MONNERET Nicolas
JEANSON Ludovic

Rapport

Projet LO43 – Printemps 2007

Sujet :

Calcul de plus courts chemins à l'échelle d'une région

Resp. UV LO43: Mr Jean-Charles CREPUT
Resp. TP LO43: Mr David MEIGNAN

SOMMAIRE

MONNERET Nicolas
JEANSON Ludovic

INTRODUCTION

Dans le cadre de l'Unité de Valeur LO43 : « Bases de la programmation orientée objet », il est demandé aux élèves de réaliser un projet afin de mettre nos connaissances acquises durant le semestre en application.

Pour ce projet nous devions réaliser un système de calcul de plus courts chemins à l'échelle d'une région à l'aide d'une carte et d'un fichier xml fournis.

La première partie de ce rapport présentera le cahier des charges qui nous était donné. Puis nous expliquerons et détaillerons les spécifications ainsi que les choix de conception de notre projet. Enfin, nous finirons par un bilan de ce que réalise l'application et de ce qu'elle pourrait réaliser.

MONNERET Nicolas
JEANSON Ludovic

I – Cahier des charges :

En mode « Utilisation », le but est de calculer les trajets les plus courts entre deux points (origine-destination) saisis sur une carte de la région et de délivrer le plan de route à suivre, visuellement et sous forme texte en énumérant les routes et rues à emprunter, et en fournissant la distance du trajet.

Pour cela, il faudra impérativement utiliser les 3 éléments suivants :

- Un algorithme de plus court chemin.

- Une carte de la région autour de Belfort fournie dont l'étendue est donnée par les 2 points de coordonnées géodésiques Lambert II (897990, 2324046) et (971518, 2272510). Cette carte défini une image de fond de taille 9807 ´ 6867 qui correspond à une surface d'environ 73 km ´ 51 km, la précision est de 7.5 m by pixel. L'étendue exacte en m est donnée par les deux points Lambert II.

- Le réseau routier correspondant à cette région (region_belfort_streets.xml), donné sous forme d'un graphe sur lequel doit être appliqué l'algorithme de plus court chemins. Les coordonnées des points dans ce fichier sont données dans l'unité « pixel » de la carte de la région. Il faudra pouvoir changer ce système d'unité.

En mode « Utilisation », il faudra pouvoir faire et tenir compte des éléments suivants:

- Visualiser la carte et le réseau routier en renvoyant à l'utilisateur les informations géographiques appropriées (coordonnées des points sélectionnés, nom de rues, etc.).

- Zoomer avant/arrière, taille réelle, vue globale.

- Editer et visualiser le système d'unités et la précision. On choisira l'unité Km avec une précision au mètre près, et non pas l'unité « pixel » initiale.

- Sauvegarder le fichier de rues et routes dans ce nouveau système d'unités.

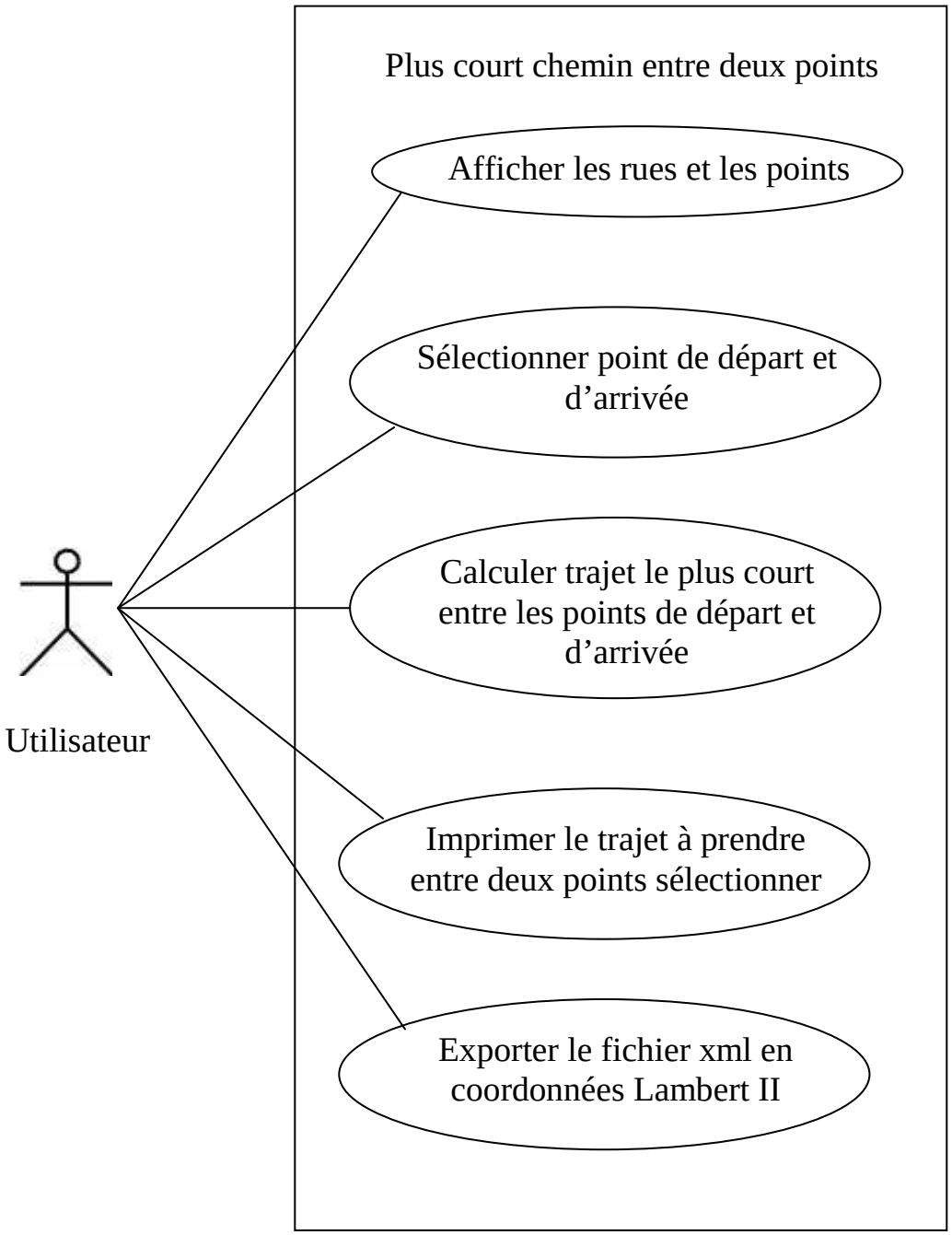
- Calculer le plus court chemin entre deux points saisis et renvoyer les informations appropriées.

Eventuellement, si le temps le permet, on pourra aller plus loin et développer d'autres fonctionnalités.

II – Spécifications et choix de conception:

Dans cette partie, nous allons analyser la manière dont nous avons implémenté notre projet.

Diagramme des cas d'utilisation:



MONNERET Nicolas
JEANSON Ludovic

Explications :

Cette application est une application mono-utilisateur. Un seul mode d'accès est disponible. Il s'agit du mode utilisateur. Celui –ci dispose de quatre fonctionnalités principales :

- Premièrement, il a la possibilité d'afficher le réseau de rues et de points (ce réseau est chargé depuis le fichier xml.

- Ensuite, il peut sélectionner deux point : un point de départ et un point d'arrivée. Lorsqu'un point est sélectionné, ses coordonnées Lambert II ainsi que la rue dans la quelle ils se trouvent sont affichées à l'écran.

- Une fois que ces deux points sont sélectionnés, l'utilisateur a la possibilité de calculer le trajet entre les deux points sélectionnés. Un détail des rues à prendre pour aller du point de départ au point d'arrivée ainsi que la distance en km séparant ces deux points sont affichés.

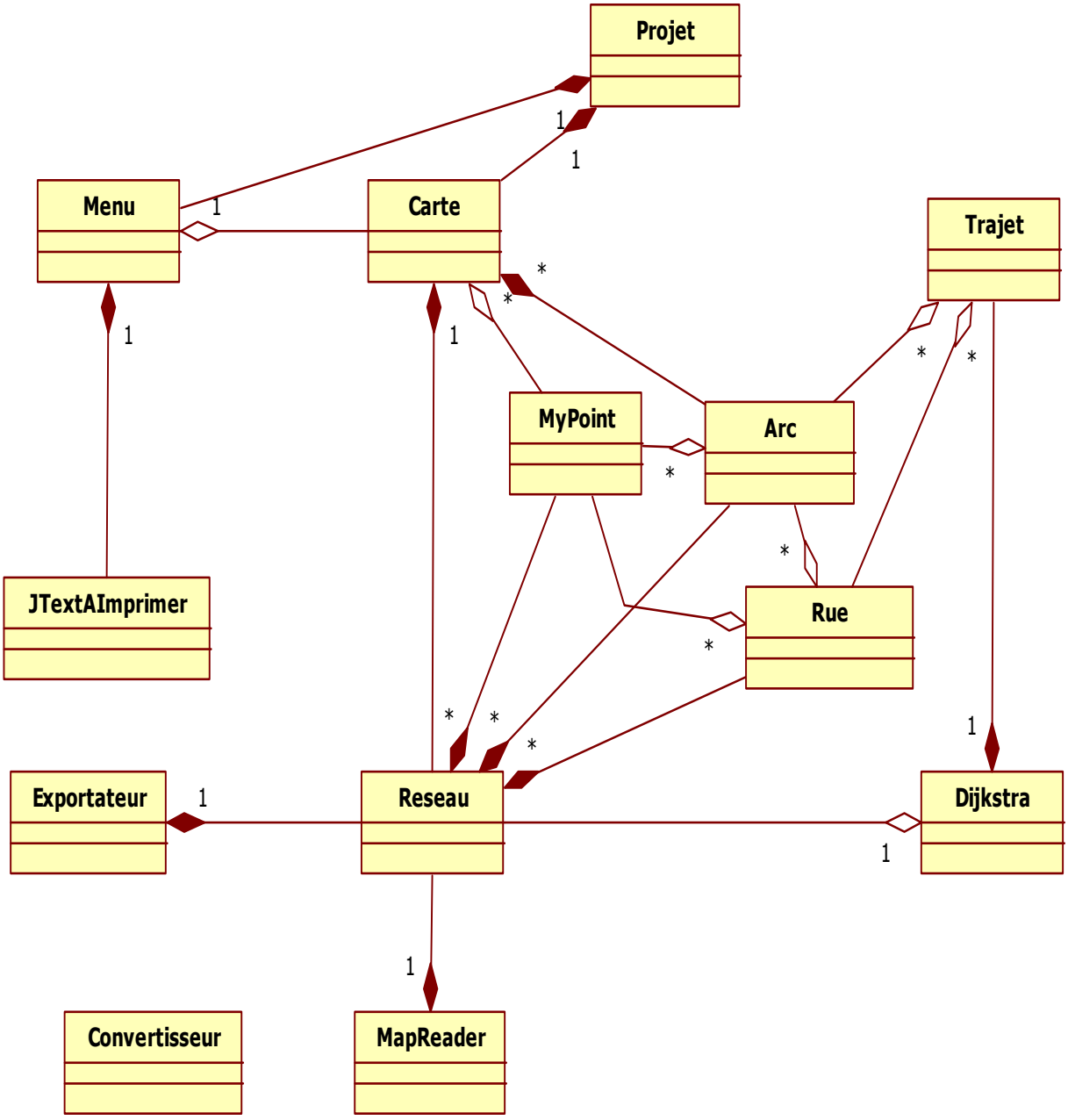
- Afin de pouvoir garder une trace du trajet calculé et de le rendre plus utile lors d'un trajet en voiture, une fonction d'impression du trajet avec rue de départ, rue d'arrivée, distance à parcourir et détail des rues à prendre est disponible.

- Enfin, la dernière fonctionnalité disponible consiste en une exportation du fichier xml original (avec des coordonnées Lambert II) en un fichier xml de même structure mais avec des coordonnées Lambert II.

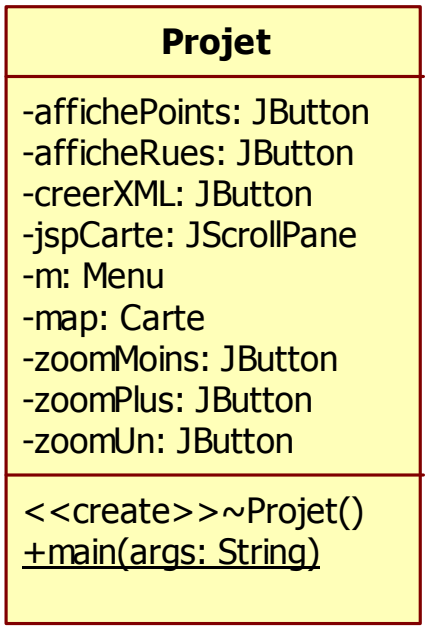
MONNERET Nicolas
JEANSON Ludovic

Diagramme de classe:

Voici la manière dont le réseau est implémenté ; il s'agit ici d'une représentation simplifiée du diagramme de classes. Chaque classe est détaillée dans les pages à suivre avec leurs méthodes et attributs.



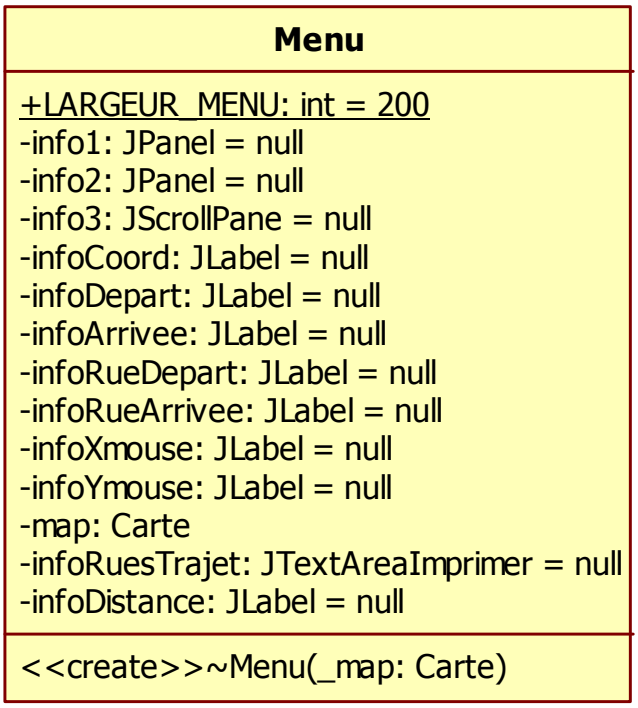
Explications du modèle et détail des classes:



Cette classe est la classe de base de notre application. C'est elle qui contient le main.
Celui-ci crée un objet de type Projet().

Si l'on regarde de plus près le constructeur de la classe Projet, on s'aperçoit tout d'abord qu'il hérite de la classe JFrame. Nous utiliserons cette classe afin de créer et gérer l'interface Homme-Machine.

Dans les attributs de cette classe, on s'aperçoit qu'elle dispose de deux objets dont nous avons nous même défini les classes. Nous aurons donc une étude approfondie de ces deux objets plus tard.
Concernant les objets permettant l'affichage, nous avons décidé d'implémenter nos boutons et menus sous forme de borderLayout. Ceci défini donc plusieurs zone (north, south, east, west, centrer). Il faudra donc penser à définir les tailles de chaque composant À afficher à l'aide de la méthode setPreferredSize() pour pouvoir ensuite utiliser la méthode pack(). La carte a été définie à l'intérieur d'un JScrollPane. Ceci permet d'afficher les barres de défilement lorsque la carte est trop grande pour être affichée en entier.
Pour chaque bouton que nous avons défini, nous avons implémenté des méthodes de traitement d'événements.
Nous avons également implémenté un traitement supplémentaire qui consiste à contrôler le mouvement de la molette de la souris. Ci une action se produit sur cette molette, le zoom est modifié.
Enfin, le bouton permettant l'export du fichier xml avec des coordonnées Lambert II crée un objet de type exportateur et appelle la fonction exec() de l'objet créé (pour plus de détail veuillez consulter le détail de la classe Exportateur plus loin dans ce rapport).



Cette classe va définir l'apparence et le contenu du menu de notre application.
Celui-ci, instancié dans la classe Projet se situera à gauche de notre application.
Cette classe hérite de la classe JPanel.
Nous diviserons notre menu en trois parties que nous ajouterons l'une en dessous de l'autre.

La première partie du menu contiendra les informations sur les coordonnées du pointeur de la souris sur la carte.
Ces coordonnées, avant d'être affichées, auront subit un traitement de conversion de coordonnées en pixel à des coordonnés Lambert II.
La deuxième partie contient elle les coordonnées et nom de rues des points sélectionnés. Elle contient également les boutons de calcul du trajet, de remis à zéro du trajet et de l'impression du trajet. Des traitements d'événements lors du clique sur un point de la carte (mise à jour des champs dans le menu) et lors du clique sur les boutons sont utilisés.
A noté que À chaque fois qu'une coordonnée doit être affichée, elle est préalablement convertie en Lambert II à nouveau à l'aide de la méthode pxToLambert() de la classe Convertisseur.
Lors du clique sur le bouton de calcul de trajet, un objet de la classe Dijkstra est créé et la méthode execute() de la classe Dijkstra (qui prend comme paramètres les points de départ et d'arrivée) est appelée (plus de détail sur la classe Dijkstra dans le pages qui suivent). Une fois le trajet entre les deux points définis obtenus, la distance entre ces deux points s'affiche et le détail des rues s'affiche également dans la troisième partie du menu.
Cette partie contient une instance de la classe JTextAreaImprimer qui n'est qu'une extension de la classe JTextArea avec une méthode supplémentaire ImprimerJTextArea() qui permet d'imprimer sur l'imprimante choisie le contenu textuel de cet objet.
Un dernier bouton appelé resetTrajet, présent dans la deuxième partie du menu permet de remettre à zéro les points de départ, d'arrivée, la distance du trajet et le JTextAreaImprimer.

Carte
-affDepart: boolean = false -affArrivee: boolean = false -affPoints: boolean = false -affArcs: boolean = false -affTrajet: boolean = false -arrivee: ImageIcon = null -depart: ImageIcon = null -fond: ImageIcon = null -hauteur: int -largeur: int -ptArrivee: MyPoint = null -ptDepart: MyPoint = null -res: Reseau -trajet: ArrayList<Arc> = null -zoom: double +xMouse: int +yMouse: int
<<create>>~Carte() +afficherArc() +afficherDepart() +afficherArrivee() +afficherPoints() +AfficherTrajet(b: boolean) -dessinerArcs(graph: Graphics2D) -dessinerPoints(graph: Graphics2D) -dessinerTrajet(graph: Graphics2D) +getAffDepart(): boolean +getAffArrivee(): boolean +getDepart(): MyPoint +getArrivee(): MyPoint +getEchelle(): double +getReseau(): Reseau +getZoom(): double +paint(g: Graphics) +setTrajet(_trajet: ArrayList<Arc>) +updateZoom(_zoom: double)

Après avoir défini le menu situé en haut dans la classe projet, le menu situé à gauche qui permet l'affichage des informations relatives au curseur de la souris et aux points de départ et d'arrivée sélectionnés par l'utilisateur dans la classe Menu, il ne reste plus qu'à définir la partie contenant la carte.

Cette classe qui permet d'afficher la carte à l'écran mais également qui effectue de nombreux traitements hérite de la classe JPanel.

Les points à soulignés dans l'analyse de cette classe sont la présence des deux points sélectionnés par l'utilisateur (point de départ et d'arrivée) ainsi qu'une instance de la classe Reseau (classe que nous détaillerons plus loin) ainsi qu'une liste d'Arc appelée trajet.

Le constructeur va charger un Reseau à l'aide d'un fichier xml. Ceci est réalisé par la méthode loadxml() de la classe MapReader.

Pour ce qui est des autres points à souligner, votre attention devrait se porter sur le surchargement de la méthode paint(). En effet, la classe JPanel défini déjà cette méthode mais pas suffisamment pour nous permettre d'afficher notre carte sur l'écran. Il faut donc surcharger cette méthode et appeler la méthode repaint() de la classe JPanel pour rafraîchir l'affichage dès qu'une modification apparaît sur la carte.

Trois méthodes quasi identiques vont nous permettre de dessiner les Arcs, les Points ainsi que le trajet sur la carte calculé dans la classe Dijkstra (dessinerArcs(), dessinerPoints(), dessinerTrajet()).

Reseau
-cheminCarte: String = null -echelle: double = 7,5 -nom: String -listPts: ArrayList<MyPoint> -listArcs: ArrayList<Arc> -listRues: ArrayList<Rue>
<<create>>~Reseau() <<create>>+Reseau(_nom: String) +addArc(a: Arc) +addPoint(p: MyPoint) +addRue(r: Rue) +getCheminCarte(): String +getDestinations(mp: MyPoint): List +getDistance(start: MyPoint, end: MyPoint): double +getEchelle(): double +getListArcs(): ArrayList<Arc> +getListRues(): ArrayList<Rue> +getListPoints(): ArrayList<MyPoint> +getNom(): String +getPointByNum(numPt: int): MyPoint +getPointPlusProche(_x: int, _y: int): MyPoint +getRueByPoint(pt: MyPoint): Rue -isArc(pt1: MyPoint, pt2: MyPoint): boolean +setCheminCarte(carte: String)

Cette classe a été citée de nombreuses fois précédemment ; c'est pourquoi il est temps de la détailler.

La première remarque à faire est qu'une instance de la classe Reseau n'est en réalité q'une classe composée des listes d'Arc, de MyPoint et de Rue qui constituent notre réseau routier.

Ces listes sont initialisées dans la classe MapReader.

Concernant les methodes de la classe Reseau, ce sont toutes les méthodes relatives à l'accès ou à la modification de ces listes.

Un point à noté pour la méthode getPointPlusProche() est qu'elle permet de récupérer, le MyPoint le plus proche du pointeur de la souris lorsque l'utilisateur clique sur la carte pour définir ses points de départ et d'arrivée. Afin de faciliter le clique de l'utilisateur, elle retourne le MyPoint le plus proche à plus ou moins dix pixels du clique.

Enfin, la dernière méthode à noter est la méthode getDestination() qui sera utilisée par la classe Dijkstra lors du calcul du plus court chemin entre deux points. Cette méthode retourne la liste des MyPoint reliés à un MyPoint passé en paramètre par un Arc.

MapReader
-chemin: String = null -res: Reseau = null
<<create>>~MapReader() <<create>>+MapReader(_chemin: String) +loadXML(): Reseau

Cette classe, comme cela a été dit précédemment, est la classe qui va permettre de lire le fichier xml et ainsi créer un Reseau à partir de ce fichier.
Concernant l'API utilisée pour traiter le fichier xml, nous avons utilisé l'API DOM (Document Object Mode). L'utilisation de cette API ne vas pas (comme le ferait l'API SAX) seulement parcourir le document XML, mais va, en plus, en fabriquer une représentation en mémoire. Cette représentation est un arbre, que nous pourrons ensuite facilement parcourir. Cet arbre représente l'organisation et le contenu du document XML. En contrepartie de sa facilité d'utilisation, DOM sera plus gourmand en ressources et en temps.
Nous avons décidé, afin de faciliter le traitement du calcul du plus court chemin entre deux points de définir une liste de MyPoint, une liste de Rue (les deux étant déjà définis totalement dans le fichier XML) mais également une liste d'Arc.
En effet, en n'utilisant qu'une liste de MyPoint et de Rue, l'algorithme de plus court chemin aurait mal fonctionné.
Prenons cinq points. Le trajet proposé par Dijkstra est dessiné en vert La figure 1 montre ce que l'algorithme de plus court chemin proposerait à la fin de son traitement si on utilisait que des listes de MyPoint et de Rue. La figure 2 quant à elle montre ce que l'algorithme proposera avec en lus une liste d'Arc.

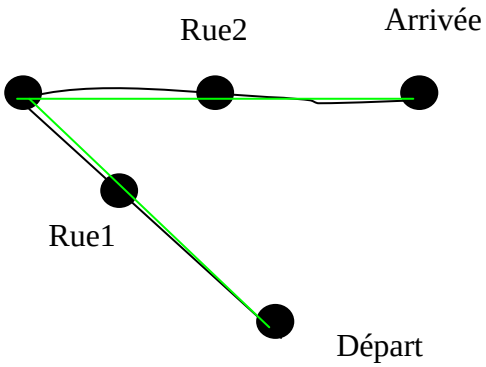


Figure 1

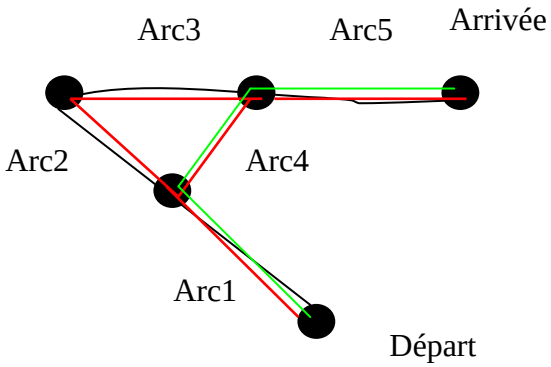
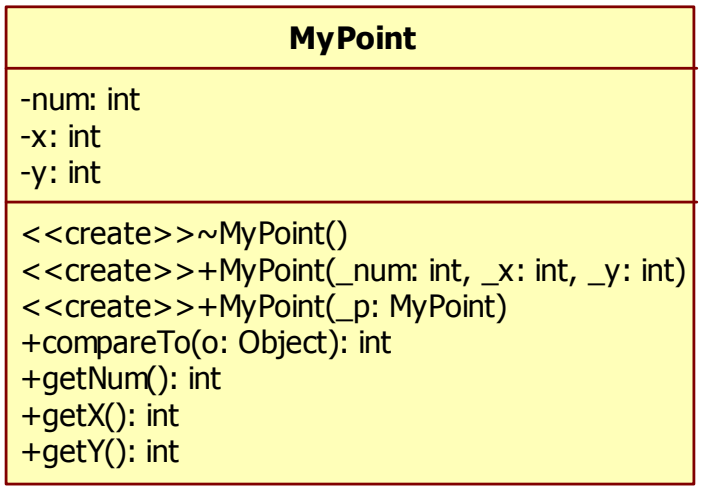
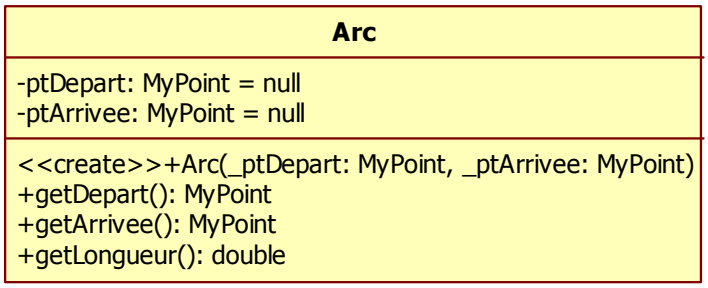


Figure 2



La classe MyPoint permet de définir un point sur la carte avec ces coordonnées et son numéro pour pouvoir le reconnaître lors de tous traitements sur ces MyPoint.
Cette classe dispose des méthodes classique d'accès aux attributs mais aussi de la méthode compareTo() qui compare deux points.



La raison de l'utilisation de cette classe est expliquée plus haut.
Comme vous pouvez le voir dans ses attributs, un Arc est composé de deux points (ses deux extrémités).
Cette classe dispose également des méthodes classique d'accès aux attributs mais également de la méthode getLongueur() qui retourne la longueur d'un Arc qui sera utilisée lors du calcul de la distance d'un trajet.

Rue
-nom: String -sens: int -listPts: ArrayList<MyPoint> -listArcs: ArrayList<Arc>
<<create>>+Rue(_r: Rue) <<create>>+Rue(_nom: String, _sens: int, _listPts: ArrayList<MyPoint>) <<create>>+Rue(_nom: String, _sens: int, _listPts: ArrayList<MyPoint>, _listArcs: ArrayList<Arc>) +addArc(a: Arc) +containsArc(a: Arc): boolean +getListArcs(): ArrayList<Arc> +getListPts(): ArrayList<MyPoint> +getNom(): String +getSens(): int

La dernière classe permettant de décrire notre Reseau est celle définissant une rue. Une rue dispose donc des données présentent dans le fichier xml (nom, sens, liste de points sous forme d'une ArrayList<MyPoint>) mais également d'une liste d'Arc, ceux –ci ayant été définis plus haut. De même que pour les deux classes précédentes, la classe Rue dispose des méthodes d'accès aux attributs mais également des méthodes d'ajout d'un Arc à un objet de type Rue mais également de la méthode containsArc() qui permet de savoir si une instance d'Arc est contenue dans un objet Rue.

Dijkstra
+INFINITE_DISTANCE: double = Double.MAX_VALUE -shortestDistanceComparator: Comparator -res: Reseau -unsettledNodes: SortedSet -settledNodes: Set -shortestDistances: Map -predecessors: Map -trajet: Trajet = null
<<create>>+Dijkstra(_res: Reseau) -init(start: MyPoint) +execute(start: MyPoint, destination: MyPoint) -extractMin(): MyPoint -relaxNeighbors(u: MyPoint) -markSettled(u: MyPoint) -isSettled(v: MyPoint): boolean +getShortestDistance(mp: MyPoint): double -setShortestDistance(mp: MyPoint, distance: double) +getPredecessor(mp: MyPoint): MyPoint -setPredecessor(a: MyPoint, b: MyPoint) -makeTrajet(start: MyPoint, destination: MyPoint) +getTrajet(): Trajet

Cettte classe implémente l'algorithme de Dijkstra. Nous l'avons créé à l'aide de l'explication fournie sur le site : http://renaud.waldura.com/doc/java/dijkstra/

Vous trouverez sur la page suivante une explication de l'implémentation de cet algorithme.

Explication:

Pour chaque noeud, de voisin en voisin en partant du départ, on parcours le Reseau jusqu'a l'arrivée en gardant toujours les noeuds ayant la distance la plus faible avec le départ.

Données:

d: liste qui contient pour chaque noeud du Reseau la plus courte distance entre le départ et lui

P: liste qui contient le noeud prédécesseur de chaque noeud dans le plus court chemin (utile à la fin pour remonter le trajet)

S: liste de noeuds qui ont déjà été traités

Q: liste de noeuds qui n'ont pas été traités

Dp: noeud de départ

Ar: noeud d'arrivée

Initialisations:

d: liste de distances infinies pour chaque noeud (Valeur max d'un int)

P: liste vide

S: liste vide

Q: liste vide

ajouter Dp à Q

d(Dp) = 0

Algorithme principal:

```
TantQue (Q n'est pas vide)
{
        u = extraire le noeud ayant d minimum de la liste Q (trouver, supprimer de Q, renvoyer)

        si u = Ar on sort de la boucle (terminé)
        sinon{
                ajouter u à S
                Traiter_noeuds_voisins (u)
        }
}
FinTantQue
```

```
Traiter_noeuds_voisins(u)
{
	pour chaque noeud v voisin de u qui n'est pas dans S (donc non traité)
	{
		si d(v) > d(u) + [u,v] ([u,v] distance entre les 2 noeuds)
		{
			d(v) = d(u) + [u,v] (on a trouvé une nouvelle plus courte distance jusqu'à v)
			P(v) = u (on met a jour la liste des prédécesseurs)
			ajouter v à Q (on ajoute v à la liste des non traités pour pouvoir le traiter à l'étape suivante)
		}
	}
}
```

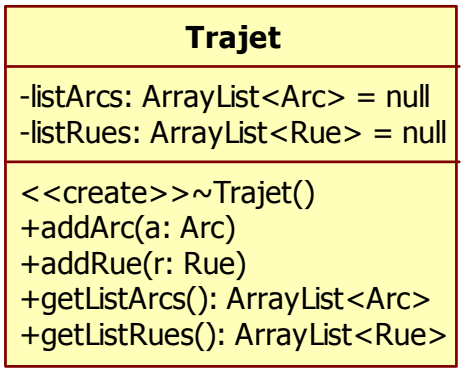
Choix d'implémentation:

d: liste HashMap: permet de retrouver un enregistrement à partir d'une clé unique

P: liste HashMap: idem

S: liste HashSet: performances constantes avec la methode contains() utilisée ici

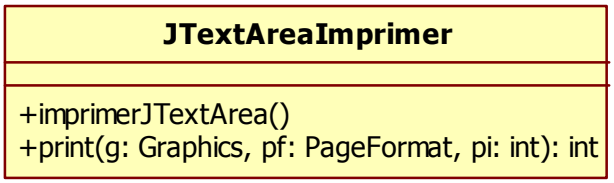
Q: liste treeSet associé à au comparateur de noeud: permet de garder une liste triée automatiquement



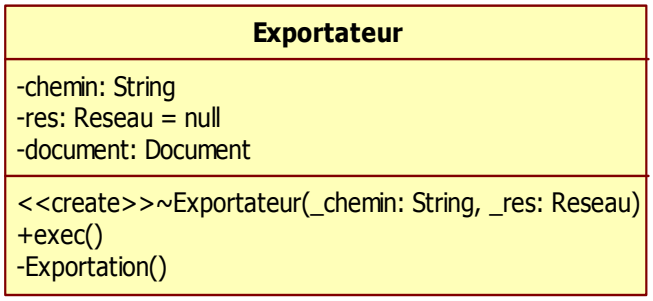
Cette classe est utilisée lors du calcul du trajet le plus cour entre deux points par la classe Dijkstra.

Ce trajet, qui va correspondre en une liste d'Arc et une liste de Rue sera ensuite utilisé lors de l'affichage du trajet sur la carte.

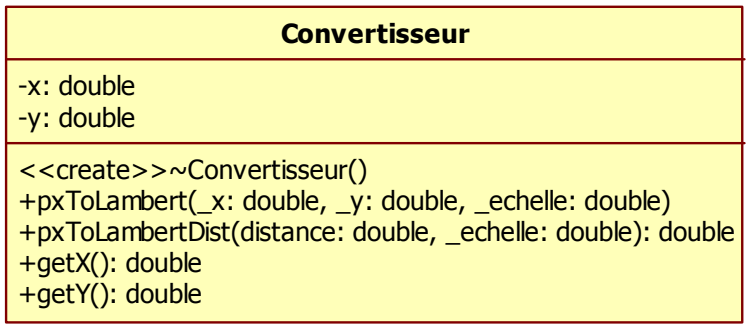
En effet, les deux méthodes getListArcs() et getListRues() seront utilisées par la classe Carte.



Comme expliqué lors du début du détail des classes, cette classe va permettre d'imprimer le contenu d'un JtextArea redéfini en JTextAreaImprimer.



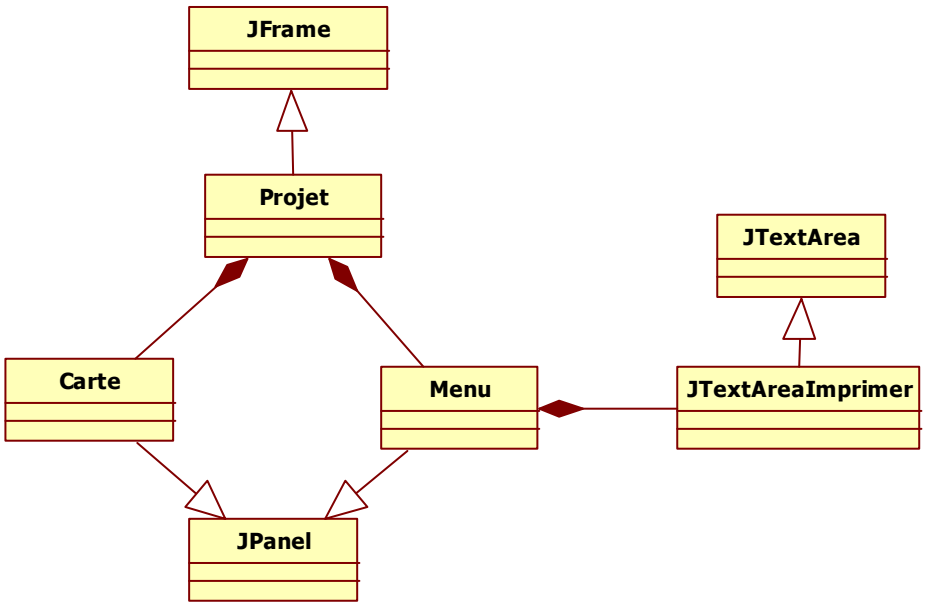
Comme nous cela était inscrit dans le cahier des charges, nous avons ajouté une classe permettant d'exporter le fichier xml original qui comportait les coordonnées des points en pixel en un autre fichier xml avec les coordonnées Lambert II.
Nous créons donc le fichier xml à l'aide de l'API DOM.



Pou finir, nous avons également créé un classe nommé Convertisseur qui permet de traduire les coordonnées d'un point de pixel en coordonnées Lambert II ou une distance exprimée en pixel en distance avec les coordonnées Lambert II.

MONNERET Nicolas
JEANSON Ludovic

Diagramme concernant l'interface graphique (UML):



Explications du modèle et détail des classes:

La fenêtre principale « Projet » (héritée de JFrame) contient une barre de menu (hérité de JPanel) et une zone d'affichage de la carte (héritée également de JPanel).
La classe Menu est composée d'une zone de texte à imprimer qui hérite de JTextArea.

A noter que la relation entre la partie graphique et la partie interne vue précédemment se fait par le biais de « Projet » qui instancie un objet de type « Carte » et un objet de type « Menu ».

Conclusion

En conclusion, toutes les fonctionnalités requises dans le cahier des charges ont été réalisées.

Cependant il reste encore de nombreuses optimisations à réaliser et notamment en ce qui concerne l'édition du réseau afin de pouvoir étendre la surface couverte par notre application.
De plus, des améliorations peuvent être réalisées dans le temps de chargement de notre application. Ce temps pourrait certainement être diminué à l'aide de l'utilisation de Thread.

Ce projet nous a permis d'améliorer nos connaissance en Java, en UML et en Génie Logiciel (amélioration interface, contrôle de l'utilisateur). Il a donc été pour nous très enrichissant de réaliser ce projet.