



Département Informatique & des communications



Licence Science de l'Informatique -Génie Logiciel et Système d'Information (LSI2-GLSI)

Développement d'Applications Réparties (DAR)

Corinne Amel ZAYANI

Corinne.zayani@fss.usf.tn

<https://us2026.uvt.tn/course/view.php?id=9>

Semestre	Unité	Eléments Constitutifs de l'UE (ECUE)		Volume horaire Cours
Semestre 5	UEF520 :	ECUEF521	Développement Mobile	10,5
		ECUEF522	Développement d'applications réparties	21

LSI-GLSI

Semestre	Unité	Eléments Constitutifs de l'UE (ECUE)	Volume Crédit horaire	Coéfficients
5	UEF520 : Développement d'applications réparties	ECUEF522	21	3 1,5

Avant de commencer

- Activer votre compte microsoft
- <https://www.inscription.tn/OREgMx/Inscription.jsp>
 - précisez la clé d'inscription

Les accords



- » Venez à l'heure
- » Prenez des notes (Cours / TP)
- » Soyez assidu (pas plus que 3 absences)
- » Faites un résumé de 5 minutes au début et à la fin de chaque séance
- » Une partie de la note de l'examen de TP portera sur la participation via l'espace de cours disponible sur l'ENT : <https://us2026.uvt.tn/course/view.php?id=9>
- » Réponse aux QCMs
 - Participation aux forums
 - Dépôt des activités

Pré requis

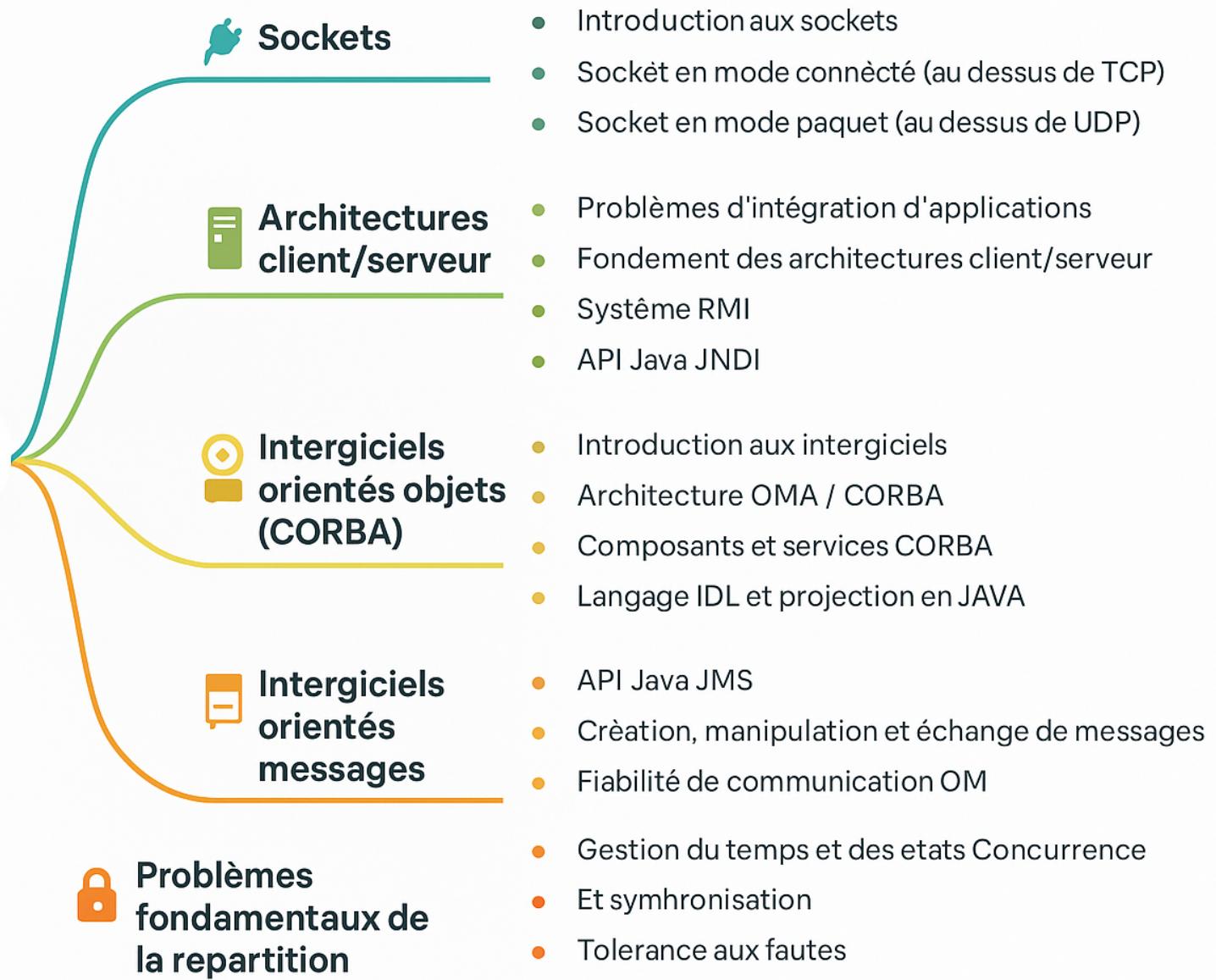
- » Méthodologie de conception UML
- » SE
- » Gestion de fichiers & DB
- » POO
- » Réseaux
- » Langage de programmation : JAVA

Objectifs générales

- » Distinguer entre application centralisé et répartie
- » savoir résoudre des problèmes liés à l'accès concurrentes
- » Développer des applications orientées objets distribuées en mode synchrone et asynchrone
- » Apprendre à programmer et exploiter d'applications réparties

Plan

Chapitres



CHAPITRE 1 :



- Introduction aux sockets
- Socket en mode connexe (au dessus de TCP)
- Socket en mode paquet (au dessus de UDP)

Objectifs spécifiques

- » À la fin de cet item, l'apprenant sera capable de :
 - » exploiter l'API Java java.net pour la programmation réseau
 - » Implémenter des communications synchrones et asynchrones.
 - » Utiliser les threads pour gérer plusieurs clients simultanément.
 - » Analyser les flux de communication et de traitement

»

Plan

- » Se rappeler de concepts de socket
- » Découvrir le package java.net
- » Savoir préparer une squelette de communication entre client/serveur
- » Commencer à distinguer entre les flux de communication et flux de traitement

Le modèle client/serveur

» C'est quoi son objectif ?

.....

.....

.....

» Représentez ce modèle en expliquant les éléments qui caractérisent une architecture client-serveur.

.....

.....

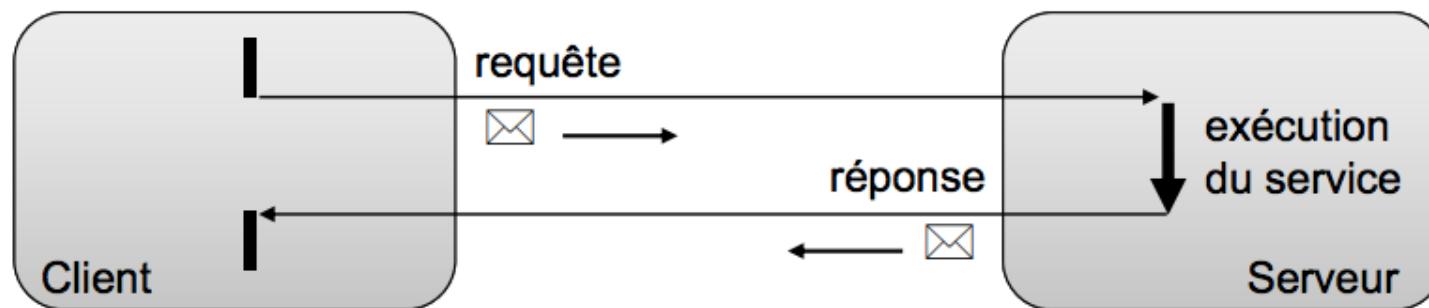
.....

.....

.....

Le modèle client/serveur

- » Son objectif : réduire les coûts
- » Le modèle client-serveur établit une communication entre deux processus informatiques via un échange de messages.
- » Les processus sont généralement réalisés sur des machines, systèmes d'exploitation et réseaux homogènes ou hétérogènes.
- » Le processus client externalise les services à exécuter au processus serveur.



Le modèle client/serveur

- » Dans une architecture client-serveur, la communication entre les deux entités repose sur l'échange de messages : **c'est ce qu'on appelle le flux de communication.**
- » Pour rendre ces échanges possibles, il est nécessaire d'établir un **protocole**,
- » Exp : sur le web, un client et un serveur échangent des documents HTML en s'appuyant sur le protocole
- » Le **protocole** est un ensemble de règles qui définissent :
 - » **le format des messages** (texte, binaire, etc.),
 - » **les types de messages** (requête, réponse, notification, etc.),
 - » et la manière dont l'émetteur et le récepteur doivent les **interpréter**.
- » L'interprétation, le traitement et la réaction aux messages reçus relèvent du **flux de traitement**.

Mise en oeuvre du modèle client/serveur

- » deux niveaux d'utilisation sont distingués pour assurer l'échange de messages entre un client et un serveur,
 - Niveau bas : recours direct aux protocoles de transport (TCP ou UDP) via les sockets. Ce niveau offre un contrôle fin de la communication, mais demande de gérer manuellement la connexion et les échanges. (détailé dans ce chapitre.)
 - Niveau haut : intégration de la communication au sein du langage de programmation grâce à des mécanismes comme le RPC (Remote Procedure Call). Ces solutions reposent elles-mêmes sur les sockets, mais masquent leur complexité en offrant une interface plus simple au développeur. (abordé dans les chapitres suivants).
 - Une socket peut être vue comme un canal de communication reliant deux programmes situés sur des machines éventuellement distinctes.

L'API socket

- **Mode connecté : TCP (Transmission Control Protocol)**

- Une liaison est explicitement ouverte, des messages sont échangés, puis la connexion est fermée.
- Le serveur conserve un état entre les différentes requêtes d'un même client.
- TCP garantit :
 - l'ordre des messages,
 - le contrôle de flux,
 - et la fiabilité des transmissions.
- Ce mode est particulièrement adapté aux échanges nécessitant une durée de communication plus longue (plusieurs requêtes/réponses successives).

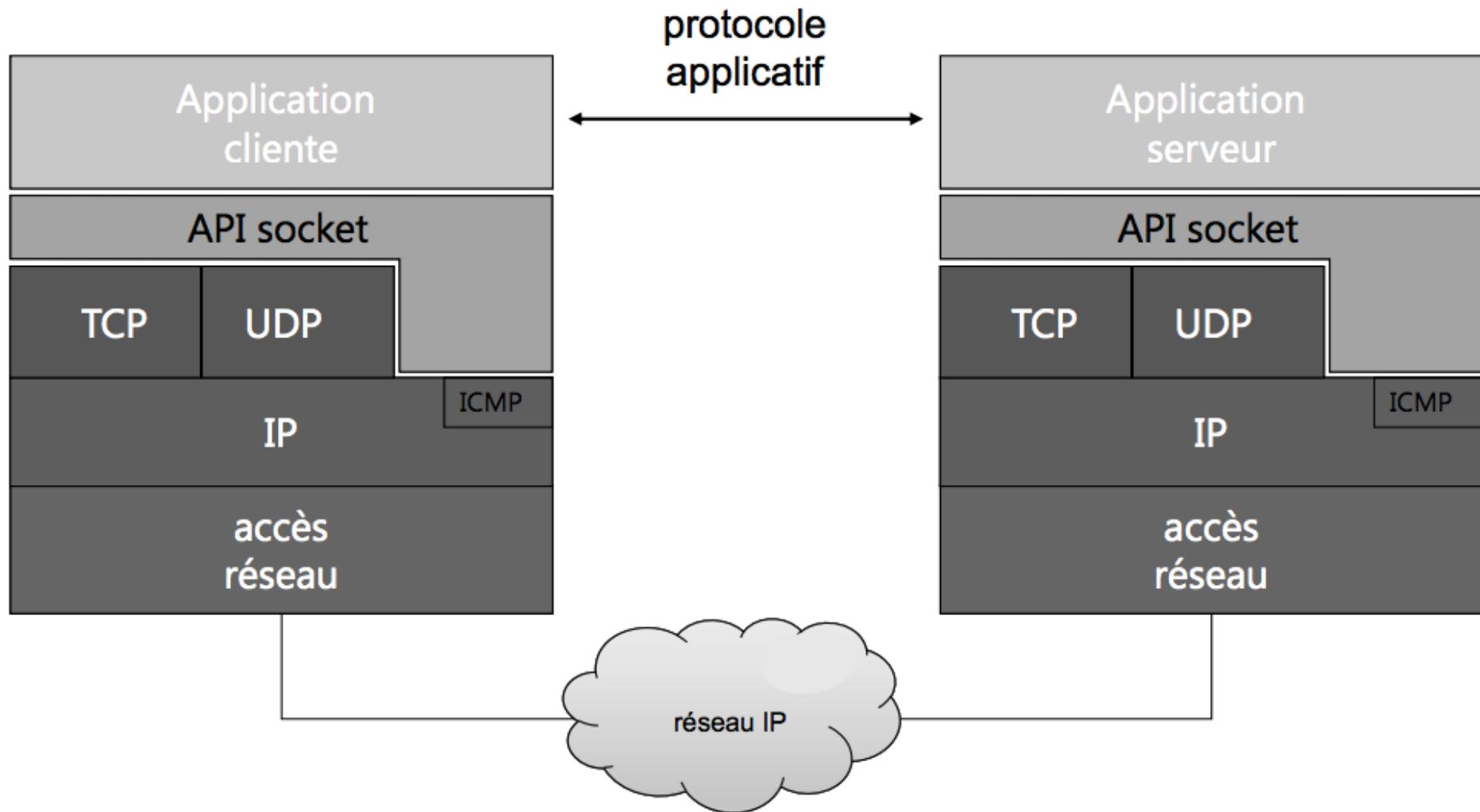
- **Mode non connecté : UDP (User Datagram Protocol)**

- Chaque requête est indépendante : il n'y a pas de liaison persistante.
- Le serveur ne garde pas d'état entre deux messages.
- Le client doit fournir son adresse à chaque envoi, car aucune connexion n'est maintenue.
- UDP ne garantit ni la fiabilité, ni l'ordre, ni le contrôle de flux.
- Il est toutefois adapté aux communications rapides et légères, comme lorsqu'une réponse peut tenir en un seul message.

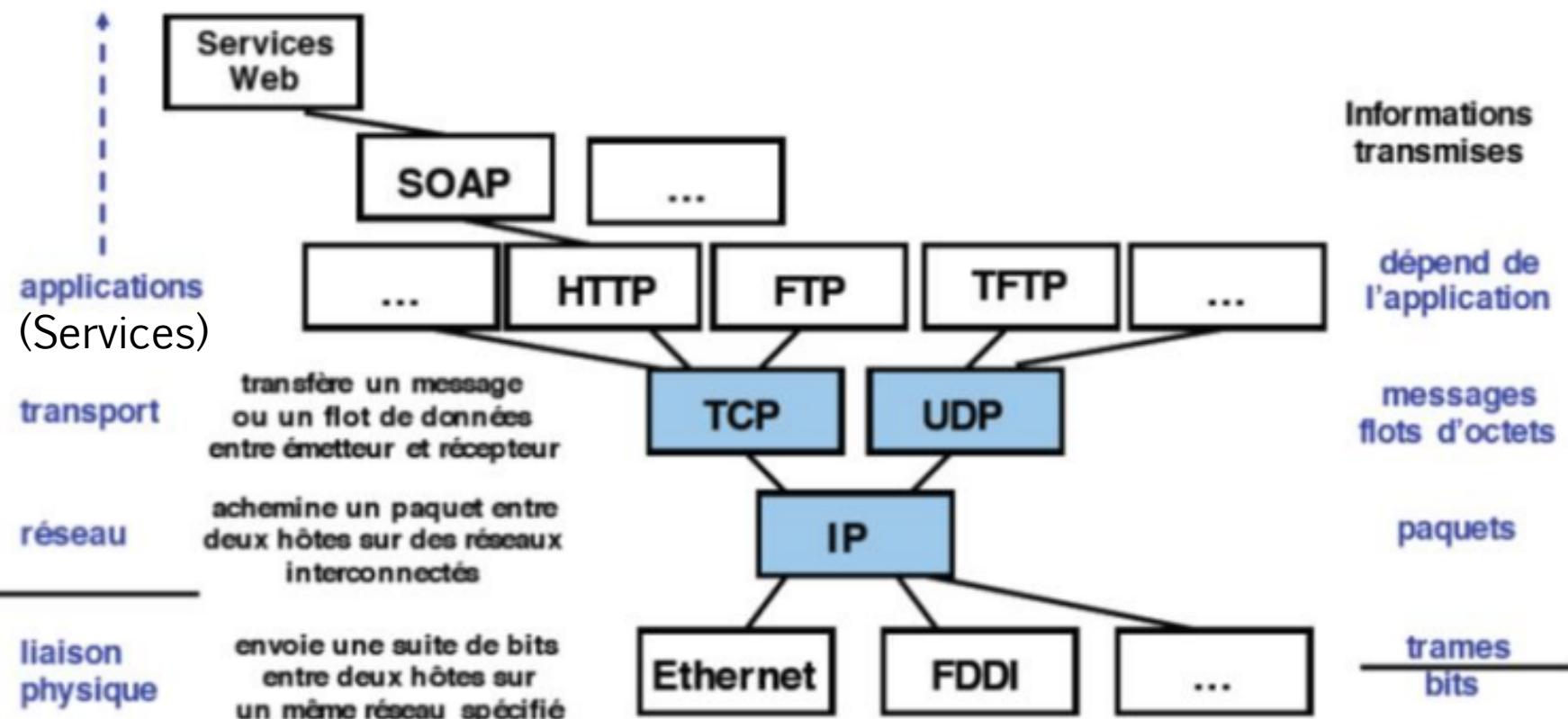
- **Points communs entre TCP et UDP**

- La communication est toujours initiée par le client ; le serveur attend passivement les requêtes.
- Le client doit connaître la référence du serveur (adresse IP + numéro de port). Cette information peut être obtenue :
 - via un annuaire de services,
 - ou par convention (ports standards réservés, comme 80 pour HTTP, 25 pour SMTP).
- Un même serveur peut traiter plusieurs clients simultanément, soit en utilisant un seul thread partagé, soit en créant un thread distinct par client.

L'API socket



L'API socket



HTTP : *HyperText Transfer Protocol* : protocole du Web

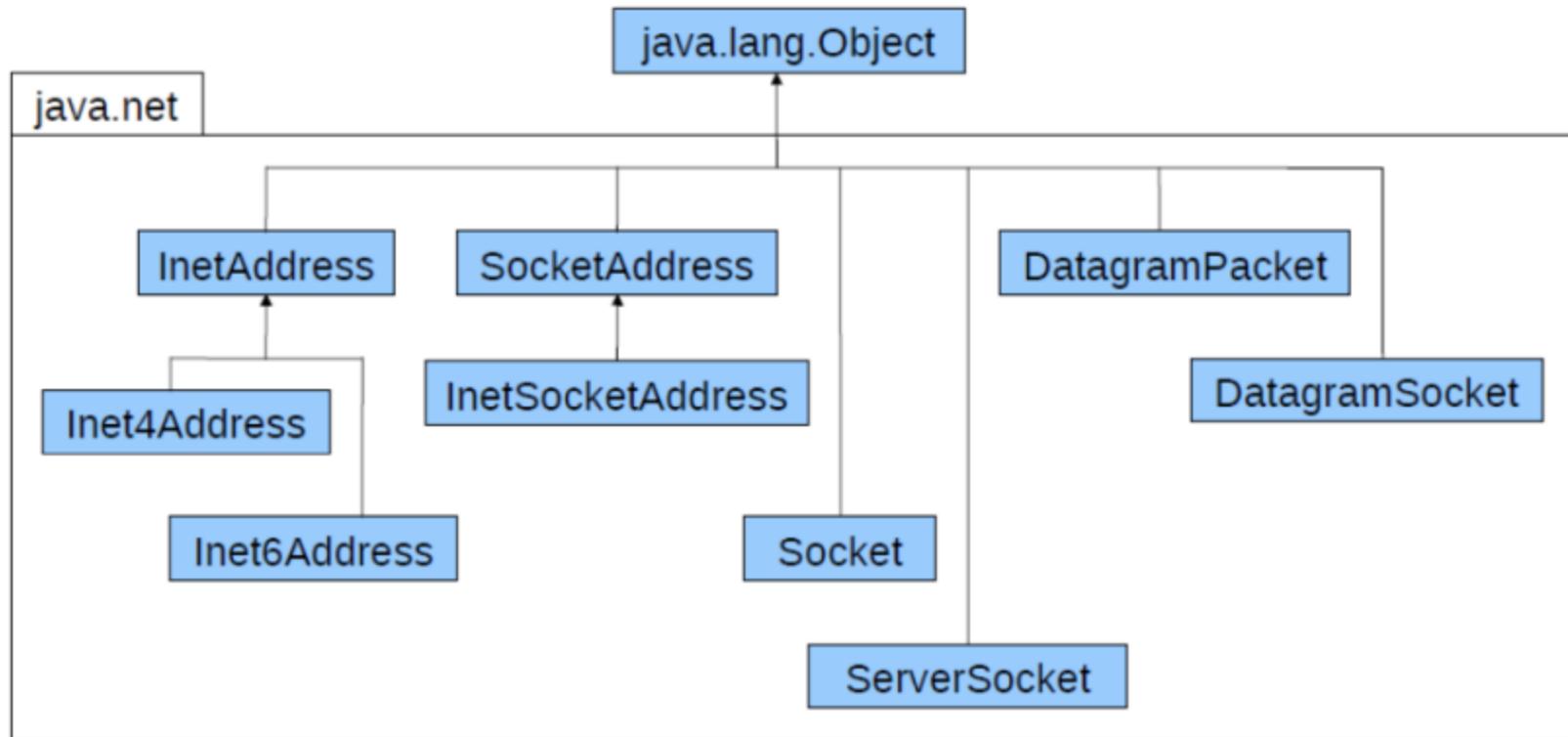
TFTP, FTP : *(Trivial) File Transfer Protocol* : transfert de fichiers

TCP : *Transmission Control Protocol* : transport en mode connecté

UDP : *User Datagram Protocol* : transport en mode non connecté

IP : *Internet Protocol* : Interconnexion de réseaux, routage

Package : java.net



<https://www.jmdoudoux.fr/java/dej/chap-net.htm#net-4>

CHAPITRE 1 : Socket

Principe de base d'une application C/S avec les sockets en mode connecté

Savoir exploiter le package [java.net](#) pour exploiter Flux de communication

<https://www.jmdoudoux.fr/java/dej/chap-net.htm#net-4>

N.B C/S : Client / Serveur
S.o.p = System.out.println

Principe de base d'une application client/serveur avec les sockets

Création d'un serveur

```
public static void main(String[] args){  
/* 1ère étape :  
création de l'objet ServerSocket  
ServerSocket socketServeur = new .....
```

Création d'un client

```
S.o.p("Le serveur attend la connexion d'un client ");
```

```
}// Fin de la méthode main
```

Principe de base d'une application client/serveur avec les sockets

Création d'un serveur

Création d'un client

```
public static void main(String[] args){  
/* 1ère étape :  
création de l'objet Socket côté client*/
```

```
Socket socket = new .....
```

```
S.o.p("je suis connecté");
```

```
}// Fin de la méthode main
```

Principe de base d'une application client/serveur avec les sockets

Création d'un serveur

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet ServerSocket
    ServerSocket socketServeur = .....;
```

S.o.p("Le serveur attend la connexion d'un client ");

/*La 2ème étape : attendre la connexion du client
création de l'objet Socket côté serveur
suite à la connexion du client */

Socket socket =

S.o.p("un client est connecté");

}// Fin de la méthode main

Création d'un client

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet Socket côté client*/
```

Socket socket = new Socket("localhost", 1234);

S.o.p("je suis connecté");

}// Fin de la méthode main

Principe de base d'une application client/serveur avec les sockets

Création d'un serveur

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet ServerSocket
    ServerSocket socketServeur = new ServerSocket(1234);
```

S.o.p("Le serveur attend la connexion d'un client ");

/*La 2ème étape : attendre la connexion du client
création de l'objet Socket côté serveur
suite à la connexion du client */

Socket socket =

S.o.p("un client est connecté");

/*La dernière étape : fermeture de la Socket */

socket.close();

}// Fin de la méthode main

Création d'un client

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet Socket côté client */
```

Socket socket = new Socket("localhost", 1234);

S.o.p("je suis connecté");

/*La dernière étape : fermeture de la Socket */

.....

}// Fin de la méthode main

Principe de base d'une application client/serveur avec les sockets

Création d'un serveur

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet ServerSocket
    */
    ServerSocket socketServeur = new ServerSocket(1234);
```

S.o.p("Le serveur attend la connexion d'un client ");

/*La 2ème étape : attendre la connexion du client
création de l'objet Socket côté serveur
suite à la connexion du client */

Socket socket = socketServeur.accept();

S.o.p("un client est connecté");

/*La dernière étape : fermeture de la Socket */

socket.close();

}// Fin de la méthode main

Création d'un client

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet Socket côté client */
    
```

Socket socket = new Socket("localhost", 1234);

S.o.p("je suis connecté");

/*La dernière étape : fermeture de la Socket */

socket.close();

}// Fin de la méthode main

Principe de base d'une application client/serveur avec les sockets

Création d'un serveur

```
public static void main(String[] args){
/* 1ère étape :
création de l'objet ServerSocket
ServerSocket socketServeur = new ServerSocket(1234);
```

S.o.p("Le serveur attend la connexion d'un client ");

/*La 2ème étape : attendre la connexion du client */

Socket socket = socketServeur.accept();

S.o.p("un client est connecté");

/*La dernière étape : fermeture de la Socket */

socket.close();

}// Fin de la méthode main

Création d'un client

```
public static void main(String[] args){
/* 1ère étape :
création de l'objet Socket côté client*/
```

Socket socket = new Socket("localhost", 1234);

S.o.p("je suis connecté");

/*La dernière étape : fermeture de la Socket */

socket.close();

}// Fin de la méthode main

Principe de base d'une application client/serveur avec les sockets

Création d'un serveur

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet ServerSocket
    */
    ServerSocket socketServeur = new ServerSocket(1234);
```

S.o.p("Le serveur attend la connexion d'un client ");

/*La 2ème étape : attendre la connexion du client */

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

S.o.p("un client est connecté");

```
    inputStream is = .....  
    int nb=.....
```

/*La dernière étape : fermeture de la Socket */

```
    socket.close();
```

}// Fin de la méthode main

Création d'un client

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet Socket côté client*/
    
```

```
    Socket socket = new Socket("localhost", 1234);
```

S.o.p("je suis connecté");

```
    OutputStream ..... = socket.getOutputStream();
    os.write( 10);
```

/*La dernière étape : fermeture de la Socket */

```
    socket.close();
```

}// Fin de la méthode main

Principe de base d'une application client/serveur avec les sockets

Création d'un serveur

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet ServerSocket
    */
    ServerSocket socketServeur = new ServerSocket(1234);
```

S.o.p("Le serveur attend la connexion d'un client ");

/*La 2ème étape : attendre la connexion du client */

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

S.o.p("un client est connecté");

```
    inputStream is = socket.getInputStream();
    int nb=is.read();
```

/*La dernière étape : fermeture de la Socket */

```
    socket.close();
```

}// Fin de la méthode main

Création d'un client

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet Socket côté client*/
    
```

```
    Socket socket = new Socket("localhost", 1234);
```

S.o.p("je suis connecté");

```
    OutputStream os = socket.getOutputStream();
    os.write( 10);
```

/*La dernière étape : fermeture de la Socket */

```
    socket.close();
```

}// Fin de la méthode main

Principe de base d'une application client/serveur avec les sockets

Création d'un serveur

```
public static void main(String[] args){
/* 1ère étape :
création de l'objet ServerSocket
ServerSocket socketServeur = new ServerSocket(1234);
```

```
S.o.p("Le serveur attend la connexion d'un client ");
```

```
/*La 2ème étape : attendre la connexion
du client */
```

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

```
S.o.p("un client est connecté");
```

```
InputStream is = socket.getInputStream();
int nb=is.read();
```

```
int x = nb*3;
```

```
/*La dernière étape : fermeture de la
Socket */
```

```
socket.close();
```

```
}// Fin de la méthode main
```

Création d'un client

```
public static void main(String[] args){
/* 1ère étape :
création de l'objet Socket côté client */
```

```
Socket socket = new Socket("localhost", 1234);
```

```
S.o.p("je suis connecté");
```

```
OutputStream os = socket.getOutputStream();
os.write( 10);
```

```
/*La dernière étape : fermeture de la
Socket */
```

```
socket.close();
```

```
}// Fin de la méthode main
```

Principe de base d'une application client/serveur avec les sockets

Création d'un serveur

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet ServerSocket
    */
    ServerSocket socketServeur = new ServerSocket(1234);
```

S.o.p("Le serveur attend la connexion d'un client ");

/*La 2ème étape : attendre la connexion du client */

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

S.o.p("un client est connecté");

```
    inputStream is = socket.getInputStream();
    int nb=is.read();
```

```
    int x = nb*3;
```

```
    OutputStream os =socket.getOutputStream();
    ....
```

/*La dernière étape : fermeture de la Socket */

```
    socket.close();
```

}// Fin de la méthode main

Création d'un client

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet Socket côté client*/
    
```

```
    Socket socket = new Socket("localhost", 1234);
```

S.o.p("je suis connecté");

```
    OutputStream os = socket.getOutputStream();
    os.write( 10);
```

.....
.....

/*La dernière étape : fermeture de la Socket */

```
    socket.close();
```

// Fin de la méthode main

Principe de base d'une application client/serveur avec les sockets

Création d'un serveur

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet ServerSocket
    */
    ServerSocket socketServeur = new ServerSocket(1234);
```

S.o.p("Le serveur attend la connexion d'un client ");

/*La 2ème étape : attendre la connexion du client */

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

S.o.p("un client est connecté");

```
    inputStream is = socket.getInputStream();
    int nb=is.read();
```

int x = nb*3;

```
    OutputStream os =socket.getOutputStream();
    os.write( x);
```

/*La dernière étape : fermeture de la Socket */

socket.close();

}// Fin de la méthode main

Création d'un client

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet Socket côté client*/
    
```

Socket socket = new Socket("localhost", 1234);

S.o.p("je suis connecté");

```
    OutputStream os = socket.getOutputStream();
    os.write( 10);
```

```
    inputStream is = socket.getInputStream();
    int nb=is.read();
```

/*La dernière étape : fermeture de la Socket */

socket.close();

}// Fin de la méthode main

Principe de base d'une application client/serveur avec les sockets

Création d'un serveur

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet ServerSocket
    */
    ServerSocket socketServeur = new ServerSocket(1234);
```

S.o.p("Le serveur attend la connexion d'un client ");

/*La 2ème étape : attendre la connexion du client */

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

S.o.p("un client est connecté");

```
    inputStream is = socket.getInputStream();
    int nb=is.read();
```

```
    int x = nb*2;
```

```
    OutputStream os =socket.getOutputStream();
    os.write( x);
```

/*La dernière étape : fermeture de la Socket */

```
    socket.close();
```

}// Fin de la méthode main

Création d'un client

```
public static void main(String[] args){
    /* 1ère étape :
    création de l'objet Socket côté client*/
    
```

```
    Socket socket = new Socket("localhost", 1234);
```

S.o.p("je suis connecté");

```
    OutputStream os = socket.getOutputStream();
    os.write( 10);
```

```
    inputStream is = socket.getInputStream();
    int nb=is.read();
```

S.o.p("la multiple de 10 est " + nb);

/*La dernière étape : fermeture de la Socket */

```
    socket.close();
```

// Fin de la méthode main

Exercice (2 min)

- » Complétez la figure suivante en se basant sur le scénario précédent
- » Précisez la différence entre les flux de communication et de traitement selon le scénario précédent

