Programmation et modélisation orientées objet
(USAL5A & USAL2H)

Canaux de communication

Inconvénients des versions locales et distantes

1. Les versions locales utilisent des `BlockingQueue` de la JDK pour implanter les canaux de communication. Toutefois, leur utilisation directe dans le code ne permet pas de distinguer les extrémités (en lecture ou en écriture) des canaux. Ceci est une source d'erreur, que le compilateur ne peut pas détecter.

2. Les versions distantes utilisent des connexions TCP pour implanter les canaux de communication. Celles-ci permettent d'échanger des octets par défaut, mais pour échanger des objets, il faut les *sérialiser*. En java, des adaptateurs standard de la JDK permettent cela. Pour passer de la version locale à la version distante, il faut toutefois dupliquer puis modifier le code du client et du serveur :

   Il faut remplacer la classe `BlockingQueue` par les classes `ObjectInputStream` et `ObjectOutputStream` et les méthodes `put` et `take` de `BlockingQueue` par `writeObject` et `readObject` (et enfin `InterruptedException` par `IOException`).

   Dupliquer le code est problématique si l'on souhaite conserver (et maintenir) les deux versions (pour les tests par exemple qui sont plus performant dans la version locale).

3. Dans la version distante, il faut de plus introduire des *casts* à chaque utilisation de `readObject` car les classes `ObjectInputStream` et `ObjectOutputStream` ne sont pas génériques. Ceci est aussi une source d'erreur, que le compilateur ne peut pas détecter.

Pour remédier à ces différents inconvénients, tout en factorisant le code du client et du serveur, nous introduisons des interfaces Java pour modéliser l'extrémité d'un canal en lecture (resp. en écriture) permettant d'échanger des types de donnée génériques (les *casts* ne seront donc plus nécéssaires) :

```
interface ObjectReader<A> {
    A readObject() throws IOException;
}

interface ObjectWriter<B> {
    void writeObject(B arg) throws IOException;
}
```

1 Communications locales

Questions

1. Ecrire deux classes `LocalObjectReader<A>` et `LocalObjectWriter<B>` qui implantent respectivement les interfaces `ObjectReader<A>` et `ObjectWriter<B>` ci-dessus en utilisant l'interface `BlockingQueue<E>` de la JDK. L'exception `InterruptedException` sera propagée, mais convertie auparavant en `IOException`.

2. Modifier l'implantation de `RemoteList<E>` et de `RemoteMap<K,V>` pour utiliser ces interfaces (et leurs implantations locales).

2 Communications distantes

Questions

1. Ecrire deux classes `RemoteObjectReader<A>` et `RemoteObjectWriter<B>` qui implantent respectivement les interfaces `ObjectReader<A>` et `ObjectWriter<B>` ci-dessus en utilisant les classes `ObjectInputStream` et `ObjectOutputStream` de la JDK.

2. Tester `RemoteList<E>` et de `RemoteMap<K,V>` avec ces implantations des canaux distants. On ecrira une classe `ClientMain` pour lancer le client et une classe `ServerMain` pour lancer le serveur. Les méthodes `main` de ces classes devront bien sûr d'abord établir la connexion TCP et ajouter les adaptateurs pour la sérialisation d'objets.

A Package channel