Exercices

Pour l'exercice qui suit, il est utile de disposer :

- jdk9,
- un outil d'édition de code

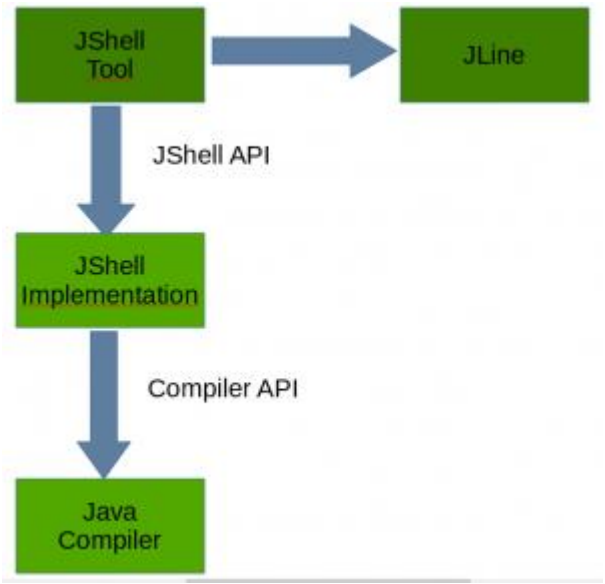
Exercice1

But de l'exercice : utiliser les nouveautés java 9 et surtout jshell

Avant Java 9, tester l'affichage d'un bout de code affichant Hello World, nécessitait un ScrapBook sous éclipse.

Lorsque du code est entré dans la console de `JShell`, il est traité par `JLine`. C'est une librairie Java pour gérer les saisies sur console.
Une fois le code saisi, il est analysé par `JShell` afin d'en déterminer le type (méthode, variable…) puis est encapsulé dans une classe en respectant les règles suivantes :

- Tous les imports sont placés en haut de la classe.
- Les variables, méthodes et déclarations de classes deviennent des membres statiques
- Les expressions et déclarations sont encapsulées dans une méthode à l'intérieur de la classe.



Travail à faire :

1. Configurez votre variable d'environnement JAVA_HOME afin qu'elle pointe vers le répertoire de le JDK 9.
2. Lancez `jshell` qui se trouve dans le répertoire bin\
3. Pour notre 1er test, afficher « exercice1. Pour ce faire, tapez la commande suivante : `System.out.println("exercice1");`
4. Listez les imports chargés au démarrage `/list -start`
5. Déclarer une variable entière `exercice` à 1 ;
6. Créez une interface `Printable` avec :
    a. Une méthode `print private` qui affiche une chaine de caractères s sur un `Printer` passés en paramètre,
    b. Une méthode `error public` qui affiche une `Exception` e sur un `Printer` passés en paramètre,
    c. Une méthode `debug public` qui affiche une `Map` m sur un `Printer` passés en paramètre,
7. Créez une classe `Event` qui implémente `Printable` avec :
    a. Un attribut `title` chaine de caractères
    b. Un attribut `start` date dans le futur par rapport au calendrier courant

   c. Un constructeur, des getters et setters où est contrôlé `start` par rapport à la date locale et le déclenchement de `error`, si la date est passée et le déclenchement de `debug` dans le cas contraire,
8. Faire un programme principal où est instancié un `Event` pour chaque jour de formation.
9. Supprimer la variable `exercice` avec `/drop` ,
10. Lister l'ensemble des variables créées dans JShell avec `/vars` ,
11. Lister l'ensemble des ytpes créées dans JShell avec `/types` ,
12. Importer la classe `LocalDate` pour l'utiliser dans la méthode `debug`.

Pour les exercices qui suivent, il est utile de disposer :

- jdk9, jdk10, jdk11, jdk12
- un outil d'édition de code
- un outil de construction de livrable

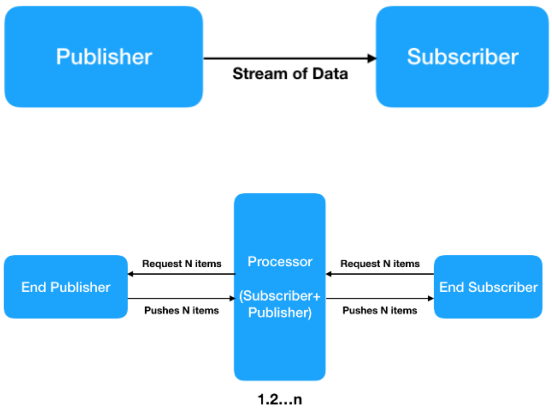
Via un IDE du type Eclipse, les exercices suivants sont à construire par l'emploi d'un builder externe du type Maven (version supérieure à 3.3.7)

Exercice2

But de l'exercice : utiliser les nouveautés l'API Flow

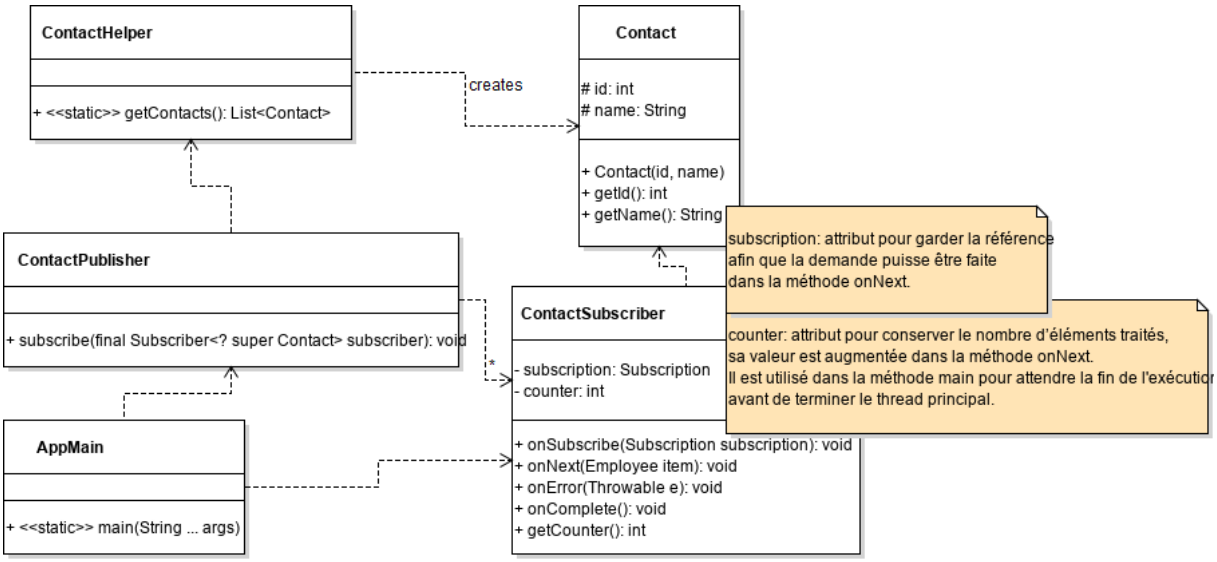
L'utilisation de Stream Reactive est basée sur les définitions issues de https://www.reactivemanifesto.org/, Java 9 a introduit le support des flux réactifs via l'API `java.util.concurrent.Flow`.

Les flux réactifs concernent le traitement asynchrone du flux, il devrait donc y avoir un publisher et un subscriber. Le publisher publie le flux de données et le subscriber utilise les données.
Parfois, les données sont transformées entre le publisher et le subscriber. Le processeur est l'entité située entre le publisher final et le subscriber pour transformer les données reçues du publisher afin que le subscriber puisse les comprendre.
Il peut y avoir une chaîne de processeurs.



Travail à faire :

1. Configurez votre variable d'environnement JAVA_HOME afin qu'elle pointe vers le répertoire de le JDK 9.
2. Via un IDE créez un projet maven, avec une version supérieure à 3.3.7
   a. Déclarez des propriétés pour que le compilateur soir en java 9
   b. Ajoutez un encoding pour les fichiers en utf-8
3. Créez un package `exercice2.streams` afin de coder les classes du diagramme suivant. Il faut connaître les classes de l'API Flow.
   Soit une classe `Contact` utilisée pour créer le flux de messages à envoyer du publisher au subscriber. La classe `ContactHelper` dispose d'une méthode `getContacts` pour retourner une liste de `Contact` qui sera tranformée en stream.



La classe `ContactSubscriber` qui implémente `Subscriber<Contact>` avec:

- La requête de subscription est appelée dans la méthode `onSubscribe` pour démarrer le traitement. Notez également qu'elle a de nouveau appelé la méthode `onNext` après le traitement de l'élément, ce qui oblige le publisher à traiter l'élément suivant.
- `OnError` et `onComplete` affichent des infos, mais dans le scénario réel, ils doivent être utilisés pour effectuer des mesures correctives en cas d'erreur ou le nettoyage des ressources lorsque le traitement est terminé.

La classe `ContactPublisher` qui hérite de la classe `SubscriptionPublisher<Contact>` ne possède qu'une méthode `subscribe` pour tracer dans un log les abonnements.

La classe `AppMain` joue le rôle de scénario de test pour lancer le publisher, le configurer par rapport au stream de Contacts et le subscriber pour les recevoir et les traiter.

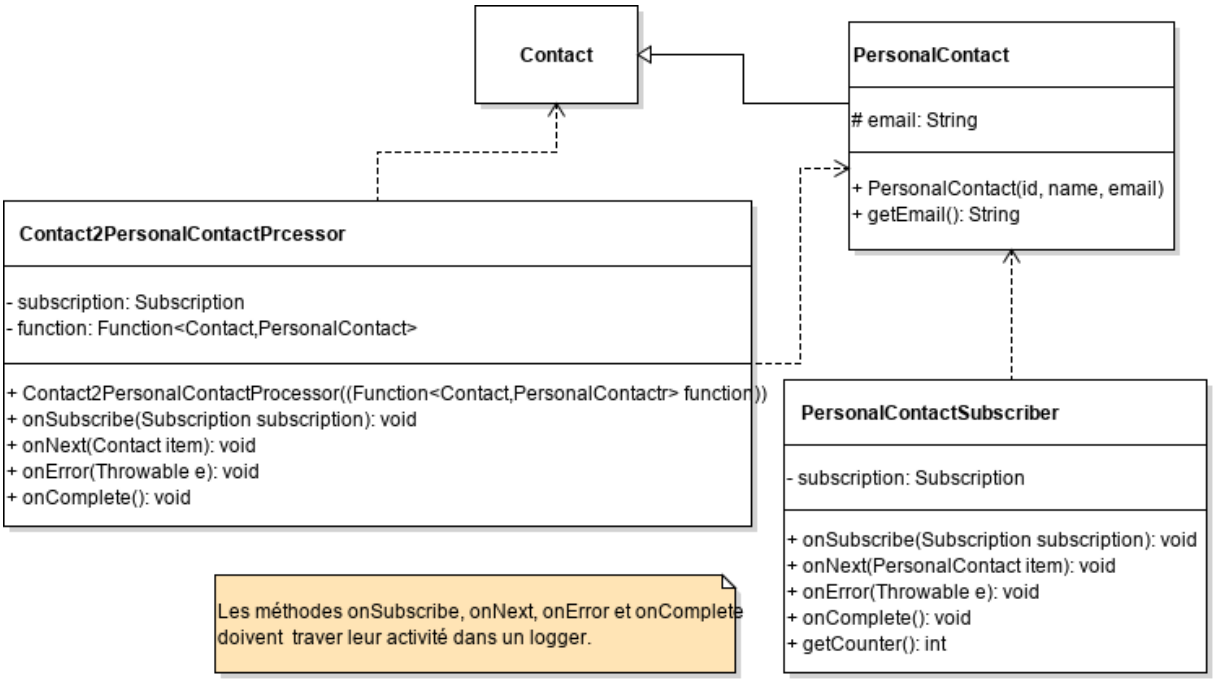
4. Transformation de message via un processeur pour transformer le message entre un publisher et un subscriber. La démarche est :

Faire un autre subscriber nommé `PersonalContactSubscriber` qui s'abonne à un type de message différent à traiter tel que `PersonalContact`.

Faire un processeur nommé `Contact2PersonalContactPrcessor` qui hérite de `SubscriptionPublisher<PersonalContact>` et implémente `Processor<Contact, PersonalContact>`.

- `function` sera utilisée pour convertir l'objet `Contact` en objet `PersonalContact`. Nous convertirons le message `Contact` entrant en message `PersonalContact` via la méthode `onNext`, puis nous utiliserons la méthode d'envoi `SubmissionPublisher` pour l'envoyer au subscriber.
- Puisque `Processor` fonctionne à la fois comme subscriber et comme publisher, nous pouvons créer une chaîne de processeurs entre les publisher finaux et les subscriber.

Refaire la classe `AppMain` avec son scénario de test afin d'utiliser la seconde classe de subscriber avec en amont un le processeur de transformation.



5. Nous voulons utiliser la méthode d'annulation d'abonnement pour ne plus recevoir de message dans le subscriber. Notez que si nous annulons l'abonnement, le subscriber ne recevra pas le signal `onComplete` ou `onError`.
   Enrichir la méthode `onNext(Contact item)` dans lequel le subscriber ne consomme que 3 messages, puis annule l'abonnement.
   Ces 3 messages devrons être écrit dans un fichier conntact.txt dont la gestion sera faite avec un *try with resource*s.
6. Optionnel Pour implémenter un algo de back pressure, il faut enrichir notre implémentation en suivant la démarche de référence sur
   https://github.com/ReactiveX/RxJava/wiki/Backpressure

Exercice3

But de l'exercice : construire des composants respectant jigsaw

En termes simples, la modularité est un principe de conception qui nous aide à atteindre un couplage lâche entre les composants via des contrats clairs et dépendances entre composants implémentation cachée utilisant une encapsulation forte

Travail à faire :

1. Configurez votre variable d'environnement JAVA_HOME afin qu'elle pointe vers le répertoire de le JDK 9.
2. Le but de cet exercice est de faire plusieurs classes de traductions de mots (une langue par composant) ainsi qu'un module principale qui les utilisent.
   Faire un premier composant contenant une interface Translatable et une classe AppMain qui l'utilise