TP5 : *Ivy*, *Rejeu* et image radar

Objectifs :

- Savoir utiliser le bus logiciel *Ivy* pour mettre en œuvre la répartition dans une application *Java*,
- Savoir utiliser les mécanismes de gestion d'événements en *Java*,
- savoir utiliser le simulateur de trafic aérien *rejeu* dans une application *Java*.

Documentation :

Bus logiciel *Ivy*
https://www.eei.cena.fr/products/ivy/

Javadoc *Ivy*
https://www.eei.cena.fr/products/ivy/documentation/ivy-java-api/index.html?fr/dgac/ivy/package-summary.html

Expressions régulières pour *Ivy* (jointe en Annexe)

Documentation électronique *rejeu* (fournie avec le code de l'exercice 3)

Exercice 1 : Communications sur le bus *Ivy*

Ivy est un bus logiciel, c'est à dire un système réparti permettant d'interconnecter sur un même réseau plusieurs composants logiciels qui peuvent s'exécuter sur des machines différentes. *Ivy* a été conçu pour prototyper des systèmes interactifs, principalement des systèmes de l'ATC.
Le principe de communication est le suivant : chaque système (identifié par la suite comme un client) peut envoyer des messages textuels sur le bus, et s'abonner à des messages dont les caractéristiques sont spécifiées par une expression régulière (regexp, syntaxe et exemples en annexe). Lorsqu'un message est émis sur le bus par un client quelconque, tout autre client abonné à la regexp correspondante reçoit un événement qui lui permet de réagir.
Par défaut, tout client met en place, si il n'existe pas déjà, un bus *Ivy* sur l'interface de réseau *loopback* (et sur le port 2010), ce qui fait que seul vous (sur votre machine) pouvez voir les messages. Pour voir les messages de toute la salle il faudrait passer en paramètre l'adresse de broadcast du réseau votre salle de TP (xxx.xxx.xxx.255). Enfin, le broadcast étant restreint en interne à des réseaux locaux, il n'est pas possible d'utiliser *Ivy* sur Internet. Dans la suite du TP vous travaillerez en loopback, donc il est inutile de spécifier une adresse au lancement des clients.

Depuis deux terminaux distincts lancez deux instances de l'application de test/débuggage *ivyprobe* :

```
$ ivyprobe
```

Abonnez l'un des *ivyprobe* à l'expression régulière `^hello (.*)` au moyen de la commande bind.
Syntaxe de deux commandes *ivyprobe* utiles (attention au point au début de la commande !) :

```
.bind mon_expression_reguliere
.help
```

Envoyez le message "hello toto" sur le bus depuis l'autre *ivyprobe*.

Que reçoit la première application ?

Exercice 2 : Utilisation du bus *Ivy* en java

2.1 Récupérez les fichiers sur e-campus et collez le contenu du dossier src dans un nouveau projet *Java*. Que se passe-t-il ?

2.2 Pour remédier à ce problème il faut déclarer la librairie *ivy-java-1.2.18.jar* (située dans le dossier lib) dans le projet :

- Faites un clic droit sur le nom du projet,
- sélectionnez l'item de menu nommé « Properties »,
- sélectionnez la catégorie « Librairies »,
- appuyez sur le bouton « + » en regard de l'inscription « Classpath »
- choisissez l'item de menu « *Add JAR/Folder* »,
- sélectionnez la librairie *ivy-java-1.2.18.jar*.

Cette librairie ne sera pas recopiée dans le projet (seul le chemin vers cette librairie est mémorisé). Quel est l'intérêt de cette méthode ? Quel est son inconvénient ?

Quelle autre solution pouvez vous proposer ?

2.3 En vous aidant de la javadoc de l'API *Ivy*, modifiez le code pour réagir aux messages du type "Hello PRENOM1, my name is PRENOM2" et renvoyer "Nice to meet you PRENOM2!", où PRENOM1 et PRENOM2 seront deux expressions régulières simples décrivant n'importe quel nom.
Testez votre programme avec *ivyprobe*.

2.4 Modifiez votre code pour utiliser les expressions lambda de *Java 8*.
Aide : ici `IvyMessageListener` est une interface fonctionnelle (dite « SAM » dans la terminologie *Java 8*), on peut donc passer en 2ème argument de la méthode `bindMsg()` directement le bloc de code correspondant au contenu de l'implémentation de son unique méthode, plutôt que de se « fatiguer » à écrire une classe interne anonyme qui implémente cette interface.

Exercice 3 : Récupération des données *rejeu*

Rejeu est un simulateur de trafic aérien utilisé avec *Ivy* pour prototyper des systèmes de l'ATC, principalement des images radar.

3.1 Depuis un terminal, allez dans le répertoire contenant *rejeu*, donnez lui des droits d'exécution (`chmod +x rejeu`) puis lancez-le avec la commande suivante :

```
$ ./rejeu trafic.txt -s auto
```

Observez les messages transmis par le biais d'*ivyprobe* (n'oubliez pas de vous abonner à tous les messages transitant sur le bus depuis votre instance d'*ivyprobe*). Vous pourrez voir en particulier des messages TrackMovedEvent, émis pour chaque mouvement d'appareil. Ce sont eux qui nous intéressent ici.

3.2 Cherchez dans la documentation de *rejeu* la syntaxe exacte de ces messages. Déduisez en une expressions régulière qui permet de capturer le numéro de vol, l'indicatif d'appel, la vitesse, l'altitude, le cap, la tendance et les coordonnées au format *Cautra*.

3.3 L'objectif de cette question est de créer une application écoutant les messages de *rejeu* sur le bus et affichant :

- l'identifiant du trafic lors du premier contact,
- l'identifiant du trafic et ses coordonnées chaque fois qu'elles changent.

Pour commencer à architecturer proprement votre code vous utiliserez telle quelle la classe principale *Ex3Question3_IvyRejeuApplication* fournie et vous créerez un package *fr.enac.sita.visuradar.communication* avec une classe *Ex3Question3_CommunicationManager* à qui vous confierez la gestion des communications sur le bus.

Exercice 4 (pour les killers !!!) : application à une image radar

Vous allez maintenant vérifier que les infos récupérées depuis *rejeu* sont les bonnes en les visualisant sur une image radar fournie. Cette image radar est développée en *JavaFX*, une bibliothèque graphique avec laquelle vous apprendrez à développer en cours de Programmation événementielle. Pour le moment vous allez juste utiliser l'appli fournie en suivant la procédure énoncée ci-dessous :

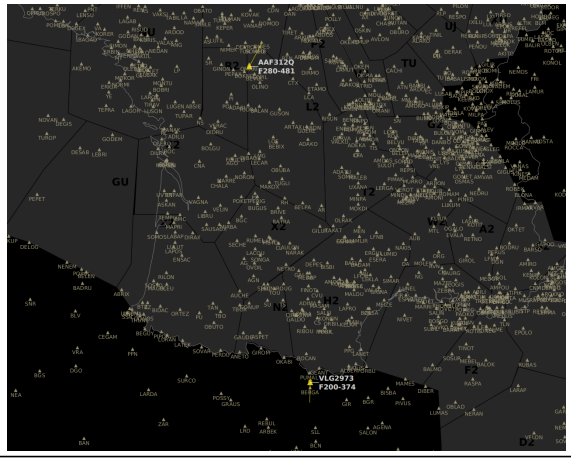
- Créez un projet *JavaFX* nommé *VisuRadarCommunicationTests* (dépliez l'item « Java with Ant » dans la fenêtre de création d'un nouveau projet et choisissez l'option *JavaFX*),
- dans la fenêtre de création du nouveau projet veillez à décocher la case "Create Application Class" (la classe principale est fournie dans le code à importer),
- placez à la racine de votre projet un dossier lib contenant les librairies *ivy-java-1.2.18.jar* et *VisuRadarCommunicationTests.jar* (cette librairie fournit l'ensemble de la visualisation et des interactions nécessaires pour tester vos communications avec *rejeu*),
- déclarez ces librairies dans votre projet comme fait dans l'exercice 2,
- placez le <u>contenu</u> du dossier *data* à la racine de votre projet,
- placez le package *fr.enac.sita.visuradar* dans le dossier src de votre projet.

C'est ce dernier package qui contient la classe principale de l'application. Vous modifierez cette classe et ajouterez votre dernier gestionnaire de communications, que vous modifierez pour alimenter le modèle de données de l'application avec les infos des vols provenant de *rejeu* (créations, modifications…).

Ce modèle de données est fourni dans la bibliothèque *VisuRadarCommunicationTests.jar* sous la forme d'un singleton nommé *Model*, qui centralise toutes les données de l'application. Un singleton est un *patron de conception* qui assure l'existence d'une instance unique d'une classe donnée, et que l'on peut invoquer directement depuis n'importe quelle autre classe dans l'application. Ce modèle de données est « écouté » par les autres classes fournies grâce aux services d'observabilité offerts par *JavaFX* (que nous détaillerons en cours de Programmation événementielle) et tout est affiché automatiquement à chaque modification.

Quelques indications pour utiliser la librairie :

- Obtenir le modèle : `Model.INSTANCE`
- Obtenir la liste des vols dans le modèle : `getFlightList()`
- … → Sous NetBeans vous pouvez explorer les services offerts par les différentes classes de votre application ou de l'une de ses bibliothèques référencées grâce aux palettes *Navigator* (menu Window > Navigator) et *Javadoc* (menu Window > IDE Tools > Javadoc Documentation),
- Vous pouvez également vous aider de la javadoc de la bibliothèque fournie (dossier *VisuRadarCommunicationTests_javadoc*).



Annexe : Expressions régulières pour *Ivy*

Un expression régulière est une chaîne de caractères décrivant la structure et le contenu d'un texte en suivant une syntaxe particulière. Les patterns sont reconnus dans les applications utilisant les expressions régulières (*regexp*). Lorsqu'un pattern est reconnu (un *match*), l'application peut réagir en fonction.

Plusieurs versions de syntaxe existent. *Ivy* utilise les Perl Compatible Regular Expression (*pcre*).
Doc pcre : http://perldoc.perl.org/perlre.html#Regular-Expressions

En résumé, on construit une *regexp* en utilisant des caractères (ceux-là seront reconnus tels quels) et des expressions employant des méta-caractères (*modifiers*, *classes* et *quantifiers*) qui permettent de décrire des structures reconnaissables.

Principaux *modifiers*

^	Match the beginning of the line
.	Match any character (except newline)
$	Match the end of the string (or before newline at the end of the string)
|	Alternation
\	Quote the next metacharacter

Principales *classes*

\S	Match a non-whitespace character
\d	Match a decimal digit character
\D	Match a non-digit character
[...]	Match a character according to the rules of the bracketed character class. Example: [a-z] matches "a" or "b" or "c" ... or "z"

Principaux *quantifiers*

*	Match 0 or more times
+	Match 1 or more times
?	Match 1 or 0 times
{n}	Match exactly n times
{n,}	Match at least n times

Quelques exemples de *regexp*

(.*)	correspond à n'importe quelle expression
(\d*)	correspond à une séquence de chiffres
(\d{4})	correspond à un entier de 4 chiffres
^salut (toto|titi) !$	correspond à "salut toto !" ou "salut titi !" exclusivement

En général, plus la regexp est précise, meilleures sont les performances de l'application qui les utilise.
On peut tester les expressions régulières sur plusieurs sites, par exemple : http://www.regextester.com