Programmation et modélisation orientées objet
(USAL5A & USAL2H)

Client-serveur en réseau

Connexion TCP

On souhaite implanter une application client-serveur simple Java, où le serveur crée une socket d'écoute sur le port `55000` et attend qu'un client se connecte.

1. Implanter le client et le serveur en utilisant les classes `Socket` et `ServerSocket` de la JDK. On affichera un message à chacune des étapes, côté client et côté serveur (en utilisant la classe `Logger` côté serveur).

2. Afficher de plus les adresses IP et les numéros de port utilisés pour la connexion.

3. Compléter le code précédent pour que le client envoie un octet au serveur qui doit alors l'afficher, et renvoyer le double de ce nombre au client, qui affiche alors le résultat.

   Pour les communications, on utilisera les classes `InputStream` et `OutputStream` de la JDK (cf. figure en Annexe). Les instances sont obtenues à partir de la socket de la manière suivante :

   ```
   OutputStream os = s.getOutputStream();
   InputStream is = s.getInputStream();
   ```

Les `InputStream` et `OutputStream` permettent de communiquer uniquement des octets. Pour communiquer des données plus complexes, il faut utiliser des « adaptateurs », à choisir en fonction du type des données échangées.

1 Sérialisation des types primitifs (protocole basique)

On souhaite déployer en réseau l'application client-serveur qui implante « la liste distante » (protocole basique qui utilise des entiers). Pour cela nous utiliserons la possibilité offerte par la jdk de « sérialiser » des types primitifs, permettant ainsi de les envoyer à traver le réseau.

On utilisera les classes `DataInputStream` pour lire des entiers (méthode `readInt`) et `DataOutputStream` pour écrire des lignes (méthode `writeInt`). Les instances sont créées de la manière suivante :

```
DataOutputStream out = new DataOutputStream(os);
DataInputStream in = new DataInputStream(is);
```

Question

1. Implanter le protocole des listes distantes (client et serveur) pour le protocole de base utilisant des entiers.

2 Sérialisation de messages (protocole textuel)

On souhaite implanter une application client-serveur avec le protocole textuel simple suivant :

- le client envoie "hello"
- le serveur répond "hello"
- le client envoie "name" suivi de son nom
- le serveur répond alors "bye" suivi du nom du client
- le client répond enfin "bye"

On utilisera les classes `BufferedReader` pour lire des lignes (méthode `readLine`) et `PrintWriter` pour écrire des lignes (méthode `println`). Les instances sont créées de la manière suivante :

```
PrintWriter out = new PrintWriter(new BufferedOutputStream(os), true);
BufferedReader in = new BufferedReader(new InputStreamReader(is));
```

Questions

1. Implanter le protocole ci-dessus (client et serveur). On vérifiera que le protocole est respecté à la lettre coté client et côté serveur, sinon l'exception IOException sera levée (avec un message explicite).

3 Sérialisation d'objet (protocole de haut niveau)

On souhaite déployer en réseau l'application client-serveur qui implante « la liste distante » (protocole de haut niveau qui utilise des objets). Pour cela nous utiliserons la possibilité offerte par la JDK de « sérialiser » des objets de classes quelconques, permettant ainsi de les envoyer à traver le réseau. Ces classes doivent simplement implanter l'interface `Serializable` (qui ne contient aucune méthode, la sérialisation est réalisée directement par la JVM).

On utilisera les classes `ObjectInputStream` pour lire des entiers (méthode `readObject`) et `ObjectOutputStream` pour écrire des lignes (méthode `writeObject`). Les instances sont créées de la manière suivante :

```
ObjectOutputStream out = new ObjectOutputStream(os);
ObjectInputStream in = new ObjectInputStream(is);
```

Question

1. Implanter le protocole des listes distantes (client et serveur) pour le protocole de haut niveau utilisant les types `Request` et `Answer`.

A Input/Output streams et Reader/Writer

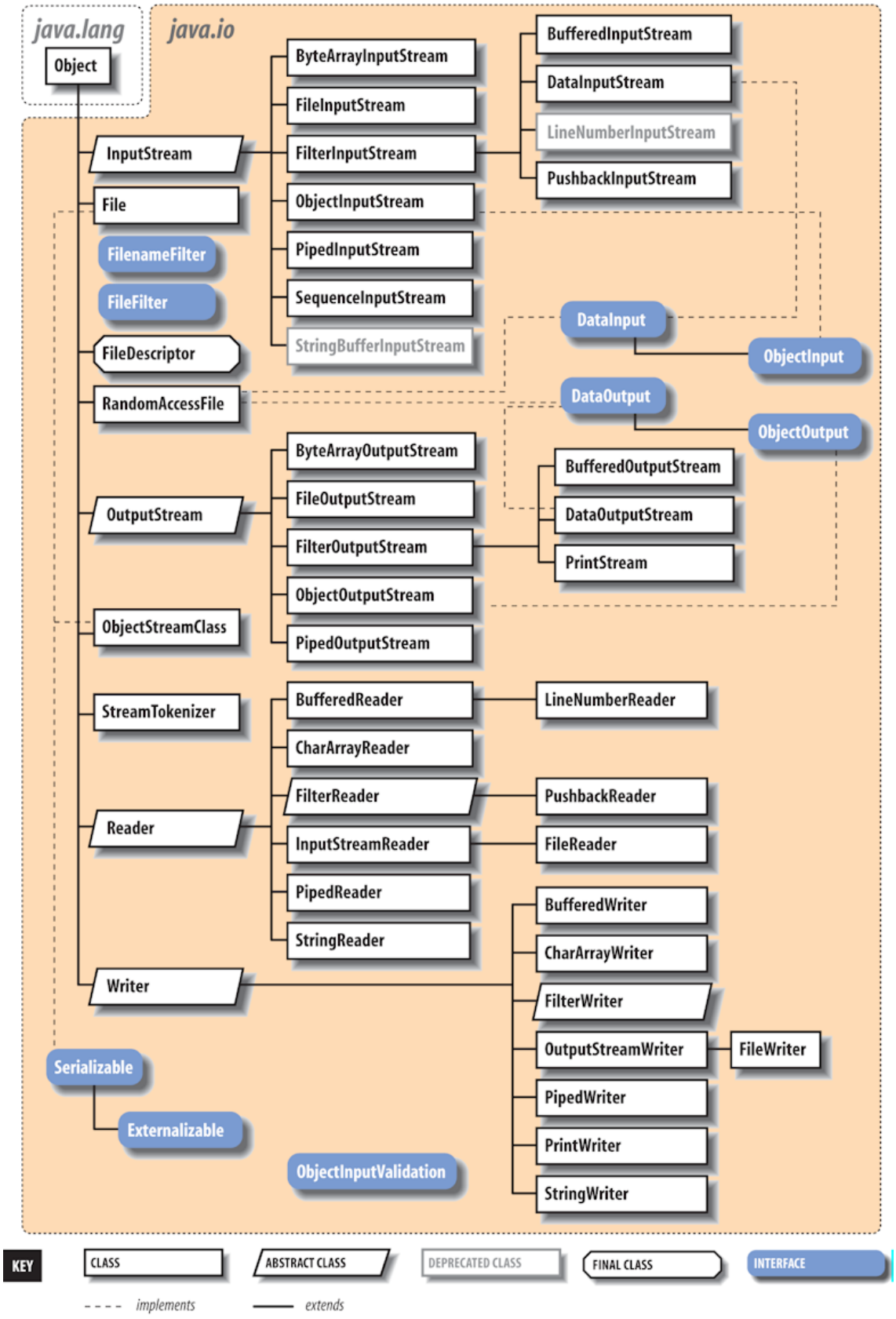

Figure tirée de « Learning Java », (4th Edition, 2013) de P. Niemeyer, D. Leuck.