Licence Pro DevOps

Java :
Réseau

Plan

1) Établir une connexion UDP
2) Établir une connexion TCP

UDP

- Rappel :
  - Mode **non connecté**
  - Pas de notion de serveur ou client
  - Pour envoyer un message :
    - l'emetteur doit connaître l'**IP** de la machine qui reçoit, et le **port** sur lequel elle écoute
    - Le récepteur doit écouter les messages arrivant sur un port donné
  - Connexion **"non fiable"** : pas d'accusé reception, ordre de reception non garanti...

UDP en Java

- Côté émetteur, code minimal :

```
String AdresseRecepteur = "127.0.0.1";
InetAddress adresse = InetAddress.getByName(AdresseRecepteur);

int portRecepteur = 2000;

String message = "hello world";

DatagramSocket socket = new DatagramSocket();

DatagramPacket paquet = new DatagramPacket(  message.getBytes(),
                                             message.length(),
                                             adresse,
                                             portRecepteur );

socket.send(paquet);
socket.close();
```

- Attention : ces méthodes peuvent lever des exceptions de type **SocketException** ou **IOException**
- Un unique socket peut servir pour l'envoi de plusieurs messages

UDP en Java

- Côté récepteur, code minimal :

```
int port = 2000;
int BUF_SIZE      = 1000;
byte[] buf = new byte [BUF_SIZE];

DatagramSocket socket = new DatagramSocket(port);
DatagramPacket paquet = new DatagramPacket(buf, BUF_SIZE);

socket.receive(paquet);

String texte= new String(buf, 0, paquet.getLength());
System.out.println("Reception d'un paquet provenant de "
                   +paquet.getAddress()
                   +" de port "+paquet.getPort()
                   + "\n MESSAGE= "+texte+"\n");
socket.close();
```

- Même remarque que précédemment concernant les exceptions
- Idem : un unique socket permet de recevoir plusieurs messages
- L'appel à receive est **bloquant**
  - généralement, il est encapsulé dans une boucle "infinie" contenue dans un thread

UDP en Java

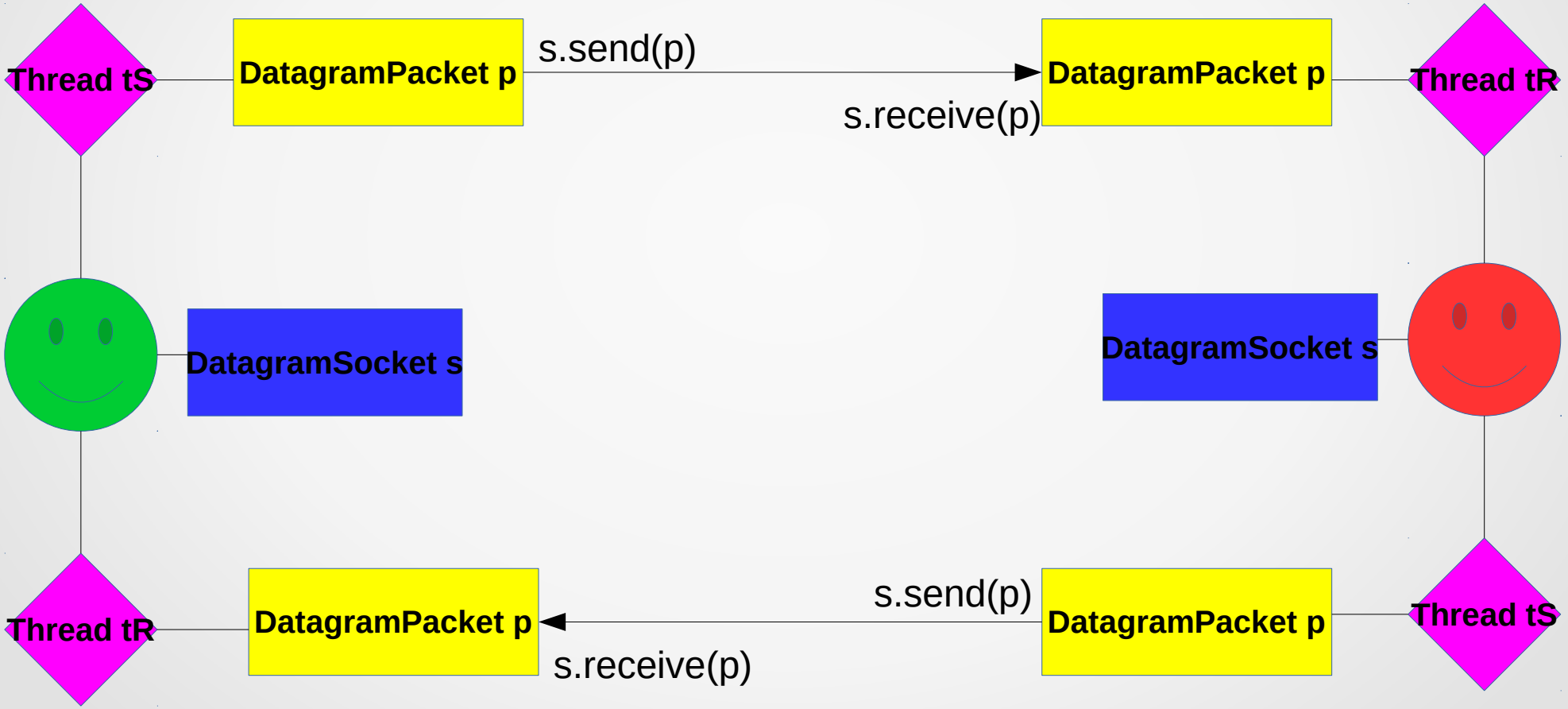
- Côté récepteur, code minimal :

```
int port = 2000;
int BUF_SIZE      = 1000;
byte[] buf = new byte [BUF_SIZE];

DatagramSocket socket = new DatagramSocket(port);
DatagramPacket paquet = new DatagramPacket(buf, BUF_SIZE);

socket.receive(paquet);

String texte= new String(buf, 0, paquet.getLength());
System.out.println("Reception d'un paquet provenant de "
                   +paquet.getAddress()
                   +" de port "+paquet.getPort()
                   + "\n MESSAGE= "+texte+"\n");
socket.close();
```

- Si l'émetteur envoie des messages avant la création du socket du récepteur, les messages sont perdus
- Une fois que le socket du récepteur est créé, les messages reçus sont stockés dans une file, et récupérés FIFO lors des appels à **receive**

UDP en Java

- Résumé d'une communication bilatérale :



TCP

- Rappel : mode “connecté”
  - Le serveur attend des nouvelles connexions
  - Le client demande une connexion
  - Le serveur accepte la connexion
  - Les deux peuvent alors communiquer dans les deux sens
- Connexion “fiable” : accusés reception, ordre garanti...

TCP en Java

- Côté serveur : création d'un SocketServeur pour écouter les nouvelles connexions :

```
int port = 2000;
ServerSocket servSocket = new ServerSocket(port);

Socket socket = servSocket.accept();
```

- L'appel à accept() est bloquant
- Ce code peut générer des IOException

TCP en Java

- Côté client : on crée un socket avec l'IP et le port du serveur

```
String adresseStr = "127.0.0.1";
InetAddress adresse = InetAddress.getByName(adresseStr);
int port = 2000;

Socket socket = new Socket (adresse, port);
```

- Ce code peut générer des IOException

TCP en Java

- Des deux côtés on dispose d'un socket avec lequel on peut créer :
  - Un flux de données entrant (réception de messages) :

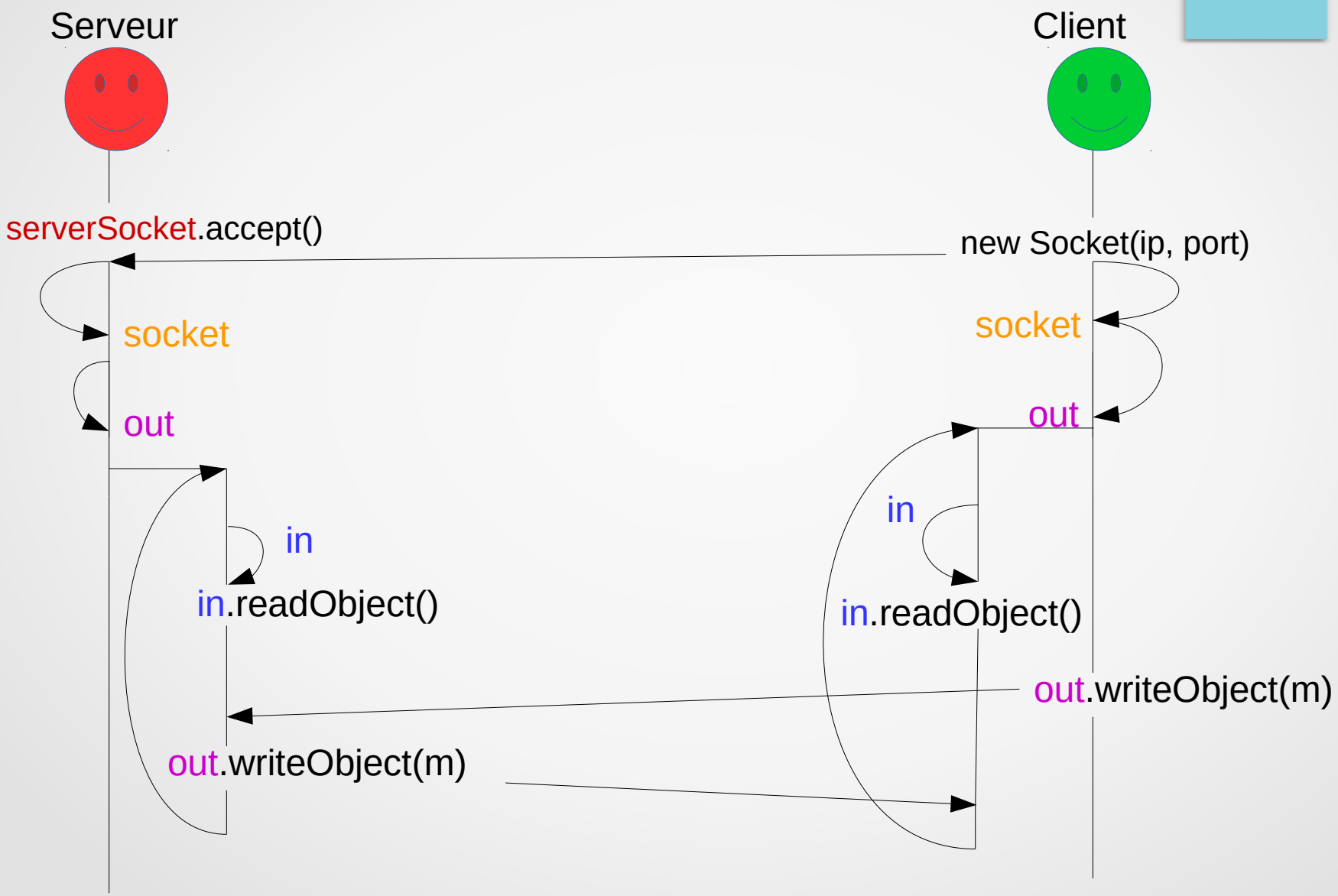
```
ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());

String mEnvoi = "Hello world";
out.writeObjects(mEnvoi);
```

  - Un flux de données sortant (envoi de messages) :

```
ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

String mRecu = (String) in.readObject();
```

- Généralement, ces deux parties sont dans des threads distincts
- L'appel à readLine() est bloquant
- readLine() renvoie **null** si et seulement si le client s'est déconnecté

TCP en Java


Serveur
Client
serverSocket.accept()
new Socket(ip, port)
socket
socket
out
out
in
in
in.readObject()
in.readObject()
out.writeObject(m)
out.writeObject(m)