SCIENCES SORBONNE UNIVERSITÉ

**SRCS : Systèmes Répartis Client/Serveur**

# TME 5 : Messagerie instantanée en ligne

**Objectifs pédagogiques :**

- **protobuf/gRPC**
- **Interface de service**
- **Publish/subscribe**

## Prérequis

Vous devez être en mesure de créer un projet eclipse en liant les différentes librairies java de Google nécessaires au bon fonctionnement de protobuf et gRPC.
Pour compiler et exécuter des programmes qui utilise Protobuf et Grpc il est nécessaire d'avoir les librairies java suivante :

- protobuf
- grpc
- guava
- perfmark

Si vous êtes sur une machine de la PPTI, ces librairies sont déjà installées et compilées. Si vous êtes sur votre propre machine, il est nécessaire que vous téléchargiez ces librairies et de les compiler ensuite.

Sous Eclipse il faut donc déclarer ces 4 nouvelles librairies java pour pouvoir ensuite les déclarer dans le build path d'un projet Eclipse (= le classpath du point de vue de la JVM) les différents fichiers jars de ces librairies. Pour chaque librairie faire ceci :

- Dans la barre de menu de Eclipse cliquez sur *Window*
- Cliquez sur *Preferences*
- Déroulez l'onglet *Java* sur le panneau de gauche
- Déroulez l'onglet *Build Path*
- Cliquez sur le menu *User Libraries*
- Cliquez sur le bouton *New...*
- Nommez la librairie
- Sélectionnez la nouvelle librairie créée et cliquez sur le bouton *Add External JARs...*
- Ajoutez l'ensemble des fichiers jars de la librairie :
  - ◇ pour protobuf, les jars se trouvent dans */usr/local/google_soft/protobuf/protobuf-3.11.0/jars*
  - ◇ pour grpc, les jars se trouvent dans */usr/local/google_soft/grpc/grpc-java/jars*
  - ◇ pour guava, les jars se trouvent dans */usr/local/google_soft/guava/guava-master/jars*
  - ◇ pour perfmark, les jars se trouvent dans */usr/local/google_soft/perfmark/perfmark-master/jars*
- cliquez sur *Apply and Close* pour valider.

N'oubliez pas de lier ces librairies à votre projet eclipse (clic droit sur le projet -> *Build Path* -> *Add libraries...* -> *User Library* -> cocher les librairies voulues)

**IMPORTANT :** Afin d'éviter un bug lié à une dépendance à une cinquième librairie, vous n'oublierez pas d'appliquer la méthode `disableStat` fournie dans la classe `srcs.grpc.util.BuilderUtil` sur chaque builder `ServerBuilder` et `ManagedChannelBuilder` que vous utiliserez.

# Objectifs

Dans ce TME, nous allons implanter un mécanisme de messagerie instantanée en ligne (un chat) qui permet de diffuser un message à tout utilisateur participant à une discussion. Chaque utilisateur possède un pseudo pour s'identifier sur le chat. Un utilisateur doit pouvoir :

- s'abonner à la messagerie (subscribe). Lors de l'abonnement, l'utilisateur fournit un pseudo et une adresse de rappel (ici une IP et un port). Si le pseudo n'est pas utilisé par un autre utilisateur, l'appel renvoie vrai. Il renvoie faux sinon et l'opération d'abonnement est annulée.
- se désabonner (unsubscribe).
- connaître la liste des utilisateurs connectés
- envoyer un message. Cette méthode renvoie le nombre d'utilisateurs a qui on a envoyé le message (mais qui ne l'ont pas nécessairement reçu)
- recevoir les messages envoyés par n'importe quel utilisateur dès que l'abonnement à la messagerie est actif

Architecturalement, le chat est géré par un serveur qui centralise l'ensemble des abonnés et qui rediffuse les messages à tout abonné. Les utilisateurs doivent donc avoir la capacité de recevoir des messages du serveur. Ainsi le serveur centralisateur respectera l'interface Java suivante :

.

```
package srcs.chat.implem;

import java.util.List;

public interface Chat {
	public boolean subscribe(String pseudo, String host, int port);
	public int send(String pseudo, String message);
	public List<String> listChatter();
	public void unsubscribe(String pseudo);
}
```

Un utilisateur respectera l'interface suivante :

.

```
package srcs.chat.implem;

public interface MessageReceiver {
	void newMessage(String pseudo_expediteur, String contenu);
}
```

**Question 1**

En suivant ces interfaces, écrire un fichier protobuf permettant de définir les messages et les services qui seront mis en oœuvre dans cette application.

**Question 2**

Compiler avec la commande `proto` en suivant ce schéma et rafraîchir le projet eclipse. S'assurer que les fichiers produits compilent.

```
protoc
--proto_path=/chemin/vers/votre/dossier/contenant/votre/fichier/proto/a/compiler
--proto_path=/chemin/vers/fichiers/protos/protobuf
--plugin=/chemin/vers/le/plugin/grpcjava
```

```
--grpc-java_out=/chemin/vers/votre/repertoire/src/projet/eclipse
--java_out=/chemin/vers/votre/repertoire/src/projet/eclipse
votre_fichier.proto
```

N.B : Sur le machines de la PPTI :
- le répertoire des fichiers protobuf est */usr/local/protobuf-master/src*
- le fichier plugin grpc-java est */usr/local/grpc-java-master/compiler/build/exe/java_plugin/protoc-gen-grpc-java*

**Question 3**

Écrire la classe `srcs.chat.implem.ChatImpl` qui implante le code métier du serveur central. La rediffusion d'un message se fera de manière non bloquante.

**Question 4**

Écrire une classe `ChatProxy` qui implante l'interface `Chat` et qui fera office de mandataire client vers le service gRPC présent sur le serveur central. Cette classe définira un constructeur avec 3 arguments :
- une chaîne de caractères désignant le nom de la machine du client instanciant le proxy
- le port d'écoute du client
- un argument de type `MessageReceiver` dont la méthode sera appelée lorsqu'un message est reçu

La méthode `subscribe` devra déployer un serveur de rappel sur le client seulement si l'abonnement est possible (pseudo OK).

**Question 5**

Prendre connaissance du contenu du fichier de test *Testchat*, faire en sorte qu'il compile et s'assurer que les tests sont corrects.