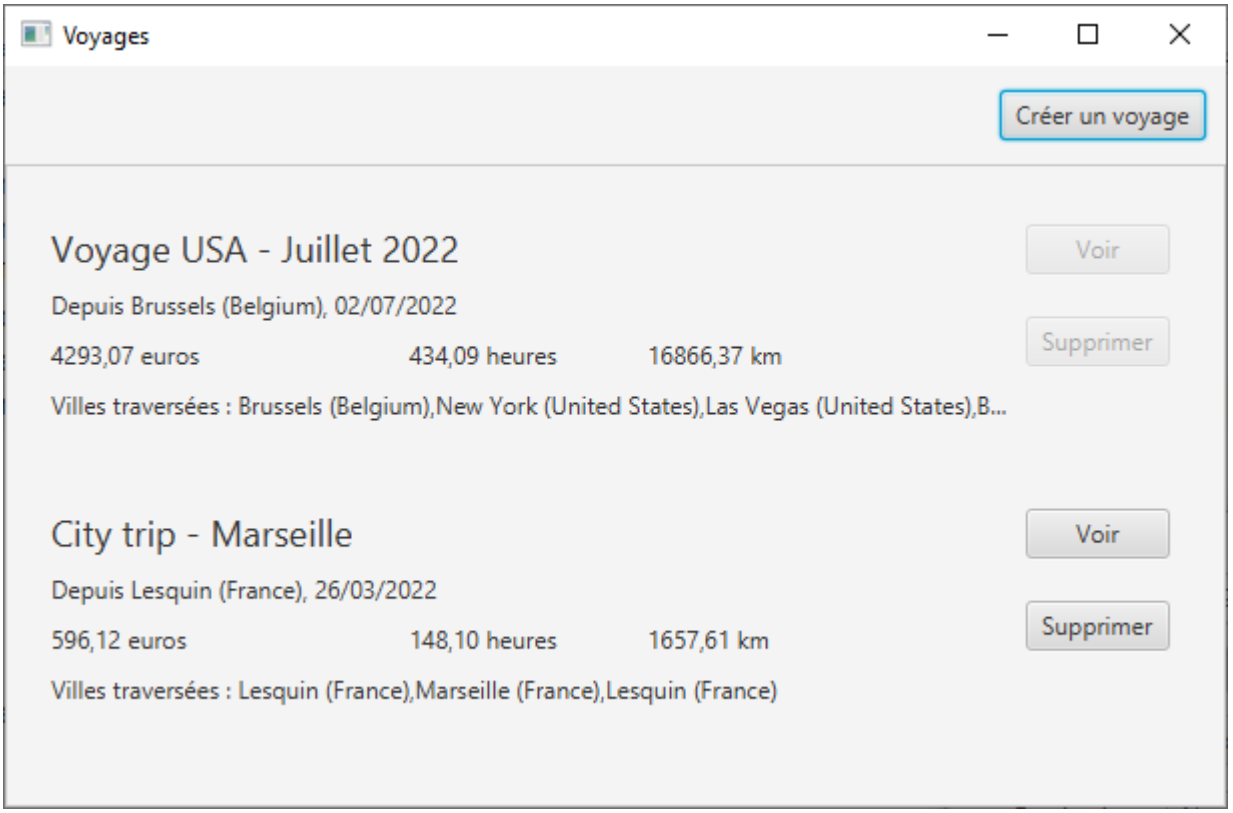
UE21 – Informatique appliquée Projet Java	HelHa Pluquet Frédéric 2021-2022

On vous demande de produire une application permettant de gérer facilement les voyages d'une agence de voyages. Les employés peuvent encoder les données sur le trajet des voyages (avions + hôtels).

L'application est composée de 2 programmes. Le premier est l'interface principale de l'employé lui permettant de gérer les voyages. Ce programme sera exécuté plusieurs fois en parallèle (une instance de programme par employé) et ces différentes instances de ce programme fonctionnent alors en réseau (via des sockets) pour qu'elles puissent être installées sur des ordinateurs différents. Le deuxième programme sera le programme serveur qui gérera toutes les synchronisations entre les différentes instances du premier programme et il n'aura pas d'interface graphique.

Vous trouverez ici une vidéo montrant leur utilisation : https://youtu.be/zyO38-veplQ

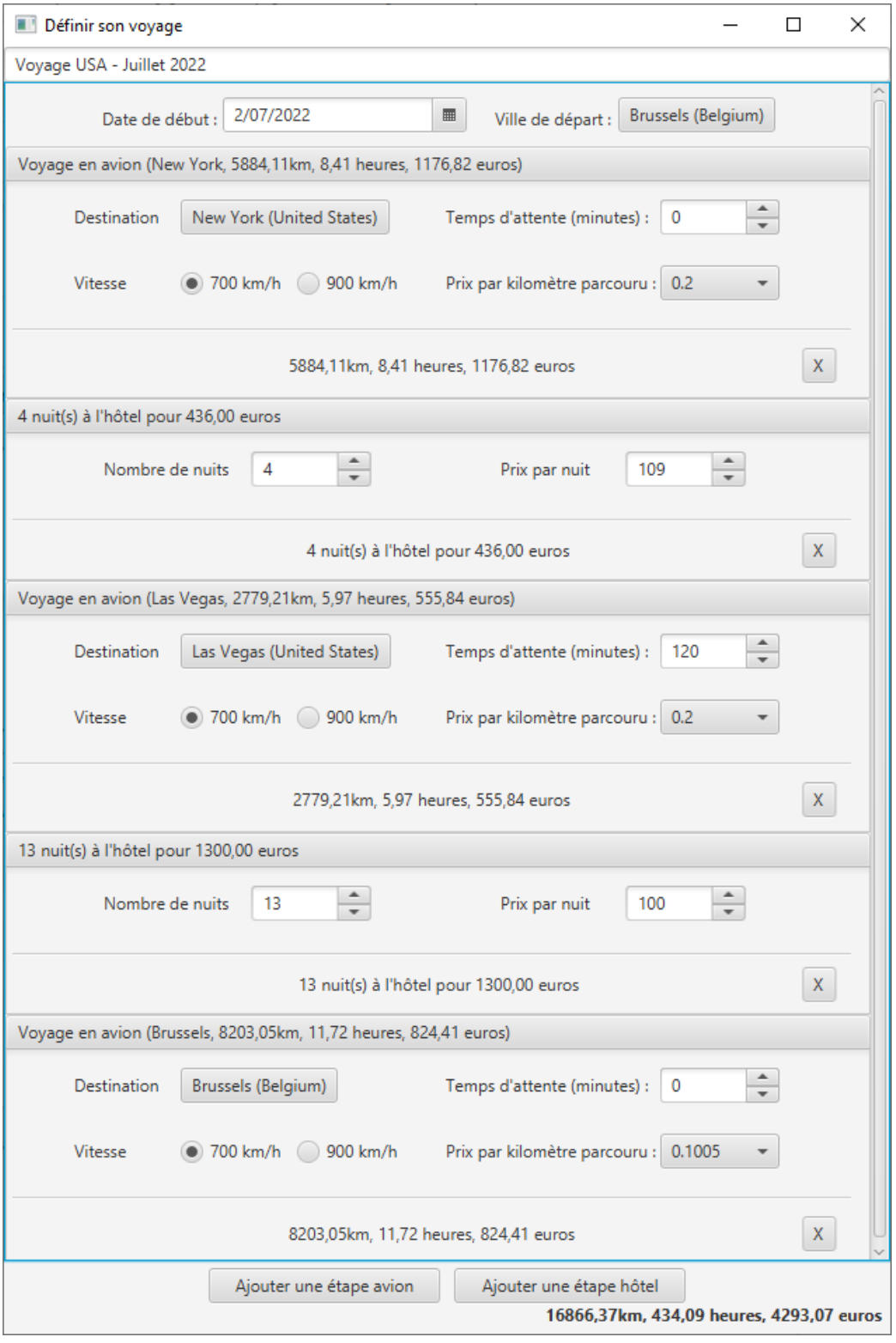
Programme 1 : la vue de tous les voyages



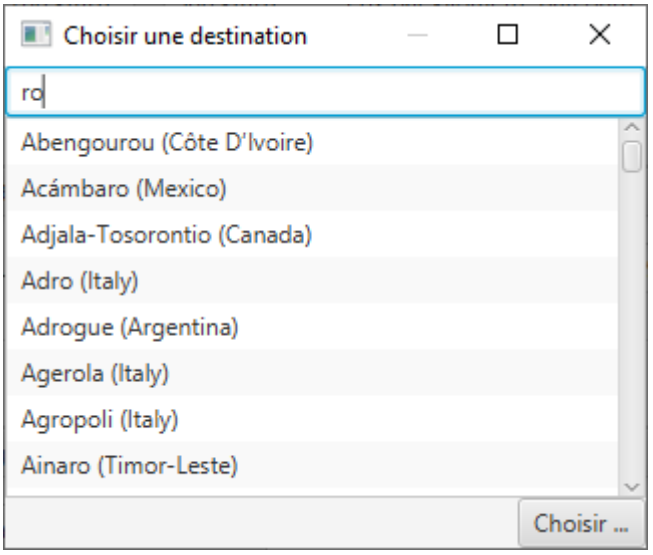
Ce programme présente une vue "Voyages" et une vue "Définir son voyage". Toutes les informations qu'elles contiennent sont fournies par le programme serveur.

Ces informations sont lues au démarrage du programme serveur depuis un fichier nommé “travels.bin” (en format binaire, via `ObjetInputStream`). A chaque modification reçue par le serveur depuis les clients, le fichier est mis à jour.

Quand on clique le bouton “Voir” d’un voyage de la vue “Voyages”, on arrive sur la vue “Définir son voyage” :



Quand on doit choisir une destination, la fenêtre suivante s'affiche :



Ces destinations viennent d'un fichier nommé "mapInfo.txt" disponible ici. Chaque enregistrement est composé de 11 lignes :

1. Le nom de la ville
2. *Le nom de la ville en ASCII*
3. La latitude
4. La longitude
5. Le nom du pays
6. *Le nom du pays en norme ISO2*
7. *Le nom du pays en norme ISO3*
8. *Le nom de la capitale administrative*
9. *Le nom de la capitale*
10. *La population*
11. *Un id*

Seuls les champs non italiques sont utiles ici. Ne changez pas ce fichier mais ignorez les lignes inutiles lors de la lecture.

Le calcul de la distance entre deux villes se fait via la fonction suivante (que vous pouvez évidemment changer pour l'adapter à vos classes, comme par exemple, retirer le `static`, changer les paramètres, …):

```
/**
* Calculate distance between two points in latitude and longitude.
* Uses Haversine method as its base.
*
* @param lat1 Latitude of start point
* @param lon1 Longitude of start point
* @param lat2 Latitude of end point
* @param lon2 Longitude of end point
* @returns Distance in Meters
*/
public static double distance(double lat1, double lat2, double lon1,
double lon2) {

    final int R = 6371; // Radius of the earth

    double latDistance = Math.toRadians(lat2 - lat1);
    double lonDistance = Math.toRadians(lon2 - lon1);
    double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
            + Math.cos(Math.toRadians(lat1)) *
Math.cos(Math.toRadians(lat2))
            * Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double distance = R * c * 1000; // convert to meters

    distance = Math.pow(distance, 2);

    return Math.sqrt(distance);
}
```

Critères de cotation

- Il faut que la compilation se déroule sans problème.
- Toutes les fonctionnalités doivent être implémentées.
- Il faut bien gérer les exceptions (réseaux, fichiers, entrées utilisateurs, …).
- Il faut que tous vos modèles soient testés complètement avec des tests d'unité.
- Il ne faut pas de complexité ou d'optimisation inutiles. Restez simple mais pas simpliste (par exemple, la duplication de code ne donne pas un code simple).
- Les interfaces graphiques doivent être les plus proches possibles de celles présentées dans la vidéo et les images.
- Il ne faut pas de duplication dans la gestion des vues communes.
- Chaque classe et chaque méthode doivent avoir une responsabilité clairement définie.
- Il faut que vous soyez cohérent avec vos propres règles de codage.
- Chaque méthode ne doit pas faire plus d'une dizaine de lignes.
- Chaque classe et méthode doit être documentée en Javadoc.
- Veillez à une bonne séparation en modèles, vues et contrôleurs.

Modalités de remise

Vous pouvez travailler par groupe de 2. Chaque groupe doit m'envoyer un mail contenant les noms, matricules et utilisateurs gitlab. Chaque groupe reçoit alors un repository GIT sur lequel vous commitez régulièrement l'avancement de votre projet (au moins 1 fois par semaine). Votre repository doit contenir un fichier README.md qui contiendra l'avancement de votre application (ce qui a été fait) et le planning prévu pour terminer le projet. Ce document doit être mis à jour toutes les semaines (maximum la veille du labo). De plus, vous devez me montrer votre avancement chaque semaine lors du labo ainsi que le planning mis à jour pour les semaines suivantes.

Le dernier commit sur la branche principale avant le 20 mai 2022 à 23h59 sera considéré comme votre remise finale.

L'examen de laboratoire sera un ensemble de modifications à effectuer sur votre projet endéans un temps imparti (de manière individuelle).

La seconde session est la continuation de ce projet et son examen sera de même nature.

Bon travail !