INFO0503
La boîte à outils

INTRODUCTION À LA PROGRAMMATION CLIENT / SERVEUR

L3 INFORMATIQUE

OLIVIER FLAUZAC & CYRIL RABAT

Plan

Adressage / Nommage

Exécutions parallèles

Entrées / sorties en Java

Flux et objets

Communications réseau

Transférer des objets dans le réseau

Retour sur la notion de client réseau

Adressage / Nommage

Nommage

Désignation des machines par des noms

- `www.google.fr`

Liaison nom / adresses IP

- www.google.fr <=>74.125.132.94

Utilisation d'un serveur **DNS**

- Mise en place de requêtes Nom → IP
- Mise en place de requêtes IP → Nom

Adresses IP et Java

Classe de représentation des adresses

- InetAddress
- déclinaison en IPv4 et IPv6
- par défaut IPv4

La classe InetAddress

- Aucun constructeur public ???
- méthodes statiques de construction
- pas de création d'adresses inexistantes

InetAddress : Création

```
public static InetAddress getByAddress(byte[] addr)
                throws UnknownHostException


public static InetAddress getLocalHost()
                throws UnknownHostException


public static InetAddress getByName(String host)
                throws UnknownHostException
```

InetAddress : Méthodes

```
public byte[] getAddress()

public String getHostAddress()

public String toString()
```

InetAddress : Exemple

```
...
        try {
            InetAddress i = InetAddress.getByName("www.univ-reims.fr");
            System.out.println(i);
            System.out.println(i.getHostName());
            System.out.print("adresse : ");
            byte[] b = i.getAddress();
            for (int k = 0; k < b.length; k++) {
                System.out.print(b[k] + " ");
            }
            System.out.println();
            System.out.println(i.getHostAddress());
        }catch (Exception e) {
            System.out.println("Erreur");
        }
...
```

```
www.univ-reims.fr/194.57.105.10
www.univ-reims.fr
adresse : -62 57 105 10
194.57.105.10
```

Exécutions parallèles

Principe

Définition de flux d’exécution multiple

Utilisation de threads (processus légers)

Architecture

- un processus lanceur
- des threads fils

Mise en œuvre

Utilisation de la classe Thread

Héritage de Thread

- masquage de la méthode run
- lancement avec la méthode start

Exemple

```
public class ThreadExemple extends Thread {
      int num;
      ThreadExemple() {
          num = 0;
      }
      ThreadExemple(int i) {
          num = i;
      }
      @Override
      public void run() {
          for (int k = 0; k < 10; k++) {
              System.out.println("Bonjour depuis " + num);
              long temps = (long) Math.random() * 100;
              try {
                  this.sleep(temps);
              } catch (Exception e) {
                  System.out.println("Erreur" + e);

              }
      }
  }
```

```
public class Lanceur {
      public static void main(String args[]){
      Thread t1 = new ThreadExemple(1);
      Thread t2 = new ThreadExemple(2);
      t1.start();
      t2.start();
    }
  }
```

Résultat

Bonjour depuis 1
Bonjour depuis 2
Bonjour depuis 1
Bonjour depuis 2
Bonjour depuis 2
Bonjour depuis 1
Bonjour depuis 2
Bonjour depuis 1
Bonjour depuis 2
Bonjour depuis 1
Bonjour depuis 2
Bonjour depuis 1
Bonjour depuis 2
Bonjour depuis 1
Bonjour depuis 2
Bonjour depuis 1
Bonjour depuis 2
Bonjour depuis 1
Bonjour depuis 2
Bonjour depuis 1
Bonjour depuis 2
Bonjour depuis 1

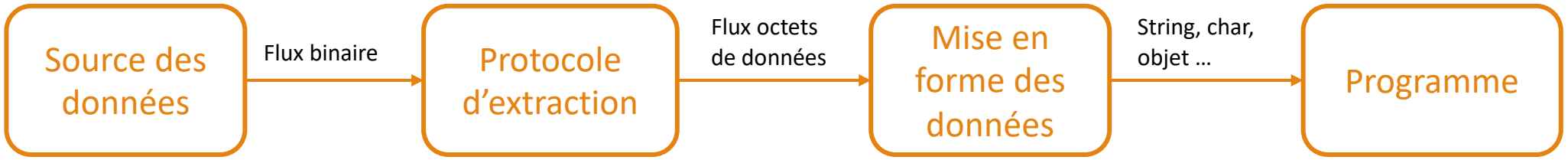
Entrées / sorties en Java

Exploitation de flux

Flux : canal d'échange d'informations

Solution Java

- Mise en place d'une hiérarchie objet
- Séparation des tâches élémentaires dans des classes
- Prise en charge de la gestion des éléments systèmes



Les flux en Java

Exploitation du package java.io

- Lecture / écriture
- Texte / binaire

Classes

- De gestion des flux
- De transcription

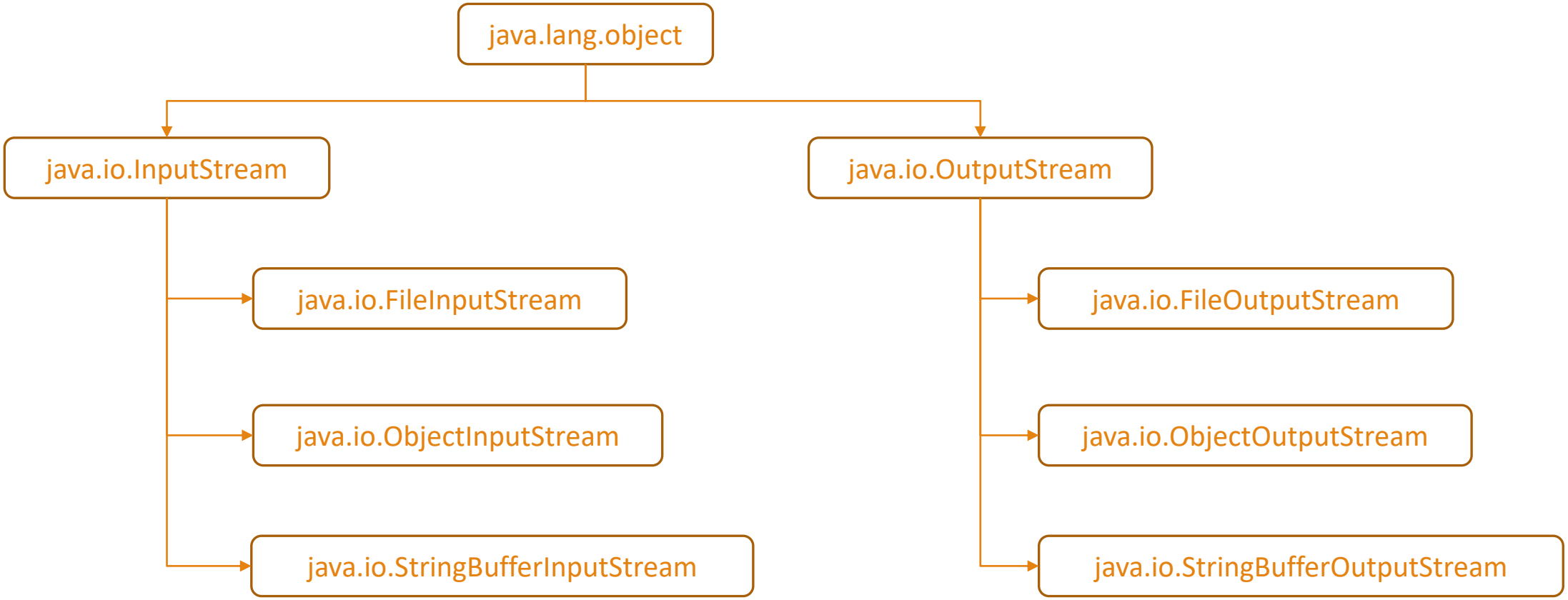
Flux binaires brut

- InputStream, OutputStream, FileInputSream, FileOutputstream …

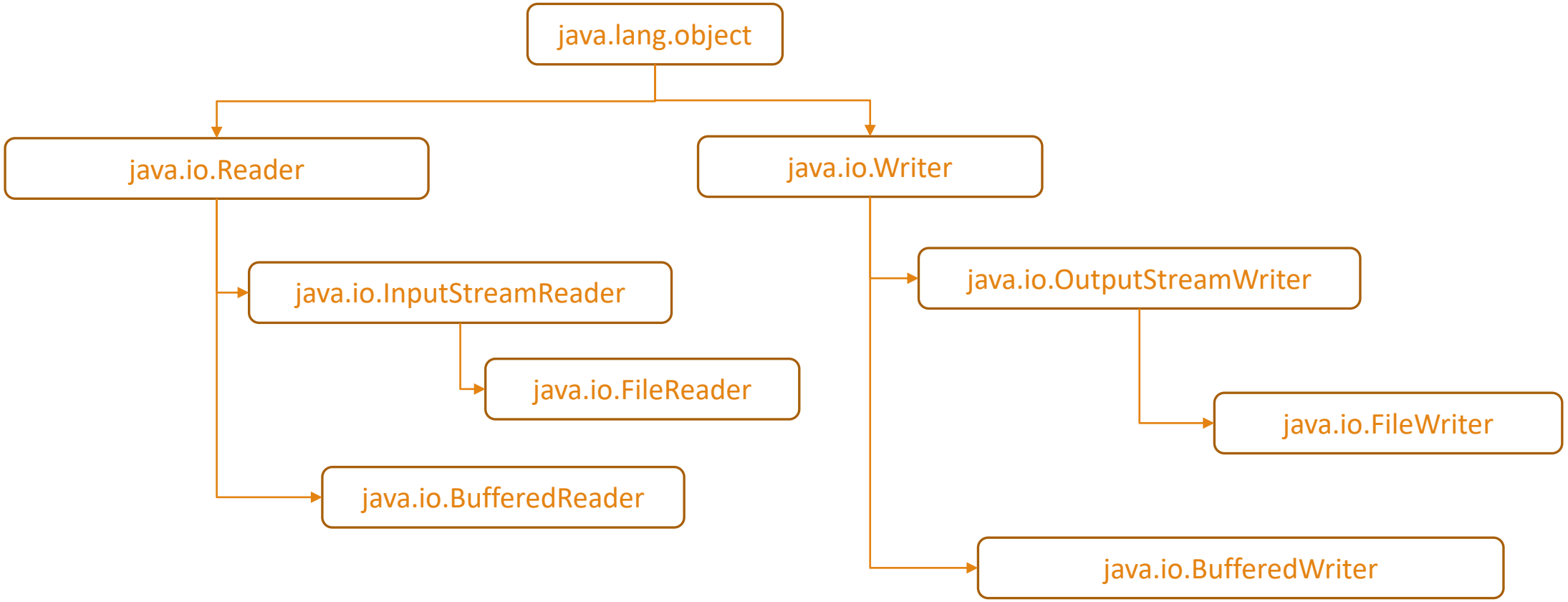
Flux de texte

- InputStreamReader, OutputSreamWriter, FileReader, FileWriter

Hiérarchie des classes



Hiérarchie des classes



Les fichiers

java.io.File : classe d'abstraction des fichiers et des chemins

Peut être en lien ou non avec le système de fichier

Gestion des séparateur selon les systèmes

Exploitation des propriétés des fichiers

- Lecture écriture

Opération sur les fichiers

- Suppression, création

Possibilité de créer des fichiers temporaires

- Destruction à la fin du programme

```
import java.io.File;
import java.io.IOException;
public class UserFile {
  public static void main(String[] args) {
    File f = new File("bob.fic");
    System.out.println("le fichier est un répertoire : " + f.isDirectory());
    System.out.println("nom absolu : " + f.getAbsolutePath());
    System.out.println("existe : " + f.exists());
    System.out.println("taille : " + f.length());
    try {
      f.createNewFile();
    } catch (IOException e) {
      System.out.println("Erreur de création");;
    }
    System.out.println("existe : " + f.exists());
    System.out.println("taille : " + f.length());
  }
}
```

```
le fichier est un répertoire : false
nom absolu : /Users/olivier/Documents/MyFiles/Enseignement/INFO0503/bob.fic
existe : false
taille : 0
existe : true
taille : 0
```

Fichiers texte

Exploitation de classe pour la gestion des fichiers texte

Création à partir

- De la chaîne de caractères du nom
- D'un objet File

Gestion de l'encodage

FileReader pour la lecture

- Méthodes : read, close
- Lecture char par char ou dans un tableau de char

FileWriter pour l'écriture

- Méthodes : append, write, flush, close

```
importjava.io.File;
importjava.io.FileWriter;
importjava.io.IOException;
Public class Ecrit{
  private static FileWriter fw=null;

  public static void main(String[]args) {
    try{
      Ecrit.fw=new FileWriter(newFile("toto.txt"));
    }catch(IOException e) {
      System.out.println("Erreur de création de fichier");
    }
    try{
      fw.write("Bonjour les amis \n");
      fw.write("NFP135 c'est top \n");
      fw.flush();
    }catch(IOException e) {
      System.out.println("Erreur de création de fichier");
    }
    try{
      fw.close();
    }catch(IOException se) {
      System.out.println("Erreur de fermeture");
    }
  }
}
```

```
importjava.io.File;
importjava.io.FileNotFoundException;
importjava.io.FileReader;
importjava.io.IOException;
publicclassLecture{
  privatestaticFileReaderfr=null;
  publicstaticvoidmain(String[]args) {
    char[]s=newchar[1000];
    try{
      fr=new FileReader(newFile("toto.txt"));
    }catch(FileNotFoundExceptione) {
      System.out.println("Erreur d'ouverture");
    }
    try{
      fr.read(s);
    }catch(IOExceptione) {
      System.out.println("Erreur de lecture");
    }
    for(charc:s) {
      System.out.print(c);
    }
    try{
      fr.close();
    }catch(IOExceptione) {
      System.out.println("Erreur de fermeture");
    }
  }
}
```

Agrégation de char

Char abrégeables avec a classe BufferedReader

Principe

- Collecte de caractère depuis un flux de caractère
- Jusqu'au délimiteur défini
- Retourne un String

Méthodes

- readLine : retourne null quand il n'y a plus rien à lire

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class Lecture {
  private static BufferedReader br = null;
  public static void main(String[] args) {
    String s = null;
    try {
      br = new BufferedReader(new FileReader(new File("toto.txt")));
    } catch (FileNotFoundException e) {
      System.out.println("Erreur d'ouverture");
    }
    try {
      while((s = br.readLine()) != null) {
        System.out.println(s);
      }
    } catch (IOException e) {
      System.out.println("Erreur de lecture");
    }
    try {
      br.close();
    } catch (IOException e) {
      System.out.println("Erreur de fermeture");
    }
    System.out.println("Erreur de création de fichier");
  }
}
```

Le Scanner

Scanner : classe assurant la lecture depuis un flux d'octets

Lecture

- Selon le type
- Jusqu'à la découverte d'un séparateur
  - Lecture au clavier : séparateur implicite

Scanner + Fichiers texte

- Lecture des lignes entières
- Construction avec un FileInputStream
- Exploitation de hasNext()

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Lecture2 {
  public static Scanner sc = null;
  public static void main(String[] args) {
    try {
      sc = new Scanner(new FileInputStream(new File("toto.txt")));
    } catch (FileNotFoundException e) {
      System.out.println("Erreur d'ouverture");
    }
    while(sc.hasNext()) {
      System.out.println(sc.nextLine());
    }
    sc.close();
  }
}
```

Flux et objets

Flux d’objets

Possibilité de manipuler l’état binaire d’un objet

Lecture / écriture de l’état binaire sur un support

Exploitation de la sérialisation

Sérialiser un objet : en extraire son état binaire

Désérialiser un objet : extraire un objet depuis un flux binaire

Les objets système ne sont pas sérialisable

- Connexion réseau
- Descripteur de fichier
- Processus
- …

Sérialisation

Implémentation de la classe Sérializable

Pas de méthode à masquer !

Exploitation de deux classes

- ObjectInputStream
  - Flux d'octets → Object
- ObjectOutputStream
  - Object → flux d'octet

Exploitation de la transformation vers le flux désiré

- Fichier
- Connexion réseau
- …

La classe Personne

```
import java.io.Serializable;
public class Personne implements Serializable{
   private String nom;
   private String prenom;
   public Personne(){
       this.nom = null;
       this.prenom = null;
   }
   public Personne(String nom, String prenom) {
       this.nom = nom;
       this.prenom = prenom;
   }
   public String getNom() { return this.nom; }
   public void setNom(String nom) { this.nom = nom;}

   public String getPrenom() { return prenom;}
   public void setPrenom(String prenom) { this.prenom = prenom; }
}
```

Ecriture d'une Personne

```
Personne p = new Personne("Flauzac","Olivier");
try{
  FileOutputStream fos = new FileOutputStream("pers.dat");
  ObjectOutputStream oos = new ObjectOutputStream(fos);
  oos.writeObject(p);
  oos.flush();
  oos.close();
}catch(Exception e){
  System.out.println("Exception" + e);
}
```

Lecture d’une Personne

```
Personne p = null;
try{
  FileInputStream fis = new FileInputStream("pers.dat");
  ObjectInputStream ois = new ObjectInputStream(fis);
  p = (Personne) ois.readObject();
  ois.close();
  System.out.println(p.getNom() +";" + p.getPrenom());
}catch(Exception e){
  System.out.println("Exception" + e);
}
```

Communications réseau

Le mode connecté : TCP

Classes définies dans l'API

- client : Socket
- serveur : ServerSocket

Gestion des flux associés

- Récupération des flux
- Transformation en texte / objet

Côté client

```
public Socket(InetAddress address, int port) throws IOException
public Socket(String host, int port) throws UnknownHostException, IOException

public void close() throws IOException

public InetAddress getLocalAddress()
public int getLocalPort()

public InetAddress getInetAddress()
public int getPort()

public InputStream getInputStream()
public OutputStream getOutputStream()
```

Côté serveur

```
public ServerSocket(int port) throws IOException

public void close() throws IOException

public Socket accept() throws IOException
```

Exemple : la source des données

```
...
 private static String address = "localhost";
 private static int port = 1500;
 ...
 try{
    Socket s = new Socket(address,port);
    PrintWriter ecr = new PrintWriter(
                  new OutputStreamWriter(s.getOutputStream()));
    ecr.print("Bonjour les amis");
    ecr.flush();
    ecr.close();
    s.close();
 }catch(Exception e){
    System.err.println("Erreur : " + e);
 }
```

Exemple : la destination des données

```
private static int port = 1500;
 ...
 try{
    ServerSocket srv = new ServerSocket(port);
    Socket s = srv.accept();
    Scanner scan = new Scanner(s.getInputStream());
    String chaine = scan.nextLine();
    System.out.println(chaine);
    scan.close();
    s.close();
 }catch(Exception e){
    System.out.println("Erreur" + e);
 }
```

Mode non connecté : UDP

Classes définies dans l'API

- DatagramPacket
- DatagramSocket

Particularités

- pas de connexion
- Nécessite de forger les paquet avant
- Gestion de la réception sur chaque acteur

Côté source

```
public DatagramPacket(byte[] buf,
                      int length,
                      InetAddress address,
                      int port)
```

Gestion des paquets

La classe DatagramPacket

Setters et getters associés aux champs

- setAddress, setPort ...
- getAdress, getPort ...

Gestion du transport : DatagramSocket

```
public DatagramSocket()

public void close() throws IOException

public void send(DatagramPacket p) throws IOException

public void receive(DatagramPacket p) throws IOException
```

La source des données

```
...
try {
   DatagramSocket datagramSocket = new DatagramSocket();
   byte[] buffer = "Bonjour les amis".getBytes();
   InetAddress receiverAddress = InetAddress.getLocalHost();
   DatagramPacket packet = new DatagramPacket(buffer,
                           buffer.length,
                           receiverAddress,
                           1500);

   datagramSocket.send(packet);
}catch (Exception e) {
   System.out.println("Erreur " + e);
}
...
```

La destination des données

```
...
try{
   DatagramSocket datagramSocket = new DatagramSocket(1500);
   byte[] buffer = new byte[50];
   DatagramPacket packet = new DatagramPacket(buffer,
                                   buffer.length);
   datagramSocket.receive(packet);
   Scanner sc = new Scanner(new ByteArrayInputStream(buffer));
   String s = sc.nextLine();
   System.out.println(s);
}catch(Exception e){
   System.out.println("Erreur " + e);
}
...
```

Transférer des objets dans le réseau

Principe

Utilisation de la sérialisation

Exploitation des flux associés aux Socket

Utilisation des flux de sérialisation

Transmission d’objets entre le client et le serveur

TCP et sérialisation

```
public class Personne implements Serializable{
   private static final long serialVersionUID = 1L;
   private String nom;
   private String prenom;

   public Personne(){
      this.nom = null;
      this.prenom = null;
   }
   public Personne(String nom, String prenom) {
      this.nom = nom;
      this.prenom = prenom;
   }

   public String getNom() { return this.nom; }
   public void setNom(String nom) { this.nom = nom;}

   public String getPrenom() { return this.prenom;}
   public void setPrenom(String prenom) { this.prenom = prenom; }
}
```

La source TCP

```
public class SourceSeriaTCP {
   private static String address = "localhost";
   private static int port = 1500;
   public static void main(String args[]){
      try{
         Personne p = new Personne("Flauzac","Olivier");
         Socket s = new Socket(address,port);
         OutputStream os = s.getOutputStream();
         ObjectOutputStream oos = new ObjectOutputStream(os);
         oos.writeObject(p);
         oos.flush();
         s.close();
      }catch(Exception e){
         System.err.println("Erreur : " + e);
      }
   }
}
```

La destination TCP

```
public class DestSeriaTCP {
   private static int port = 1500;

   public static void main(String[] args){
      try{
         ServerSocket srv = new ServerSocket(port);
         Socket s = srv.accept();
         InputStream is = s.getInputStream()
         ObjectInputStream ois = new ObjectInputStream(is);
         Personne p = (Personne) ois.readObject();
         System.out.println(p.getNom() +";" + p.getPrenom());
         s.close();
      }catch(Exception e){
         System.out.println("Erreur" + e);
      }
   }
}
```

La source UDP

```
import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;


public class SourceSeriaUDP {
    private static int port = 1500;
    public static void main(String[] args) {
        try {
            InetAddress address = InetAddress.getByName("localhost");
            Personne p = new Personne("olivier","Flauzac");
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ObjectOutputStream oos = new ObjectOutputStream(baos);
            oos.writeObject(p);
            oos.flush();
```

La source UDP

```
        byte[] Buf= baos.toByteArray();
        DatagramSocket socket = new DatagramSocket();
        DatagramPacket packet = new DatagramPacket(Buf, Buf.length, address, port);
        socket.send(packet);
    }catch(Exception e) {
    System.out.println(e);
    }
    }
}
```

La destination UDP

```
import java.io.BufferedInputStream;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class DestSeriaUDP {
    public static void main(String[] args) {
        try{
            byte[] recvBuf = new byte[5000];
            DatagramPacket packet = new DatagramPacket(recvBuf,
                                                recvBuf.length);
            DatagramSocket ds = new DatagramSocket(1500);
            ds.receive(packet);
```

La destination UDP

```
      ByteArrayInputStream byteStream = new ByteArrayInputStream(recvBuf);
      ObjectInputStream is = new ObjectInputStream(
                                  new BufferedInputStream(byteStream));
      Personne  p = (Personne) is.readObject();
      is.close();
      System.out.println(p.getNom() + " - " + p.getPrenom());
    }catch (Exception e){
      e.printStackTrace();
    }
}
}
```

Retour sur la notion de client serveur

Client / Serveur

Modèle d'échange pour les applications

Nécessite deux messages

- Une requête du client vers le serveur
- Une réponse du serveur vers le client

Non lié au type de transport

- Mise en œuvre d'un client serveur en mode connecté
- Mise en œuvre d'un client serveur en mode non connecté

Exemple un service de calcul

Définition d'un objet service : la requête

- Addition / soustraction

Définition d'un objet réponse

1. Emission de la requête par le client
2. Réception de la requête par le serveur
3. Traitement par le serveur
4. Emission de la réponse par le serveur
5. Réception de la réponse par le client
6. Consommation des données de la réponse par le client

La classe Service (Requête)

```
import java.io.Serializable;
public class Service implements Serializable{
	private static final long serialVersionUID = 1L;
	private int code;
	private double op1;
	private double op2;

	private Service(int c,double o1, double o2) {
		this.code = c;
		this.op1 = o1;
		this.op2 = o2;
	}
public static Service getInstanceServiceAdd(double o1,double o2) {
	return new Service(0,o1,o2);
}
```

```
public static Service getInstanceServiceDel(double o1,double o2) {
	return new Service(1,o1,o2);
}
public int getService() {
	return this.code;
}
public double getOp1() {
	return this.op1;
}
public double getOp2() {
	return this.op2;
}
}
```

La classe de réponse

```
import java.io.Serializable;
public class Reponse implements Serializable{
	private static final long serialVersionUID = 1L;
	private int codeRetour;
	private double res;

	private Reponse(int c,double r) {
		this.codeRetour = c;
		this.res = r;
	}
	public static Reponse getInstanceOK(double r)  { return new Reponse(1,r); }
	public static Reponse getInstanceErreur() {    return new Reponse(0,0); }
	public int getCodeRetour() { return this.codeRetour; }
	public double getRes() { return this.res; }
}
```

Mise en œuvre en mode connecté

Connexion à l'initiative du client

Mise en place d'un canal de communication pérenne

Mise en place d'une écoute côté serveur

Le client attend la réponse depuis le canal créé

Le client TCP

```
import java.io.OutputStream;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;


public class Client {
private static String address = "localhost";
   private static int port = 1500;
   public static void main(String args[]){
      try{
         Service sv = Service.getInstanceServiceAdd(89.4, 989.65);
         Socket s = new Socket(address,port);
         OutputStream os = s.getOutputStream();
         ObjectOutputStream oos = new ObjectOutputStream(os);
         oos.writeObject(sv);
         oos.flush();
         InputStream is = s.getInputStream();
         ObjectInputStream ois = new ObjectInputStream(is);
         Reponse p = (Reponse) ois.readObject();
         if(p.getCodeRetour() == 0) {
               System.out.println("Erreur");
         }else {
               System.out.println("requête OK - résultat = " + p.getRes());
         }

         s.close();
      }catch(Exception e){
         System.err.println("Erreur : " + e);
      }
   }
}
```

Le serveur TCP

```
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;
public class Serveur {
        private static int port = 1500;
        public static void main(String[] args){
                try{
         ServerSocket srv = new ServerSocket(port);
        Socket s = srv.accept();
        InputStream is = s.getInputStream();
        ObjectInputStream ois = new ObjectInputStream(is);
        Service sv = (Service) ois.readObject();
        double res = 0.0;
        if(sv.getService() == 0) {
          res = sv.getOp1() + sv.getOp2();
        }else {
           res = sv.getOp1() - sv.getOp2();
        }
        Reponse rep = Reponse.getInstanceOK(res);

        OutputStream os = s.getOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(os);
        oos.writeObject(rep);
        oos.flush();
        s.close();
   System.out.println("C'est fini");
      }catch(Exception e){
        System.out.println("Erreur" + e);
      }
   }
}
```

Mise en place de l'échange UDP

Nécessité de gérer côté client

- L'émission de la requête
- La réception de la réponse

Nécessité de gérer côté serveur

- La réception de la requête
- L'émission de la réponse

Changement de rôle pour chaque phase

- Le client est source puis destination
- Le serveur est destination puis source

Le client

```
import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.BufferedInputStream;
import java.io.ByteArrayInputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class ClientUDP {
    private static int portReq = 1500;
    private static int portRep = 1501;
```

Le client : l'émission

```
public static void main(String[] args) {
    try {
        InetAddress address = InetAddress.getByName("localhost");
        Service sv = Service.getInstanceServiceAdd(89.4, 989.65);
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(sv);
        oos.flush();
        byte[] Buf= baos.toByteArray();
        DatagramSocket socket = new DatagramSocket();
        DatagramPacket packet = new DatagramPacket(Buf,
                                                   Buf.length,address,
                                                   ClientUDP.portReq);

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

Le client : la réception

```
            byte[] recvBuf = new byte[5000];
            DatagramPacket packet = new DatagramPacket(recvBuf, recvBuf.length);
            DatagramSocket ds = new DatagramSocket(ClientUDP.portRep);
            ds.receive(packet);
            ByteArrayInputStream byteStream = new ByteArrayInputStream(recvBuf);
            ObjectInputStream is = new ObjectInputStream(
                                                new BufferedInputStream(byteStream));
            Reponse r = (Reponse) is.readObject();
            is.close();
            ds.close();
            System.out.println(r.getCodeRetour() + "-" + r.getRes());
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

Le serveur

```
import java.io.BufferedInputStream;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class ServerUDP {
    private static int portReq = 1500;
    private static int portRep = 1501;
```

Le serveur : la réception

```
public static void main(String[] args) {
    try{
        byte[] recvBuf = new byte[5000];
        DatagramPacket packet = new DatagramPacket(recvBuf, recvBuf.length);
        DatagramSocket ds = new DatagramSocket(ServerUDP.portReq);
        ds.receive(packet);
        ByteArrayInputStream byteStream = new ByteArrayInputStream(recvBuf);
        ObjectInputStream is = new ObjectInputStream(
                                    new BufferedInputStream(byteStream));
        Service s = (Service) is.readObject();
        is.close();
        ds.close();
```

Le serveur : le calcul

```
double res = 0.0;
if(s.getService() == 0) {
    res = s.getOp1() + s.getOp2();
}else {
    res = s.getOp1() - s.getOp2();
}
```

Le serveur : la réponse

```
            Reponse rep = Reponse.getInstanceOK(res);
            InetAddress address = InetAddress.getByName("localhost");
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ObjectOutputStream oos = new ObjectOutputStream(baos);
            oos.writeObject(rep);
            oos.flush();
            byte[] Buf= baos.toByteArray();
            DatagramSocket socket = new DatagramSocket();
            DatagramPacket packet2 = new DatagramPacket(Buf,
                                                Buf.length,address,
                                                ServerUDP.portRep);

            socket.send(packet2);
            socket.close();
        }catch(Exception e) {
            System.out.println(e);
        }
    }
}
```

Communication par messages

Communication par message

Modèle d'échange pour les applications

Nécessite un seul messages

Deux acteurs

- Producteur de données
- Consommateur de données

Non lié au type de transport

- Mise en œuvre en mode connecté
- Mise en œuvre en mode non connecté

Mise en oeuvre

Reprise des exemples précédents, mais sans réponse du serveur

- Affichage
- Sauvegarde du résultat

Pas de message de réponse

Mise en œuvre possible

- TCP
- UDP