TD1 – AI16 : INITIATION À LA PROGRAMMATION JAVA ET SOCKET

Dr. Ahmed Lounis (lounisah@utc.fr)

Premier programme JAVA

- Un programme JAVA est une classe qui doit être enregistrée dans un fichier de même nom et d'extension java

```
public class Prog1 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here

        System.out.println("bonjour tous le monde .....");

    }

}
```

Exercice N°0 : initiation

Ecrivez un programme java où vous:

- Déclarez un tableau de 10 nombres et vous l'affichez sur la console.
- Ecrivez des méthodes qui calculent le minimum, le maximum, la moyenne et l'écart type des valeurs de ce tableau.
- Testez ces méthodes.

Information:

Écart-type (S) = Racine carrée de la variance

Variance, (S^2) = moyenne de l'écart au carré de valeurs par rapport à la moyenne

Notion de classe et objets

- Un objet modélise toute entité identifiable, concrète ou abstraite, manipulée par l'application logicielle.

Exemple: ville, véhicule, personne.

- Un objet peut être vu comme une entité cohérente rassemblant des données et du code travaillant sur ses données.

- Une classe peut être considérée comme un moule à partir duquel on peut créer des objets.

Notion de classe et objets

- Exemple de class:

```
public class Compte {

    public String titulaire;
    public float solde;

    public Compte(String titulaire, int solde) {
        this.titulaire=titulaire;
        this.solde=solde;
    }
public void Afficher_info(){
    System.out.println("le titulaire de ce compte est "+titulaire+", il a "+solde+" dans son compte");
}
}
```

- Instanciation et accès aux méthodes:

```
Compte p=new Compte("jean mark", 50);

p.Afficher_info();
```

Modificateurs d'accès

Modificateur d'accès	Dans la classe	Dans une sous-classe	Ailleurs
private	OK		
protected	OK	OK	
public	OK	OK	OK

POO : Encapsulation


```java
public class Compte {
    private String titulaire;
    private float solde;

    public Compte(String titulaire, int solde) {
        this.titulaire=titulaire;
        this.solde=solde;
    }
    public float getSolde() {
        return solde;
    }
    public String getTitulaire() {
        return titulaire;
    }
    public void setSolde(float solde) {
        this.solde = solde;
    }
    public void setTitulaire(String titulaire) {
        this.titulaire = titulaire;
    }
    public String toString(){
       return "le titulaire de ce compte est "+titulaire+", il a "+solde+" dans son compte";
    }
}
```

Accesseur

Mutateur

Exercice N°1 : déclaration des classes et des objets

Définir une classe java "Point2D" qui caractérise un point dans un plan à deux dimensions :

1. Déclarer les attributs, les constructeurs, les accesseurs et les mutateurs définissant un point à deux dimensions
2. Ecrire un programme principal qui permet d'instancier un tableau d'objets de type "Point2D " et tester les méthodes de ces objets
3. Ajouter une méthode "calculerDistance" permettant de calculer la distance entre deux objets de type "Point2D"
4. Tester la méthode "calculerDistance" dans le programme principal

POO : l'héritage

- le concept de **l'héritage** permet d'hériter dans la déclaration d'une nouvelle classe (classe fille ou sous-classe) des caractéristiques (attributs et méthodes) déclarées auparavant dans une autre classe (appelée classe mère ou super-classe).

```
public class CompteBnp extends Compte{
    private String typeCompte;
    private float decouverte;

    public CompteBnp(String titulaire, int solde,String typeCompte,float decouverte) {
        super(titulaire, solde);
        this.typeCompte=typeCompte;
        this.decouverte=decouverte;
    }

    public float getDecouverte() {
        return decouverte;
    }

    public String getTypeCompte() {
        return typeCompte;
    }
```

POO : polymorphisme

- le concept de **polymorphisme** consiste à redéfinir les méthodes héritées de la classe mère.

```
public class CompteBnp extends Compte{
    private String typeCompte;
    private float decouverte;

    public CompteBnp(String titulaire, int solde,String typeCompte,float decouverte) {
        super(titulaire, solde);
        this.typeCompte=typeCompte;
        this.decouverte=decouverte;
    }

    public float getDecouverte() {
        return decouverte;
    }

    public String getTypeCompte() {
        return typeCompte;
    }

    @Override
    public String toString() {
        return "ce compte est un compte "+ typeCompte+", "+ super.toString()+", la est découverte de "+decouverte +" euro";
    }
}
```

Redéfinition de la méthode héritée de la class mère

Exercice N°2 : utilisation héritage et polymorphisme

Définir une classe java "Point3D" qui hérite de la class "Point2D" , caractérisant un point dans un plan à trois dimensions :

1. Déclarer les attributs, les constructeurs, les accesseurs et les mutateurs manquants
2. Redéfinir la méthode "toString" pour les objets de type "Point2D" et "Point3D".
3. Ecrire un programme principal qui permet d'instancier un tableau d'objets de type "Point3D" et tester les méthodes de ces objets
4. Redéfinir la méthode "calculerDistance" permettant de calculer la distance entre deux objets de type "Point3D"
5. Tester la méthode "calculerDistance" dans le programme principal

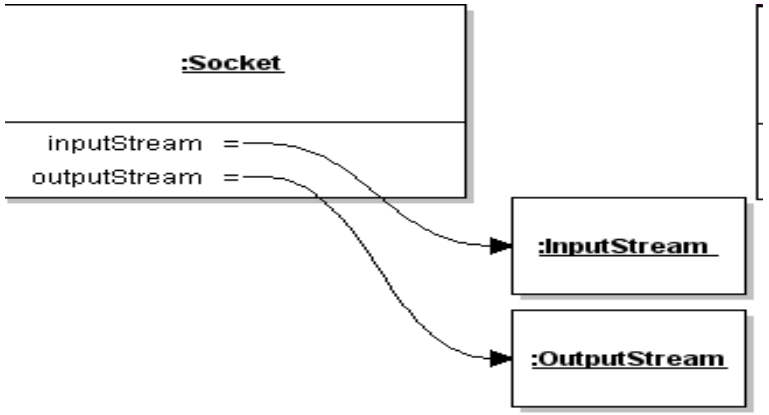
Socket réseaux

- **Définition :** un socket réseaux est un modèle permettant **la communication et la synchronisation** interprocessus afin de permettre à divers processus de communiquer aussi bien sur une même machine qu'à travers un réseau.
- **Deux modes de communication :**
  - **Mode connecté (comparable à une communication téléphonique),** utilisant le protocole TCP. Dans ce mode de communication, une connexion durable est établie entre les deux processus, de telle façon que l'adresse de destination n'est pas nécessaire à chaque envoi de données.
  - **Mode non connecté (analogue à une communication par courrier),** utilisant le protocole UDP. Ce mode nécessite l'adresse de destination à chaque envoi, et aucun accusé de réception n'est donné.

Communication par socket en JAVA

- La communication nécessite 2 sockets : 1 pour chacun des 2 programmes communicants via le réseau.
  - **1 socket pour le client.**
  - **1 socket pour le serveur.**

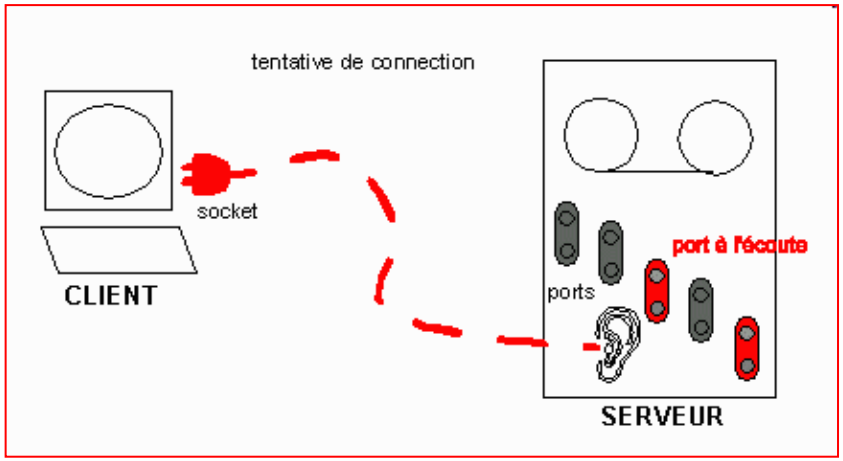
En Java, chaque instance de la classe Socket est associé à un objet de la classe **InputStream** (pour **lire** sur le socket) et à un objet de la classe **OutputStream** (pour **écrire** sur le socket).

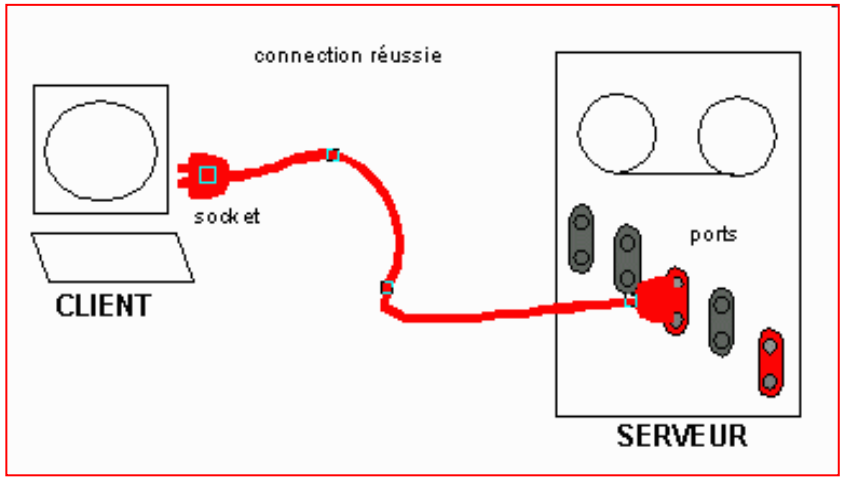


Class ServerSocket

Implémente les sockets de connexion du côté du serveur

1. **ServerSocket (numeroPort)** : crée un objet ServerSocket sur ce numéro de port.





2. **accept()** : attend une connexion d'une machine cliente. Une demande de connexion acceptée crée un socket connectant le client et le serveur.

Côté du serveur

- Le serveur utilisera deux types de sockets.
  - **Le premier, appelé socket de connexion sert à attendre un client.**
    - En Java, créer un socket de connexion peut se faire simplement en instanciant un objet de la classe **ServerSocket** du package **java.net**.

```
import java.net
…
ServerSocket conn = new ServerSocket(10080);
```

  - **Le second, appelé socket de communication sert à dialoguer avec le client.**
    - Une fois le socket de connexion créé, il suffit de lui demander d'attendre un client et de récupérer le socket de communication qui permettra de dialoguer avec le client.

```
Socket comm = conn.accept();
```

Quelques méthodes

- **ServerSocket (numeroPort, int backlog) :** crée un objet ServerSocket sur ce numéro de port avec une queue d'attente de connexion de taille spécifiée par l'entier backlog.
  - Par défaut, la taille est 50.
  - Les demandes de connections, quand la queue est pleine, sont rejetées et provoque une exception du coté du client.

- **accept() :** attend une connexion d'une machine cliente.

- **close() :** ferme le ServerSocket et toutes les sockets en cours obtenus par sa méthode accept.

- **isClosed() :** indique si le socket est fermé.

- **getLocalPort() :** retourne le numéro de port local.

- **InetAddress getInetAddress() :** retourne l'adresse du serveur.

Côté du client

- Contrairement au serveur, le client n'utilise qu'un seul socket : **le socket de communication.**

- Connexion au serveur et obtention d'un socket de communication.

```
Socket comm = new Socket ("localhost", 10080);
```

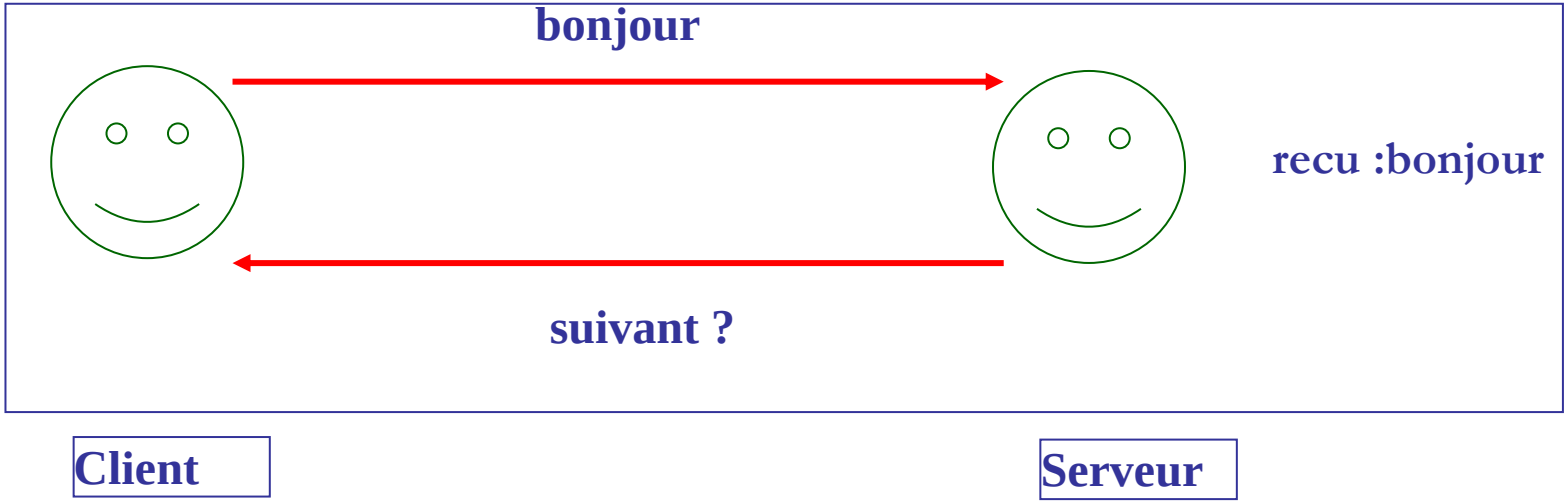
- On peut ensuite communiquer avec le serveur en utilisant les flux d'entrée et de sortie associés au socket.

Class Socket

- **Parmi les constructeurs :**
  - Socket (String host, int port)
  - Socket (InetAddress address, int port)
    - Utilise l'adresse IP : InetAdress
- **Quelques méthodes :**
  - close() : ferme proprement le socket est susceptible de lever une exception UnknownHostException.
  - getOutputStream() : retourne un flux de sortie OutputStream pour le socket.
  - getInputStream() : retourne un flux d'entrée InputStream pour le socket.
  - getPort() : retourne le numéro de port distant auquel le socket est connecté.
  - InetAddress getInetAddress() : retourne l'adresse de la machine distante.
    - **InetAddress** représente une adresse IP et éventuellement le nom de la machine.
      - **Quelques méthodes :**
        - getHostName() : retourne le nom d'hôte.
        - getHostAdress() : retourne l'IP sous forme de String.

Exemple de communication par socket

- **Une boucle :**
  - Le client envoie un message au serveur.
  - Le serveur confirme la réception et demande le mot suivant.
  - Pour se déconnecter, le client envoie le caractère « q ».



Exercice N°3 : connexion entre client/serveur

- Définir un programme Serveur.java qui permet d'initier une connexion en utilisant des sockets TCP.
  - Le programme exécute une boucle infini et affiche un message à chaque fois qu'une connexion est établie avec un client.

- Définir un programme Client.java qui permet de se connecter via des sockets TCP à Serveur.java.
  - Le programme client.java affiche à le message ''connexion réussite ….'' une fois que les deux programmes établissent une connexion entre eux.

1. Créer deux projets java (un pour le client et l'autre pour le serveur)
2. Vous pouvez tester le serveur de votre voisin avec l'adresse ip de sa machine (utiliser la commande ip sous linux). Pour éviter que votre programme socket soit bloqué au niveau réseau, choisissez un port avec une grande valeur (ex. 20000)

Lecture/écriture bas niveau

- **Ecriture directe sur l'objet socketv:**

```
Socket client=new Socket("localhost", 7000);
System.out.println("connecté ....");

OutputStream out=client.getOutputStream();
out.write("bonjour serveur".getBytes());
```

- **Lecture directe à partir de l'objet socket :**

```
InputStream in=client.getInputStream();
byte []b=new byte[1024];
in.read(b);
System.out.println("le Serveur à dit: "+new String(b));
```

Une opération de lecture sur ces flots est bloquante tant que des données ne sont pas disponibles. Cependant, il est possible de fixer une délai de garde à l'attente de données : *public void setSoTimeout(int timeout) throws SocketException*

Exercice N°4 : communication simple (envoi d'un message)

- Réaliser un envoi d'un message à partir du programme Client.java en utilisant les méthodes fournies par la class OutputStream.
- Modifier la programme Serveur.java afin de permettre la réception du message envoyé par le client et de le visualiser sur la console.
- Modifier le programme Serveur.java afin d'envoyer une réponse au programme client.java
- Visualiser la réponse du serveur dans le programme client.java

Code Java de l'exemple communication par socket (Page 19)

```
public class ServeurSocket {
   public static void main(String[] args) {
      try {
         ServerSocket conn = new ServerSocket(10080);
         Socket comm = conn.accept();
         OutputStream out = comm.getOutputStream();
         InputStream in = comm.getInputStream();
         byte b[]=new byte[20];
         String chaine;

         do{
            in.read(b);
            chaine=new String(b);
            System.out.println("reçu : "+chaine);
            out.write(("suivant ?").getBytes());
            b=new byte[20];

         }while(!chaine.startsWith("END"));
         System.out.println("Fin");
         in.close();
         out.close();
         comm.close();
      } catch (IOException ex) {
         Logger.getLogger(ServerSocket.class.getName()).log(Level.SEVERE, null, ex);
      }
   }
}
```

```
public class ClientSocket {

public static void main(String[] args) {

      int compteur=0,nbMax;

      byte b[]=new byte[20];

      try {

         Socket client = new Socket ("localhost", 10080);

         OutputStream out = client.getOutputStream();

         InputStream in=client.getInputStream();

         out.write("bonjour4".getBytes());

         System.out.println("Donnez le nombre max de messages à envoyer au serveur :");

         Scanner sc = new Scanner(System.in);

         nbMax = sc.nextInt();

         while(compteur!=nbMax){

            in.read(b);

            System.out.println("le serveur a dit: "+new String(b));

            System.out.println("Le client envoie le message:message"+compteur);

            out.write(("message"+compteur).getBytes());

            compteur++;

            b=new byte[20];

         }

         out.write(("END").getBytes());

        try {

           Thread.sleep(20);

        } catch (InterruptedException ex) {

           Logger.getLogger(ClientSocket.class.getName()).log(Level.SEVERE, null, ex);

        }

         in.close();

         out.close();

         client.close();

      } catch (IOException ex) {

         Logger.getLogger(ClientSocket.class.getName()).log(Level.SEVERE, null, ex);

      }

      } }
```

Exercice N°5 : communication alternative entre le serveur et le client

Questions

1. Testez-le code de la page précédente
2. Modifier le comportement du client pour qu'il s'arrête avant le serveur. Que se passe-t-il côté serveur ? Faites le test inverse : arrêt prématuré du serveur.

Lecture/écriture haut niveau

- Passage des types primitifs en écriture:

```
DataOutputStream outs=new DataOutputStream(client.getOutputStream());
outs.writeUTF("bonjour serveur .....");
```

- Réception des types primitifs en lecture:

```
DataInputStream ins=new DataInputStream(client.getInputStream());
System.out.println("le serveur à dit: "+ins.readUTF());
```

Lecture/écriture haut niveau

- Méthodes de la class DataInputStream:

Methods

Modifier and Type	Method and Description
int	**read**(byte[] b) Reads some number of bytes from the contained input stream and stores them into the buffer array b.
int	**read**(byte[] b, int off, int len) Reads up to len bytes of data from the contained input stream into an array of bytes.
boolean	**readBoolean**() See the general contract of the readBoolean method of DataInput.
byte	**readByte**() See the general contract of the readByte method of DataInput.
char	**readChar**() See the general contract of the readChar method of DataInput.
double	**readDouble**() See the general contract of the readDouble method of DataInput.
float	**readFloat**() See the general contract of the readFloat method of DataInput.
void	**readFully**(byte[] b) See the general contract of the readFully method of DataInput.
void	**readFully**(byte[] b, int off, int len) See the general contract of the readFully method of DataInput.
int	**readInt**() See the general contract of the readInt method of DataInput.

Lecture/écriture haut niveau

- Méthodes de la class DataInputStream

`String`	**`readLine`**`()` **Deprecated.** *This method does not properly convert bytes to characters. As of JDK 1.1, the preferred way to read lines of text is via the `BufferedReader.readLine()` method. Programs that use the `DataInputStream` class to read lines can be converted to use the `BufferedReader` class by replacing code of the form:* `DataInputStream d = new DataInputStream(in);` *with:* `BufferedReader d` `= new BufferedReader(new InputStreamReader(in));`
`long`	**`readLong`**`()` See the general contract of the `readLong` method of `DataInput`.
`short`	**`readShort`**`()` See the general contract of the `readShort` method of `DataInput`.
`int`	**`readUnsignedByte`**`()` See the general contract of the `readUnsignedByte` method of `DataInput`.
`int`	**`readUnsignedShort`**`()` See the general contract of the `readUnsignedShort` method of `DataInput`.
`String`	**`readUTF`**`()` See the general contract of the `readUTF` method of `DataInput`.
`static String`	**`readUTF`**`(`**`DataInput`**` in)` Reads from the stream `in` a representation of a Unicode character string encoded in **modified UTF-8** format; this string of characters is then returned as a `String`.
`int`	**`skipBytes`**`(int n)` See the general contract of the `skipBytes` method of `DataInput`.

Lecture/écriture haut niveau

- Méthodes de la class DataOutputStream:

Methods

Modifier and Type	Method and Description
void	**flush**() Flushes this data output stream.
int	**size**() Returns the current value of the counter `written`, the number of bytes written to this data output stream so far.
void	**write**(byte[] b, int off, int len) Writes `len` bytes from the specified byte array starting at offset `off` to the underlying output stream.
void	**write**(int b) Writes the specified byte (the low eight bits of the argument `b`) to the underlying output stream.
void	**writeBoolean**(boolean v) Writes a `boolean` to the underlying output stream as a 1-byte value.
void	**writeByte**(int v) Writes out a `byte` to the underlying output stream as a 1-byte value.
void	**writeBytes**(String s) Writes out the string to the underlying output stream as a sequence of bytes.
void	**writeChar**(int v) Writes a `char` to the underlying output stream as a 2-byte value, high byte first.
void	**writeChars**(String s) Writes a string to the underlying output stream as a sequence of characters.
void	**writeDouble**(double v) Converts the double argument to a `long` using the `doubleToLongBits` method in class `Double`, and then writes that `long` value to the underlying output stream as an 8-byte quantity, high byte first.
void	**writeFloat**(float v) Converts the float argument to an `int` using the `floatToIntBits` method in class `Float`, and then writes that `int` value to the underlying output stream as a 4-byte quantity, high byte first.
void	**writeInt**(int v) Writes an `int` to the underlying output stream as four bytes, high byte first.
void	**writeLong**(long v) Writes a `long` to the underlying output stream as eight bytes, high byte first.
void	**writeShort**(int v) Writes a `short` to the underlying output stream as two bytes, high byte first.
void	**writeUTF**(String str) Writes a string to the underlying output stream using **modified UTF-8** encoding in a machine-independent manner.

Exercice N°6 : jeu d'allumettes

Un certain nombre d'allumettes est disposé entre les deux partis, l'ordinateur (le serveur) et vous (le client). Le but du jeu est de ne pas retirer la dernière allumette. Pour se faire, une prise maximale est désignée par le joueur.

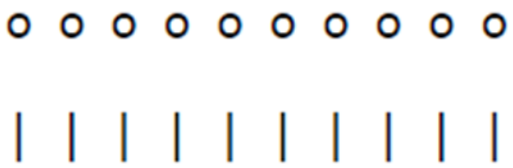
En début de partie, l'utilisateur à travers le programme client définit:

1. Le nombre d'allumettes disposées entre les deux joueur (de 10 à 60).
2. Le nombre maximal d'allumettes que l'on peut retirer.
3. Qui commence (0 pour le joueur et 1 pour l'ordinateur).

Ces paramètre doivent être communiquées au serveur. Puis, tour à tour, le client (grâce au choix de l'utilisateur) et le serveur (ordinateur) donne le nombre d'allumettes qu'il prend. La partie se termine lorsqu'il n y a plus d'allumettes sur la table. La personne ayant tirée la dernière allumette est le perdant, l'autre le vainqueur.

Exercice N°6 : jeu d'allumettes (suite)

- Pour vous aider, ouvrez et analyser le code java Allumette.java (qui est sur moodle).
- Ce code implémente le jeu d'allumette dans un seul programme.
- Éclater ce code en deux programmes client/serveur qui communiquent à travers un socket
- Après chaque jeu le programme affiche les allumettes qui restent :



Exercices supplémentaires

Transfert des objets via les sockets

- Sérialisation des objets:

```
public class Compte implements Serializable{
```

- Passage des Objet en écriture:

```
Compte cpt=new Compte("lili lili", 300);

ObjectOutputStream outs=new ObjectOutputStream(client.getOutputStream());
outs.writeObject(cpt);
```

- Réception des types primitifs en lecture:

```
ObjectInputStream ins=new ObjectInputStream(client.getInputStream());
System.out.println("les information du compte client : "+(Compte)ins.readObject());
```

transfert d'un objet

- Définir une class ''Restaurant'' caractérisée par un nom (de type String), un numéro de téléphone (de type String), et une position (de type ''Point2D '').
- Définir une class Serveur qui initie un tableau de restaurants et instancie un socket de connexion.
- Créer la class Client qui se connecte au serveur pour demander quel restaurant est proche d'une position communiquée via un socket.
- La class Serveur répond en envoyant un objet restaurant qui correspond à la requête client.

transfert d'un objet (suite)

- Modifier la réponse de la class Serveur pour renvoyer la liste des trois restaurants les plus proches triée.