également un fichier source plantUML (**dijkstra.puml**) et un fichier image (**dijkstra.png**) correspondant à un diagramme de classe UML qui aurait pu correspondre à la conception du programme. La remise se fera **au plus tard à 12h**. Un seul étudiant par groupe remettra les livrables. Plus tôt vous remettrez ce livrable, meilleure sera votre évaluation.
- A partir de 13 h, chaque groupe réalisera une présentation de 10 à 15 mn. Le planning sera effectué dans la matinée par les enseignants. Le lien Zoom vous sera également envoyé dans la matinée. Lors de cette présentation, un étudiant effectuera une démonstration, un autre commenterà le code en expliquant l'algorithme de Dijkstra et un autre commenterà le diagramme UML de l'application. Les étudiants réalisant la présentation seront choisis par les enseignants au moment de la soutenance (ainsi chaque étudiant doit connaître et savoir restituer le travail du groupe).
- A partir de 13 h, vous devrez créer une présentation de 3 diapositives au maximum et de 5 minutes au maximum présentant l'algorithme de Dijkstra. Cette vidéo (qui portera le nom **video\_dijkstra**) sera remise sous Moodle **au plus tard à 18h**. Si la vidéo est trop lourde pour être déposer sous Moodle vous pouvez simplement remettre un fichier texte (**video\_dijkstra.txt**) contenant un lien accessible vers cette vidéo.

### Remarque :

- Il est interdit d'utiliser **ChatGPT ou tout autre IA** (son utilisation entraînera un malus conséquent). Remarque : grâce aux questions posées lors de votre présentation et au code rendu, il nous sera possible de détecter l'utilisation de toute **IA**.
- Il est interdit aux différents groupes de s'entraider.

Les enseignants sont à votre disposition pour répondre à toute question.

**Bonne SAÉ à Vous !**

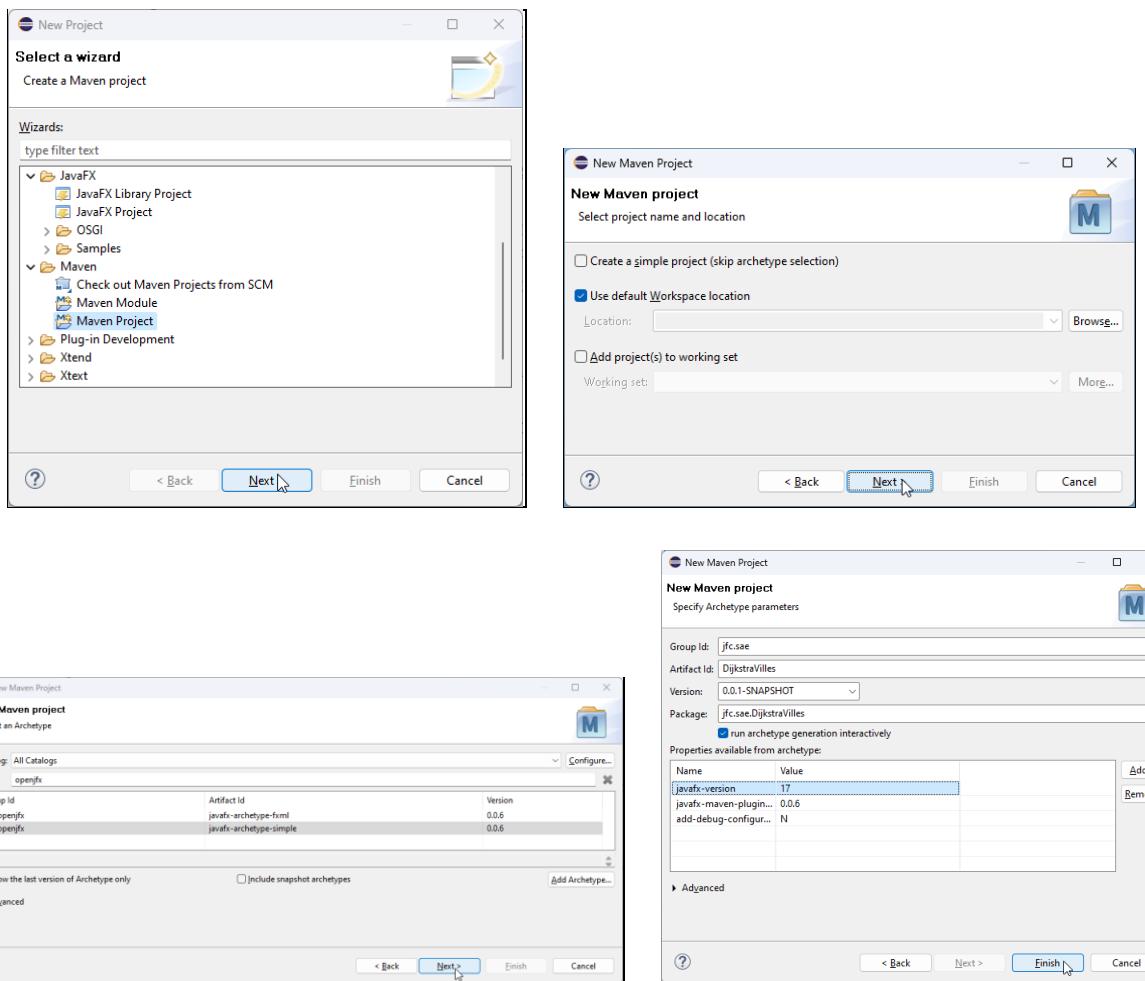
## Etape 1 – Installation de la version existante

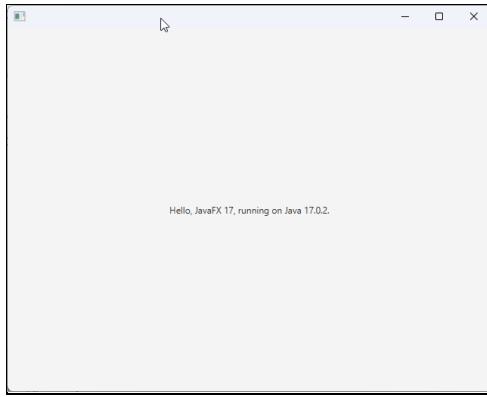
Vous travaillerez sous Linux, sur vos machines ou celles de l'IUT peu importe. À vous de vous organiser pour réaliser cette SAE.

L'application à réaliser que nous appellerons dans la suite **DijkstraVilles** utilisera une interface graphique (que vous ne développerez pas) utilisant **JavaFX**. Ainsi, vous allez devoir créer un projet Maven sous Eclipse afin d'importer la librairie JavaFX.

☞ Afin de pouvoir utiliser des dépôts distants de Maven, copiez le fichier settings.xml disponible sous Moodle (fichier contenant la définition des proxies) dans un dossier réseau. Modifiez le paramètre **Window → Preferences → Maven → User Settings** afin d'indiquer le chemin vers ce fichier.

☞ Sous Eclipse, créez un projet Maven appelé **DijkstraVilles** en suivant les captures d'écran qui suivent. Après la création vous lancerez le programme **App.java** afin de tester que JavaFX est bien installé.





La librairie **SmartGraph** est une librairie permettant de visualiser des graphes. Cette librairie est utilisée par l'application **DijkstraVilles** afin d'afficher un graphe constitué des différentes villes autour de Lens qui seront considérées.

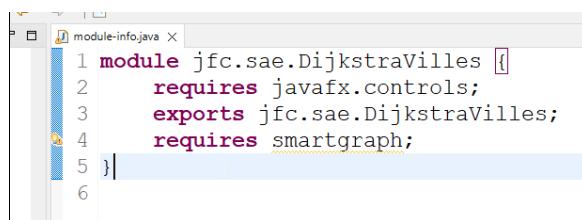
Vous allez devoir modifier le fichier Maven **pom.xml** pour pouvoir récupérer cette librairie.

☞ Sous Eclipse, ouvrez le fichier **pom.xml** du projet **DijkstraVilles** et ajoutez les lignes suivantes (**juste avant la ligne </dependencies>**) :

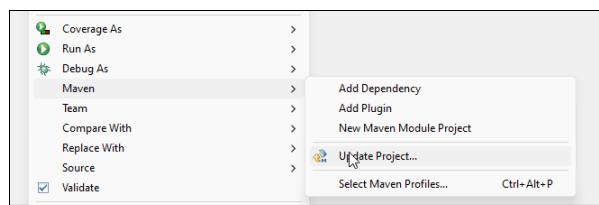
```
<dependency>
    <groupId>com.brunomnsilva</groupId>
    <artifactId>smartgraph</artifactId>
    <version>1.0.0</version>
</dependency>
```



☞ Sous Eclipse, ajoutez la ligne **requires smartgraph;** dans le fichier **module-info.java**.



☞ Sous Eclipse, mettez à jour le projet Maven **DijkstraVilles** (clic droit sur le projet **DijkstraVilles** → Maven → Update Project ...).

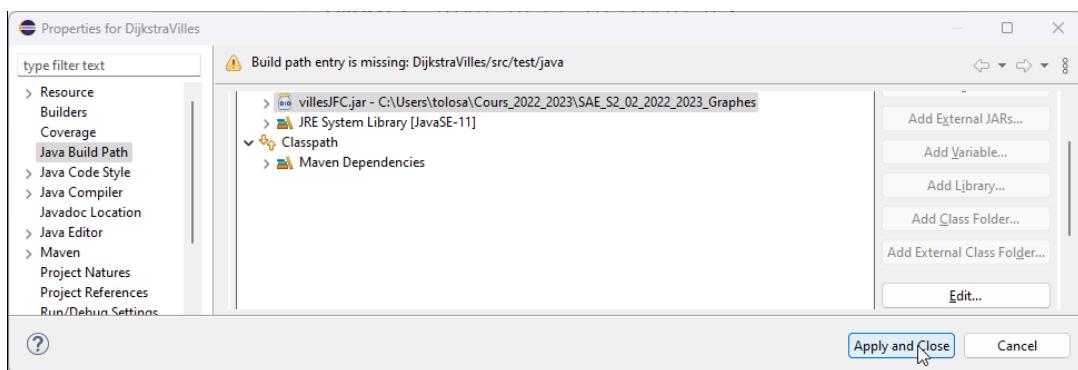
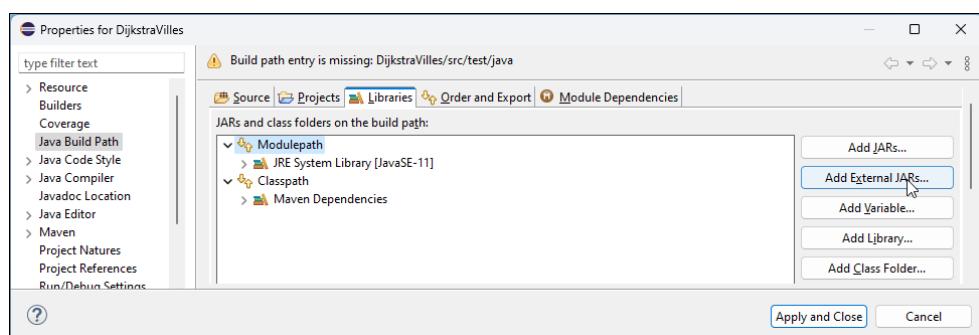
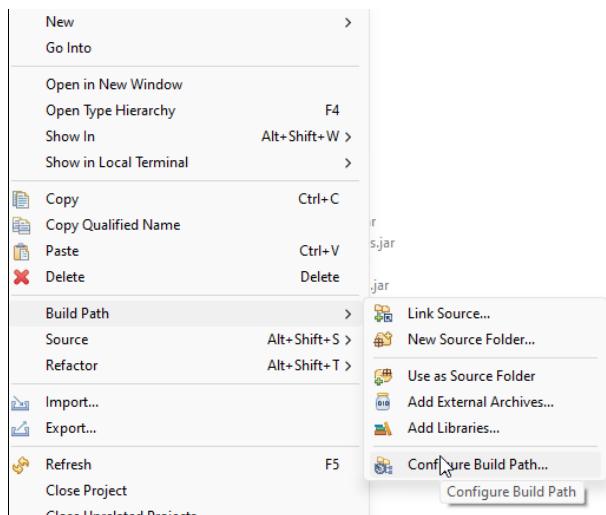


La librairie **villesJFC.jar** contient différentes classes et interfaces. En particulier, des classes permettant l'affichage de l'interface graphique de l'application **DijkstraVilles**, la classe **Ville** permettant de représenter les différentes villes et l'interface **ICalculateurPlusCourtChemin** définissant la signature de la méthode permettant le calcul d'un plus court chemin que vous allez devoir implémenter plus tard.

Nous allons maintenant incorporer la librairie externe **villesJFC.jar** à notre projet.

☞ Téléchargez le fichier **villesJFC.jar**.

☞ Incorporez l'archive externe **villesJFC.jar** au projet **DijkstraVilles**. Pour cela réalisez un clic droit sur le projet **DijkstraVilles** puis **Build Path → Configure Build Path ...** et suivez les captures d'écran suivantes.



☞ Ajoutez la ligne **requires villesJFC;** dans le fichier **module-info.java**.

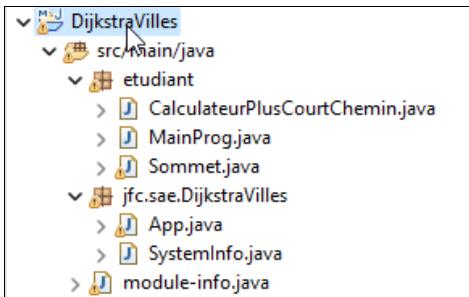
```

module-info.java ×
1 module jfc.sae.DijkstraVilles {
2     requires javafx.controls;
3     exports jfc.sae.DijkstraVilles;
4     requires smartgraph;
5     requires villesJFC;
6 }
7

```

Nous allons maintenant ajouter les différentes classes correspondant au programme principal de l'application et celles que vous allez devoir compléter dans la suite. Ces classes seront placées dans un nouveau paquetage nommé **etudiant** (placé à la racine du projet).

☞ Créez le paquetage **etudiant** et placez-y les trois fichiers **CalculateurPlusCourtChemin.java**, **MainProg.java** et **Sommet.java**.



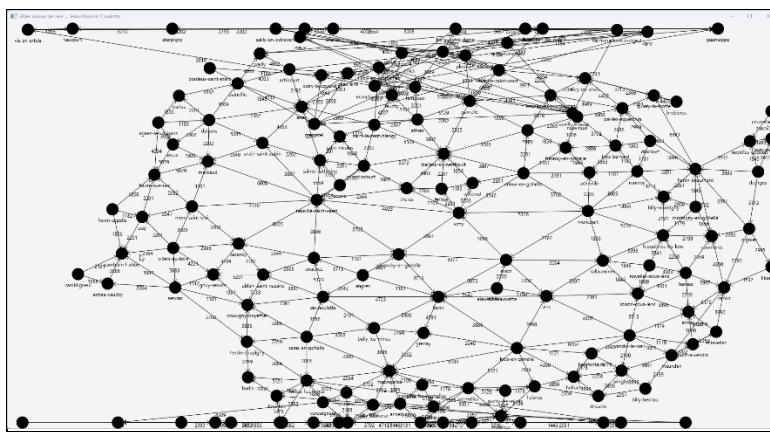
☞ Modifiez maintenant le fichier **module-info.java** en y ajoutant la ligne **exports etudiant;**:

```

module-info.java ×
1 module jfc.sae.DijkstraVilles {
2     requires javafx.controls;
3     exports jfc.sae.DijkstraVilles;
4     requires smartgraph;
5     requires villesJFC;
6     exports etudiant;
7 }

```

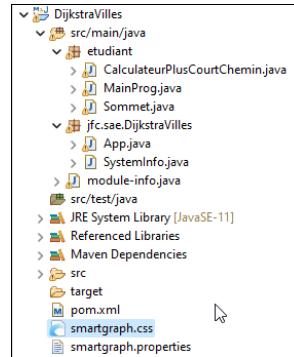
☞ Vous pouvez maintenant exécuter la classe **MainProg** ☺. Le fenêtre de l'application doit s'ouvrir avec la topologie des différentes villes autour de Lens. Deux villes voisines sont reliées par une arête sur laquelle se trouve la distance en mètres entre les deux villes.



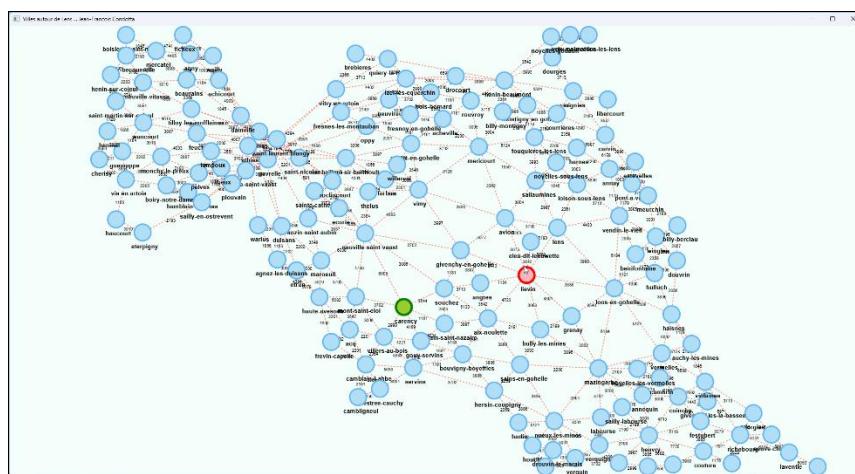
☞ Fermez l'application.

Pour terminer l'installation de la première version de notre application nous allons ajouter deux fichiers de configuration de la librairie **SmartGraph**.

☞ Copiez les deux fichiers **smartgraph.css** et **smartgraph.properties** au niveau de la racine de votre projet (au même niveau que le fichier pom.xml).



☞ Relancez l'application et testez-la. L'utilisateur doit sélectionner une ville de départ (double click sur le nœud de la ville sélectionnée) puis une ville d'arrivée (double click sur le nœud de la ville sélectionnée). Un plus court chemin entre les deux villes devra s'afficher entre les deux villes dans la fenêtre graphique et dans la fenêtre Terminal. En attendant votre implantation, le chemin affiché est juste constitué de la ville de départ et de la ville d'arrivée.



## Etape 2 – Implantation de l'algorithme de Dijkstra

- ☞ Consultez la documentation **JavaDoc** des différentes classes.
- ☞ Complétez les classes **Sommet** et **CalculateurPlusCourtChemin** afin d'implanter l'algorithme de Dijkstra.
- ☞ Réalisez le diagramme de classe correspondant à l'application.

**Bon courage à Tous !**