Application avec interface graphique

Adresse IPV4

- Une @IP sert à **identifier** d'une manière unique une machine connectée sur un réseau (privé ou internet)
- Une @IP **Version 4** tient sur **32 bits** (4 octets).
- Notation **décimale** (4 décimaux séparés par des points).
- Chaque décimale prend donc une valeur entre 0- 255.

Adresse IPV4

Exemple

Adresse IP en binaire :

11000001 00011011 00101101 00100001

En **notation décimale pointée**:

193.27.45.33

Package java.net

Provides the classes for implementing **networking applications**.

The java.net package can be roughly divided in two sections:

- A Low Level API, which deals with abstractions like **IP addresses** and **Sockets**, …
- A High Level API, which deals with abstractions like **URIs** (Universal Resource Identifiers) and **URLs** (Universal Resource Locators),…

Quelques méthodes de la classe InetAddress

The InetAddress class is the abstraction representing an IP (Internet Protocol) address.

Modifier and Type	Method and Description
`String`	**`getHostAddress()`** Returns the IP address string in textual presentation.
`String`	**`getHostName()`** Gets the host name for this IP address.
`static InetAddress`	**`getLocalHost()`** Returns the address of the local host.

Afficher mon adresse ip

```
import java.net.*;
```

```
137     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
138
            InetAddress a=InetAddress.getLocalHost();
140     }
```

Ajouter le traitement d'exception

Afficher mon adresse ip

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {


    try {
        InetAddress a=InetAddress.getLocalHost();
        System.out.println(a);
    } catch (UnknownHostException ex) {
        Logger.getLogger(adrs.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Afficher mon adresse ip

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {


        try {
            InetAddress a=InetAddress.getLocalHost();
            System.out.println(a);
            System.out.println(a.getHostName());
            System.out.println(a.getHostAddress());
        } catch (UnknownHostException ex) {
            Logger.getLogger(adrs.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

Run

Quelques méthodes de la classe String

Modifier and Type	Method and Description
char	**charAt**(int index) Returns the char value at the specified index.
int	**length**() Returns the length of this string.
boolean	**equalsIgnoreCase**(**String** anotherString) Compares this String to another String, ignoring case considerations.
boolean	**startsWith**(**String** prefix) Tests if this string starts with the specified prefix.


```
String ip= jTextField1.getText();
if (ip.equalsIgnoreCase(a.getHostAddress())){
                jTextField2.setText("oui");
}else       {jTextField2.setText("Non");}
```

Classes d’ Adresse IP

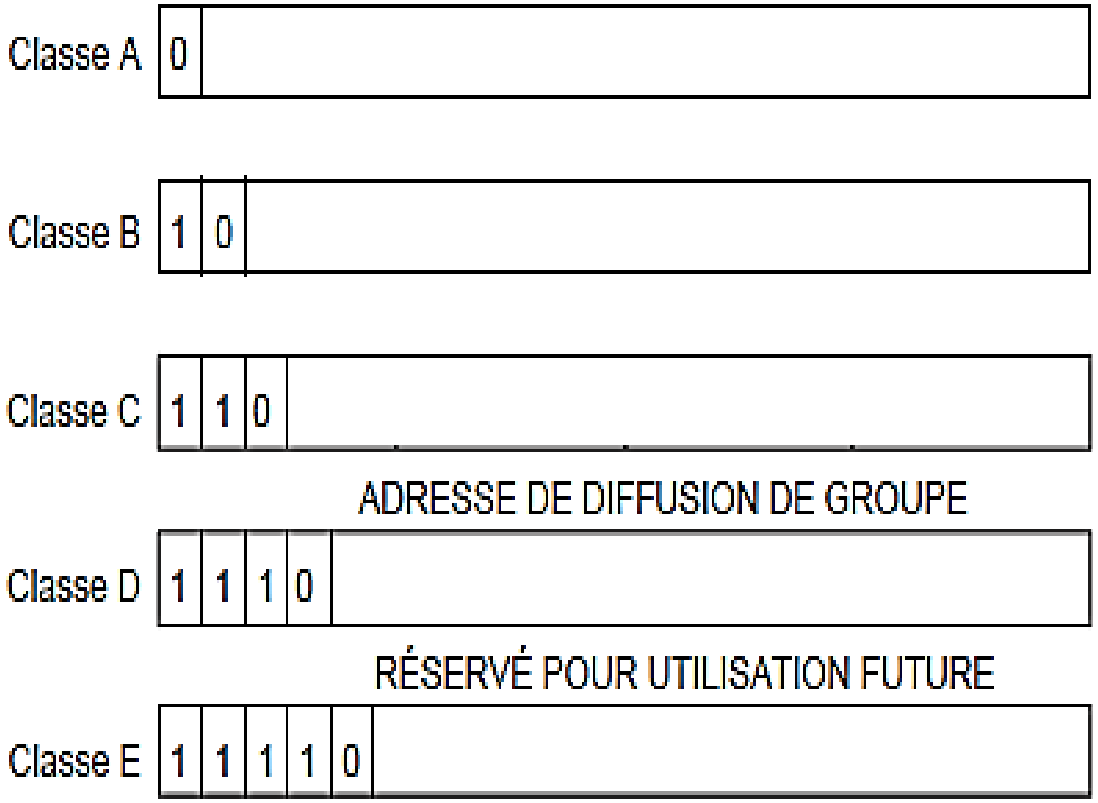
Comme une @IPV4 tient sur 32 bits, alors on peut avoir : 2 ^32 @IP différentes.

0.0.0.0

0.0.0.1

255.255.255.254

255.255.255.255



Classe Réseau	Début	Fin
Classe A	0.0.0.0	127.255.255.255
Classe B	128.0.0.0	191.255.255.255
Classe C	192.0.0.0	223.255.255.255
Classe D	224.0.0.0	239.255.255.255
Classe E	240.0.0.0	255.255.255.255

Classe	Plage d'adresse IP privée
A	10.0.0.0 - 10.255.255.255
B	172.16.0.0 - 172.31.255.255
C	192.168.0.0 - 192.168.255.255

Classe A

```
String  oct1=""          ;
int i=0;
while (ip.charAt(i)!='.'){
    oct1=oct1+ip.charAt(i);
    i++;
}

int c=Integer.parseInt(oct1);

if (c<=127){
    jTextField3.setText("A");
    if (c==10){jTextField4.setText("OUI");}
    else      {jTextField4.setText("Non");}
}
```

Ou bien....

```
if (c<=127){
    jTextField3.setText("A");
    if (ip.startsWith("10.")){jTextField4.setText("OUI");}
    else      {jTextField4.setText("Non");}
}
```

Travail demandé (3 points)

1. Classe B, C, D , E
2. Tester si l'@Ip en entrée est valide avant de lancer les différents traitements

Valide (entre 0.0.0.0 et 255.255.255.255)