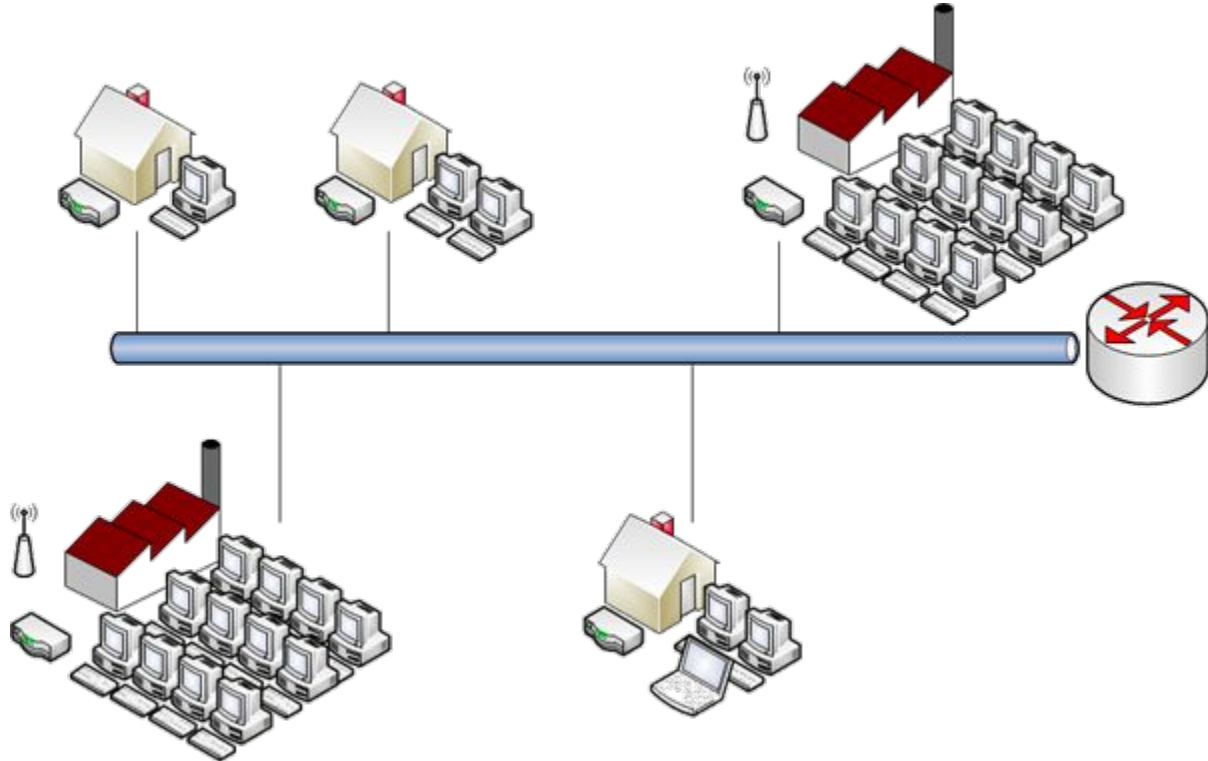




**JAVA-NET**



# Quelques notions de réseau



# Quelques notions de réseau

Ces machines arrivent à communiquer entre elles grâce à (au minimum):

- Une adresse IP ;
  - a. Adresse MAC
  - b. Adresse IP
- Un port libre et ouvert ;
  - a. Un port pour le navigateur: 80
  - b. Un port pour le client de messagerie: 25
  - c. Un port pour FTP: 22
  - d. ....
- Un protocole de communication commun.
  - a. TCP
  - b. UDP
  - c. ....

# Classes utiles: InetAddress

- permet de manipuler des adresses IP
- On doit passer par une méthode statique de cet class afin d'en récupérer une instance
  - ***InetAddress address = InetAddress.getLocalHost();***
  - ***InetAddress address =InetAddress.getByName(String nom\_de\_l\_machine)***
  - ***InetAddress address =InetAddress.getAllByName(String nom\_de\_l\_machine)***
- Si l'hôte sur lequel on souhaite faire une recherche est inconnu, l'objet ***InetAddress*** lèvera une exception de type ***UnknownHostException***.

# Classes utiles: InetAddress

```
6 public class Inet {
7     public static void main(String[] args) {
8         try {
9             InetAddress address = InetAddress.getLocalHost();
10            //address = InetAddress.getByAddress(new byte[]{(byte)192,(byte)168, 2, 44});
11            //address = InetAddress.getByAddress("localhost");
12            //address = InetAddress.getByAddress("127.0.0.1");
13            System.out.println("Nom : " + address.getHostName());
14            System.out.println("Adresse : " + address.getHostAddress());
15            System.out.println("Nom canonique : " + address.getCanonicalHostName());
16
17        } catch (UnknownHostException e) {
18            e.printStackTrace();
19        }
20    }
21 }
```

Problems @ Javadoc Declaration Console

<terminated> Inet [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Nov 2, 2018, 1:49:30 AM)

```
Nom : gnu-ThinkPad-T420s
Adresse : 127.0.1.1
Nom canonique : gnu-ThinkPad-T420s
```

# Classes utiles: InetAdress

## Exercice

Utiliser l'objet ***InetAdress*** pour récupérer, en fonction de ce que vous allez saisir:

1. soit l'adresse IP d'une adresse internet,
2. soit le nom de domaine d'une adresse IP V4.

```
Saisissez une adresse(IPv4 ou nom de domaine):
```

```
google.com
```

```
Voici le résultat trouvé :
```

```
172.217.21.78
```

```
Saisissez une adresse(IPv4 ou nom de domaine):
```

```
216.58.201.206
```

```
Voici le résultat trouvé :
```

```
ber01s09-in-f206.1e100.net
```

# Classes utiles: InetAddress

```
7 public class Dns {  
8     public static void main(String[] args) {  
9         Scanner sc = new Scanner(System.in);  
10        System.out.println("Saisissez une adresse(IPv4 ou nom de domaine): ");  
11        String hote = sc.nextLine();  
12        System.out.println("Voici le résultat trouvé : ");  
13        String result = "";  
14        try {  
15            if(hote.matches("[a-zA-Z\\.]+"))  
16                result = InetAddress.getByName(hote).getHostAddress();  
17            else  
18                result = InetAddress.getByName(hote).getHostName();  
19            System.out.println(result);  
20        } catch (UnknownHostException e) {e.printStackTrace();}  
21    }  
22 }  
23 }
```

# Classes utiles: URL

Une URL peut se décomposer comme suit:

http://www.

Le protocol

Le nom du serveur

L'emplacement dans le serveur



# Classes utiles: URL

```
7 public class Url {  
8     public static void main(String[] args) {  
9         try {  
10             URL url = new URL("http://www.este.ucam.ac.ma/");  
11             System.out.println("Authority : " + url.getAuthority());  
12             System.out.println("Default port : " + url.getDefaultPort());  
13             System.out.println("Host : " + url.getHost());  
14             System.out.println("Port : " + url.getPort());  
15             System.out.println("Protocol : " + url.getProtocol());  
16         } catch (MalformedURLException e) {e.printStackTrace();}  
17     }  
}
```

Problems @ Javadoc Declaration Console

<terminated> Url [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Nov 2, 2018, 2:00:41 AM)

Authority : www.este.ucam.ac.ma:80

Default port : 80

Host : www.este.ucam.ac.ma

Port : 80

Protocol : http

# Classes utiles: URL

## Exercice:

Ecrire une classe ***ReadNetworkFile*** qui lit le contenu d'un fichier sur le web en utilisant la méthode ***openStream()*** de la classe ***URL***

# Classes utiles: URL

```
9 public class ReadNetworkFile {
10     public static void main(String[] args) {
11
12         try {
13             URL url=new URL("http://www.google.com");
14             InputStream is=url.openStream();
15             BufferedInputStream bis=new BufferedInputStream(is);
16             StringBuilder sb=new StringBuilder();
17             int n;
18             while((n=bis.read())!=-1) sb.append((char)n);
19             System.out.println(sb);
20         } catch (MalformedURLException e) {e.printStackTrace();}
21         catch (IOException e) {e.printStackTrace();}
22     }
23 }
24
```

## URLConnection et HttpURLConnection

- Elles permettent d'établir une communication entre l'application et une URL.
- Elles permettent de récupérer le contenu d'une page web et aussi d'interagir directement avec celle-ci
- La classe **URLConnection** est une classe abstraite.
- La classe **HttpURLConnection** hérite de **URLConnection** permet de dialoguer avec des applications web via le protocole HTTP.

# URLConnection et HttpURLConnection

1. On utilise la méthode ***openConnection()*** de l'objet ***URL*** :
2. celle-ci nous retourne un objet de type ***URLConnection*** ;
3. Grâce à cet objet, on peut interagir avec notre ressource :
  - a. récupérer des données;
  - b. Envoyer des données ;
4. Nous allons récupérer les interactions grâce à la méthode ***getInputStream()*** de notre objet ***URLConnection***
5. Nous stockerons le résultat dans une chaîne de caractères

## URLConnection et HttpURLConnection

```
10 public class Browser {
11     public static void main(String[] args) {
12         try {
13             String siteWeb = "http://www.google.com";
14             URL url = new URL(siteWeb);
15             System.out.println("Authority : " + url.getAuthority());
16             System.out.println("Default port : " + url.getDefaultPort());
17             System.out.println("Host : " + url.getHost());
18             System.out.println("Port : " + url.getPort());
19             System.out.println("Protocol : " + url.getProtocol());
20             try {
21                 URLConnection urlConn = url.openConnection();
22                 System.out.println(urlConn.getContentType());
23                 String content = "", line = null;
24                 BufferedReader buf = new BufferedReader(
25                     new InputStreamReader(urlConn.getInputStream()));
26                 while((line = buf.readLine()) != null) content += line + "\n";
27                 System.out.println(content);
28             } catch (IOException e) {e.printStackTrace();}
29         } catch (MalformedURLException e) {e.printStackTrace();}
30     }
```

# Les sockets

- Les données transmises sur le réseau sont structurées et encapsulées dans ce qu'on appelle des **paquets**, ou **datagrammes**.
- Chaque **datagramme** contient une section **en-tête** et un **corps**.
- **L'en-tête** contient toutes les informations nécessaires au **transport** de l'information,
- Le **corps** contient les **données** en transit.
- Après transmission ces paquets sont rassemblés de nouveau.

# Les sockets

Les sockets s'occupe de ce découpage en paquets. Elles permettent de traiter n'importe quelle communication réseau comme une I/O vers un fichier. Les sockets permettent aussi de:

1. Se connecter à une machine distante ;
2. Recevoir et envoyer des données ;
3. Fermer une connexion établie ;
4. Attendre une connexion de l'extérieur ;
5. Écouter les communications entrantes ;
6. Utiliser un port.

Les communication entres sockets sont dites en **full-duplex**, c'est-à-dire que le client et le serveur peuvent envoyer et recevoir des signaux simultanément



# Les sockets

Une socket est un moyen de partage des données en réseau et elle se manipule comme un fichier.

En java il y a deux types de sockets :

- **Socket côté client** : permet de se connecter à une machine distante afin de communiquer avec elle ;
- **Socket côté serveur** : connexion qui attend qu'un client vienne se connecter afin de communiquer avec lui.

# Sockets côté client:

L'utilisation d'une socket côté client passe par :

1. Instancier la socket ;
2. Connecter la socket avec le service côté serveur ciblé (FTP, HTTP...) ;
3. Échanger les données avec la socket serveur (dépend du protocole utilisé : on n'utilise pas un serveur FTP comme un serveur HTTP) ;
4. Fermer la socket lorsque les communications sont terminées.

# Sockets côté client:

- La classe ***java.net.Socket*** est l'élément fondamental de toute connexion réseau côté client.
- Elle utilise par défaut le protocole TCP
- Constructeurs:
  - ***Socket(InetAddress ad, int port);***
  - ***Socket(String nom\_hote, int port);***
- Peut générer des exception en raison de :
  - hôte n'accepte pas les connexions ;
  - hôte n'accepte pas les connexions sur le port demandé ;
  - Un problème de réseau ;
  - hôte inconnue;

# Sockets côté client:

```
2
3 import java.io.IOException;
4 import java.net.InetAddress;
5 import java.net.Socket;
6 import java.net.UnknownHostException;
7
8 public class TestSocket {
9
10     public static void main(String[] args) {
11         try {
12             Socket soc = new Socket(InetAddress.getByName("www.google.com"), 80);
13             //Socket soc = new Socket("www.google.com", 80);
14         } catch (UnknownHostException e) {e.printStackTrace();}
15         catch (IOException e) {e.printStackTrace();}
16     }
17 }
18
```

# Sockets côté client

## *Remarque:*

Il faut toujours fermer une socket après utilisation sinon le port ne sera pas accessible par d'autres applications.

```
finally{
    if(soc != null){
        try {
            soc.close();
        } catch (IOException e) {
            e.printStackTrace();
            soc = null;
        }
    }
}
```

# Sockets côté client:

## Exercice

Dans un ordinateur existe 65 536 port de connexion, mais certains sont déjà réservés. A l'aide la classe ***java.net.Socket*** lister les ports ouverts et qui accepte les connexions sur un hôte.

# Sockets côté client

```
7 public class FreePort {
8     public static void main(String[] args) {
9         System.out.println("Liste des port ouvert:");
10        Socket soc=null;
11        for(int i = 1; i <= 65535; i++){
12            try {
13                soc = new Socket("127.0.0.1", i);
14                System.out.println(i);
15            } catch (UnknownHostException e) {e.printStackTrace();}
16            catch (IOException e) {}
17            finally{
18                if(soc != null){
19                    try {
20                        soc.close();
21                    } catch (IOException e) {
22                        e.printStackTrace();
23                        soc = null;
24                    }
25                }
26            }
27        }
```

# Sockets côté client

## *Exercice*

Lire le contenu d'un fichier sur le web en utilisant une **Socket** sachant que cette classe possède une méthode ***getInputStream()***



# Sockets côté client

```
9 public class ReadNetworkFileSock {
10     public static void main(String[] args){
11         Socket soc=null;
12         try{
13             soc = new Socket("www.este.ucam.ac.ma", 80);
14             String req = "GET / HTTP/1.1\r\n";
15             req += "Host: www.este.ucam.ac.ma\r\n";
16             req += "\r\n";
17             BufferedOutputStream bos = new BufferedOutputStream(soc.getOutputStream());
18             bos.write(req.getBytes());
19             bos.flush();
20             BufferedInputStream bis = new BufferedInputStream(soc.getInputStream());
21             StringBuilder sb=new StringBuilder();
22             int n;
23             while((n = bis.read()) != -1){
24                 sb.append((char) n);
25             }
26             System.out.println(sb);
27         } catch (UnknownHostException e) {
28             e.printStackTrace();
29         } catch (IOException e) {
30             e.printStackTrace();
31         }
32     }
```

# Sockets côté Serveur

Java met à disposition une classe spéciale permettant de coder des applications serveurs, la classe ***ServerSocket***

Cette classe possède:

- Un constructeur permettant de spécifier le ***port*** d'écoute du serveur
- Une méthode qui permet d'attendre une connexion cliente: ***accept()***

# Sockets côté Serveur

Lors du travail avec des sockets serveurs généralement on respecte les étapes suivantes:

1. Créer une `ServerSocket` sur le port spécifié ;
2. Cette socket attend une connexion cliente (grâce à sa méthode **`accept()`**) ;
3. depuis la connexion cliente, invoquer les méthodes **`getInputStream()`** ou/et **`getOutputStream()`** afin de communiquer avec votre serveur ;
4. Le client et le serveur interagissent ensemble;
5. Le client et/ou le serveur mettent fin à la connexion en cours ;
6. La socket serveur retourne à l'étape 2.

# Sockets côté Serveur

Lors du travail avec des sockets serveurs généralement on respecte les étapes suivantes:

1. Créer une `ServerSocket` sur le port spécifié ;
2. Cette socket attend une connexion cliente (grâce à sa méthode **`accept()`**) ;
3. depuis la connexion cliente, invoquer les méthodes **`getInputStream()`** ou/et **`getOutputStream()`** afin de communiquer avec votre serveur ;
4. Le client et le serveur interagissent ensemble;
5. Le client et/ou le serveur mettent fin à la connexion en cours ;
6. La socket serveur retourne à l'étape 2.

# Sockets côté Serveur


Ports libre dans un serveur en utilisant les *ServerSocket*

```
6 public class ServerSocketTest {  
7     public static void main(String[] args) {  
8         for(int port = 1; port <= 65535; port++){  
9             try {  
10                 ServerSocket sSoc = new ServerSocket(port);  
11                 System.out.println("Le port " + port + " est libre ");  
12             } catch (IOException e) {  
13                 //port déjà utilisé  
14             }  
15         }  
16     }  
17 }
```


# Sockets côté Serveur

Le serveur écoute sur le port 4040 et reste bloqué (***accept()***) jusqu'à la réception d'une connexion de la part d'un client

```
public class Serveur {  
    public static void main(String[] args) {  
        ServerSocket sSoc=null ;  
        Socket soc=null ;  
        try {  
            sSoc = new ServerSocket(4040);  
            soc = sSoc.accept();  
            System.out.println("Un client s'est connecté !");  
            sSoc.close();  
            soc.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```



```
public class Client {  
    public static void main(String[] args) {  
        Socket soc=null;  
        try {  
            soc = new Socket(InetAddress.getLocalHost(),4040);  
            soc.close();  
        } catch (UnknownHostException e) {e.printStackTrace();}  
        catch (IOException e) {e.printStackTrace();}  
    }  
}
```



# Sockets côté Serveur

La méthode ***accept()*** bloque le programme

=> on ne peut communiquer qu'avec un seul client à la fois.

=> utilisation des Threads

