# 1. La classe Broker

Cette classe agit comme un intermédiaire central pour établir et gérer les connexions entre différentes entités. Elle fournit les mécanismes de base pour créer des canaux de communication, que ce soit en tant que serveur ou en tant que client. L'interface de cette classe est la suivante :

```
abstract class Broker {
  Broker(string name);
  Channel accept(int port);
  Channel connect(String name, int port);
}
```

- **Broker(String name) :** Ce constructeur permet d'initialiser l'entité Broker avec un nom unique qui sera donc son identifiant, ce qui peut être utile pour l'identification dans un système distribué.
- **Channel accept(int port) :** cette méthode permet au serveur d'écouter et accepter les connexions entrantes sur un port choisit et passé en paramètre , elle est bloquante et retourne soit une exception si une erreur survient pendant l'opération ou si elle prend trop de temps, soit un objet de type « Channel » sur lequel le serveur va communiquer avec le client connecté.
- **Channel connect(String name, int port) :** cette méthode permet a un client de se connecter au serveur , elle prend en paramètre le port du serveur sur lequel on souhaite se connecter et le nom du broker du serveur. Elle est bloquante . et retourne , soit une exception si une erreur se produit pendant la connexion ou si elle prend trop de temps, soit un objet « Channel » sur lequel le client pourra communiquer avec le serveur.

# 2. La classe Channel

Cette classe représente un canal de communication bidirectionnel, FIFO et sans perte. Agissant comme une interface pour lire et écrire des données. Un canal pourra être utilise pour la communication entre le serveur et plusieurs clients; le canal devra donc être synchronisé sur toutes ses méthodes. L'interface qu'elle fournit est la suivant:

```
abstract class Channel {
  int read(byte[] bytes, int offset, int length);
  int write(byte[] bytes, int offset, int length);
  void disconnect();
  boolean disconnected();
}
```

- **int read(byte[] bytes, int offset, int length)** Cette méthode est utilisée pour lire des données depuis le canal. Elle essaie de lire un nombre spécifié de bytes (length) et de les stocker dans le tableau bytes à partir d'une position donnée (offset). Elle retourne le nombre de bytes lus avec succès et zéro si elle a réussi a écrire tous ses bits ou elle retourne une exception qui donne le nombre de bit qu'il reste a lire en signalant que l'opération prend trop de temps et sinon retourne une exception qui dit le problème que rencontre l'opération.
- **int write(byte[] bytes, int offset, int length) :** Cette méthode est utilisée pour écrire des données sur le canal. Elle prend des bytes à partir du tableau bytes (de offset à length) et les envoie sur le canal. Elle retourne le nombre de bytes écrits avec succès et zéro si elle a réussi a écrire tous ses bits ou elle retourne une exception qui donne le nombre de bit qu'il reste a écrire en signalant que l'opération prend trop de temps et sinon retourne une exception qui dit le problème que rencontre l'opération.

- **void disconnect() :** Cette méthode est responsable de la fermeture propre du canal de communication. Une tache peut fermer le canal a tout moment indépendamment des autres . Aucune nouvelle opération d'écriture ou de lecture ne peuvent être démarrée sur un canal déconnectée.
- **boolean disconnected() :** Cette méthode permet de vérifier l'état du canal. Elle retourne true si le canal est fermé ou déconnecté, et false sinon. Si lors d'une écriture ou d'une lecture le canal est déconnecté, si c'est une lecture alors l'opération se termine normalement quand même