Le réseau est l’ordinateur
(Bill Joy)

Le package java.net

Les principales classes du package java.net

- Les sockets en mode connecté

ServerSocket
accept() ...

Socket
getInputStream() getOutputStream() ...

- Les sockets en mode non connecté

DatagramSocket
receive(DatagramPacket p) send(DatagramPacket p) ...

InetAdress
getByName(String host) ...

Un client/serveur en mode connecté

- Un serveur ayant 1 client unique :

```
ServerSocket socket = new ServerSocket( 2010 ) ;
Socket client = socket.accept() ;
InputStream in = client.getInputStream() ;
// …
OutputStream out = client.getOutputStream() ;
client.close() ;
```

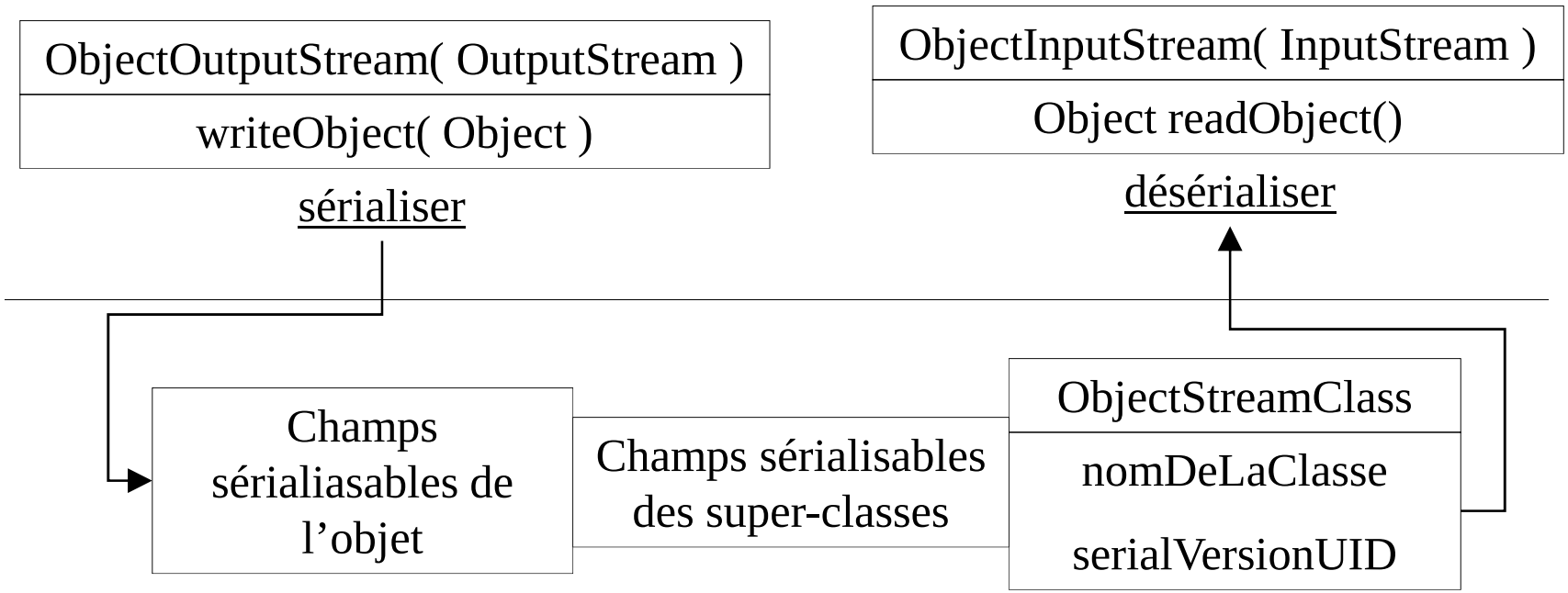
- Le client :

```
InetAddress adresseServeur = InetAddress.getByName( nomMachine ) ;
Socket client = new Socket( adresseServeur, 2010 ) ;
OutputStream out = client.getOutputStream() ;
// ...
InputStream in = client.getInputStream() ;
client.close() ;
```

La sérialisation d’objets au travers d’un réseau

Rappels sur la sérialisation d'objets

L'API de sérialisation :



Exemple de sérialisation coté serveur

```
public static void main( String [] args ){
    try{
        Bidon bidon = new Bidon() ;                          // objet à sérialiser
        ServerSocket socket = new ServerSocket( 2001 ) ;     // crée un serveur
        Socket client = socket.accept() ;                    // attend un client
        OutputStream fluxSocket = client.getOutputStream() ; // client -> flux

        // créé un flux de sérialisation et écrit dans le flux
        ObjectOutputStream flux = new ObjectOutputStream( fluxSocket ) ;
        flux.writeObject( bidon ) ;
        flux.flush() ;

        flux.close() ;
        fluxSocket.close() ;
        client.close() ;
        socket.close() ;
    }
    catch( IOException e ) {
    }
}
```

Exemple de désérialisation

```
public Object loadObject( String nomMachine )
throws IOException, ClassNotFoundException {
    // récupère l'adresse du serveur à partir de son nom
    InetAddress adresseServeur = InetAddress.getByName( nomMachine ) ;
    // créé un client et se connecte au serveur
    Socket client = new Socket( adresseServeur, 2001 ) ;
    InputStream fluxSocket = client.getInputStream() ;          // client -> flux

    // créé un flux de désérialisation et lit l'objet
    ObjectInputStream flux = new ObjectInputStream( fluxSocket ) ;
    return flux.readObject() ;
}

public static void main( String [] args ){
    ChargeObject client = new ChargeObjet() ;
    try{
        Bidon bidon = (Bidon)client.loadObject( "localhost" ) ;
    } catch( Exception e ) { // … }
}
```

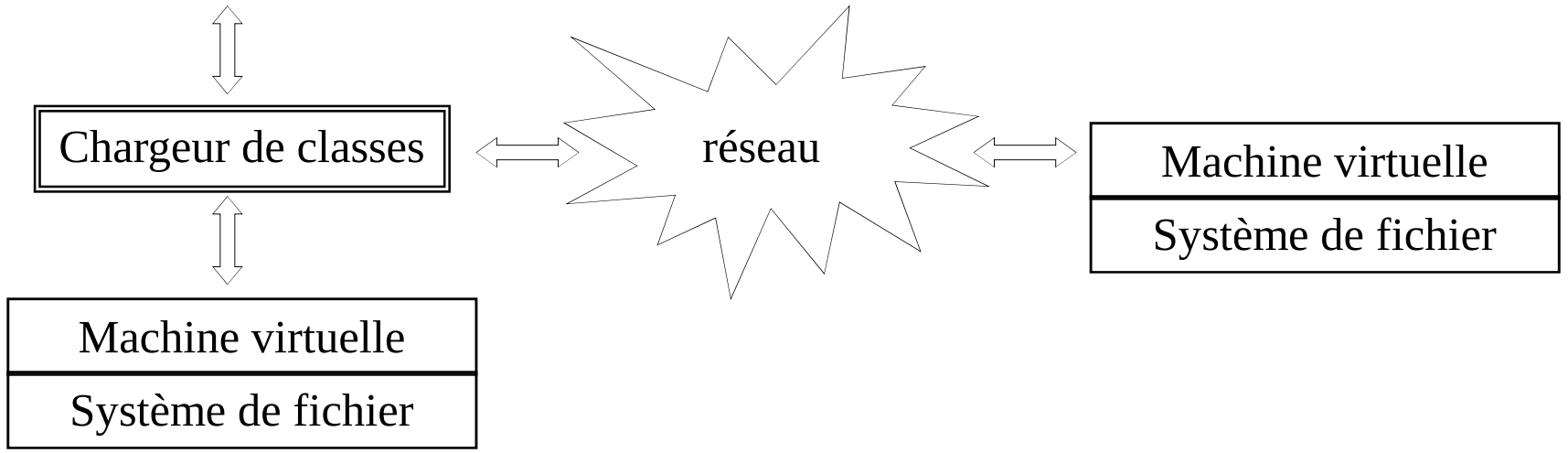
Extension du chargeur de classes

Le chargeur de classes

La machine virtuelle Java charge les classes au moment de l'exécution (en général la variable CLASSPATH définit le chemin de recherche des fichiers *.class).

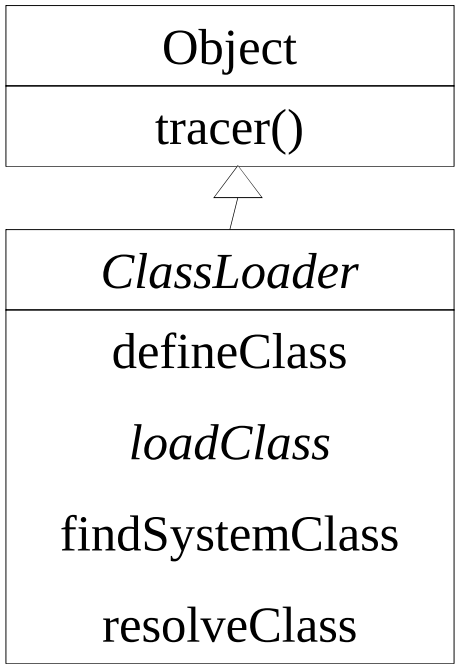
- Comment redéfinir un chargeur de classe pour une application particulière : charger une classe au travers d'un réseau ?

```
public class ClasseLocale {
        // …
        class ClasseDistante objetDistant = new ClasseDistante() ;
}
```



Le chargeur de classes

- La classe ClassLoader est une classe abstraite dont il faut définir la méthode abstraite loadClass.



Le chargeur de classes : principe

```
class MonChargeur extends ClassLoader{

  public synchronized Class loadClass(String className, boolean resolve)
  throws ClassNotFoundException {
```

- Vérifier si la classe n'a pas déjà été chargée,
  - si oui : résoudre la classe et la retourner ;
- Vérifier si la classe est dans le CLASSPATH ou dans les classes du système,
  - si oui : la retourner ;

⬇ Télécharger la classe et l'enregistrer dans un fichier ;

⬇ Lire le fichier de classe ;

- Définir la classe : convertir un tableau de byte en un objet de type Class ;
- Résoudre la classe : résoudre les références à d'autres classes (indispensable pour créer une instance de la classe avec la méthode newInstance) ;
- Retourner la classe.

```
  }
}
```

Exemple d'un chargeur de classes sur un réseau

```
class MonChargeur extends ClassLoader{

private Hashtable classes = new Hashtable() ;                    // ensemble de classes
public synchronized Class loadClass(String className, boolean resolve)
throws ClassNotFoundException {
      try {
            Class newClass = (Class)classes.get( className ) ;         // déjà chargée ?
            if( newClass == null ){
                  try{
                        newClass = findSystemClass( className ) ;      // fichier local ?
                        if( newClass != null ) return newClass ;
                  } catch (ClassNotFoundException e) {}
                  telechargeClasse( "localhost", className ) ;         // télécharge byte-code
                  byte [] buffer = chargeFichier( className ) ;        // lit fichier
                  newClass = defineClass( className, buffer, 0, buffer.length ) ;   // créé Class
                  classes.put( className, newClass ) ;                 // ajoute à l'ensemble
            }
            if( resolve ) resolveClass( newClass ) ;                   // résout la classe
            return newClass ;
      } catch( IOException e ) { throw new ClassNotFoundException( e.toString() ) ; }
}
```

Le chargeur de classes : téléchargement coté client

```
class MonChargeur extends ClassLoader{

    // …
    // ouvre une socket pour récupérer le byte-code
    // redirige la socket vers un flot d'entrée
    // écrit le byte-code dans un fichier

    private void telechargeClasse( String nomMachine, String className )
    throws IOException{
        InetAddress adresseServeur = InetAddress.getByName( nomMachine ) ;
        Socket client = new Socket( adresseServeur, 2010 ) ;
        InputStream fluxSocket = client.getInputStream() ;
        int bytesLus ;
        FileOutputStream fichierOut = new FileOutputStream( className +".class" );
        while ((bytesLus = fluxSocket.read()) != -1 )
            fichierOut.write( bytesLus );
        fichierOut.close() ;
    }
}
```

Le chargeur de classes : téléchargement coté serveur

```
public static void main( String [] args ){
    try{
        ServerSocket socket = new ServerSocket( 2010 ) ;              // créé un serveur
        Socket client = socket.accept() ;                             // accepte un client
        OutputStream fluxSocket = client.getOutputStream() ;          // socket -> flot
        FileInputStream in = new FileInputStream( "Bidon.class" ) ;  // ouvre fichier
        int byteLu ;
        while ((byteLu = in.read()) != -1) {                          // lit fichier
            fluxSocket.write(byteLu);                                 // envoi byte
        }
        in.close() ;
        fluxSocket.close() ;
        client.close() ;
        socket.close() ;
    }
    catch( IOException e ) {
        System.out.println( e ) ;
    }
}
```

Le chargeur de classes : transfert du byte-code vers un tableau de byte

```
class MonChargeur extends ClassLoader{

        // …
        // transfert du byte-code d'un fichier vers un tableau de byte

        private byte [] chargeFichier( String className )
        throws IOException, ClassNotFoundException{
                FileInputStream fichierIn = new FileInputStream( className + ".class" ) ;
                int length = fichierIn.available() ;       // taille du fichier
                if( length == 0 )
                        throw new ClassNotFoundException( className ) ;
                byte [] buffer = new byte[ length ] ;
                fichierIn.read( buffer ) ;
                return buffer ;
        }

}
```

Le chargeur de classes

- Une classe chargée dans la machine virtuelle Java ne peut pas être explicitement déchargée ;
- En autorisant la récupération du chargeur par le ramasse-miettes, on ne pourra plus charger de classe par son intermédiaire :

```
MonChargeur monChargeur = new MonChargeur() ;

// …
```

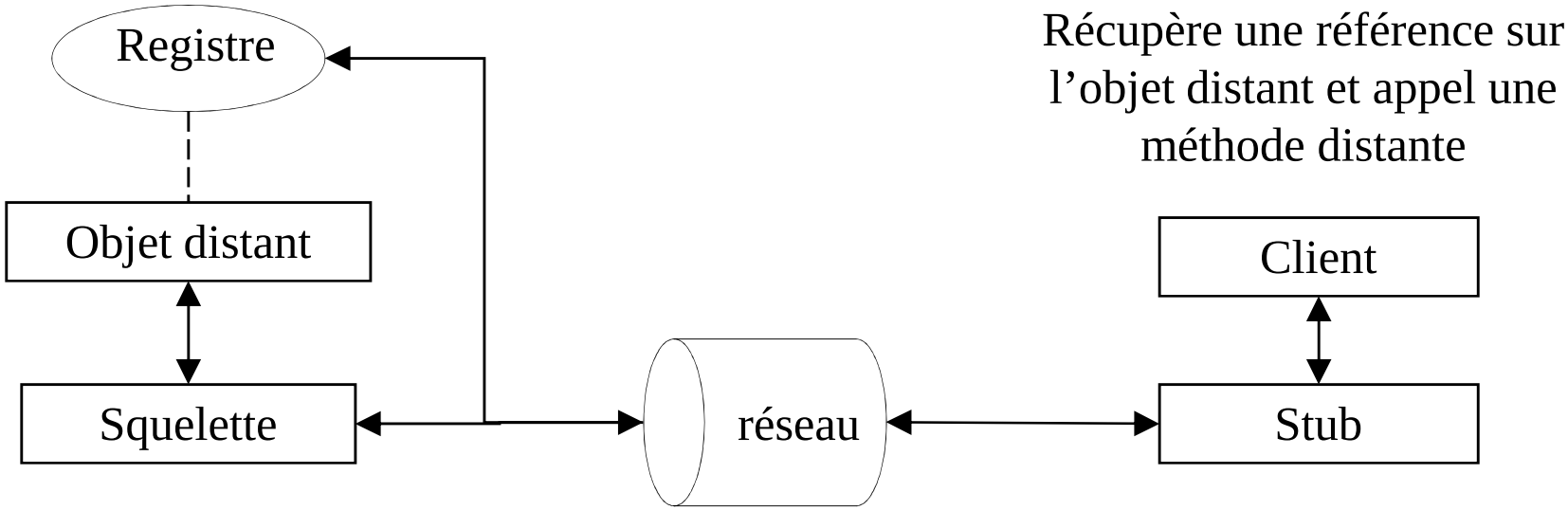
```
monChageur = null ;
```

```
public class ChargeurReseau extends ClassLoader{
   public ChargeurReseau( String nomServeur, int port ) throws IOException{
      InetAddress adresseServeur = InetAddress.getByName( nomServeur ) ;
      client = new Socket( adresseServeur, port ) ; //2001 ) ;
   }
   private void telechargeClasse( String className ) throws IOException{
      InputStream inputStream ;
      OutputStream outputStream ;
      ObjectOutputStream p ;
      BufferedWriter writer ;
      writer = new BufferedWriter(new OutputStreamWriter(client.getOutputStream()));
      writer.write( className, 0, className.length() ) ;
      writer.newLine() ;
      writer.flush() ;
      inputStream = client.getInputStream() ;
      DataInputStream dataInputStream = new DataInputStream( inputStream ) ;
      tailleFichier = dataInputStream.readInt() ;
      buffer = new byte[ tailleFichier ] ;
      for( int i=0; i<tailleFichier; i++ ){
         buffer[ i ] = (byte)inputStream.read() ;
      }
   }
}
```

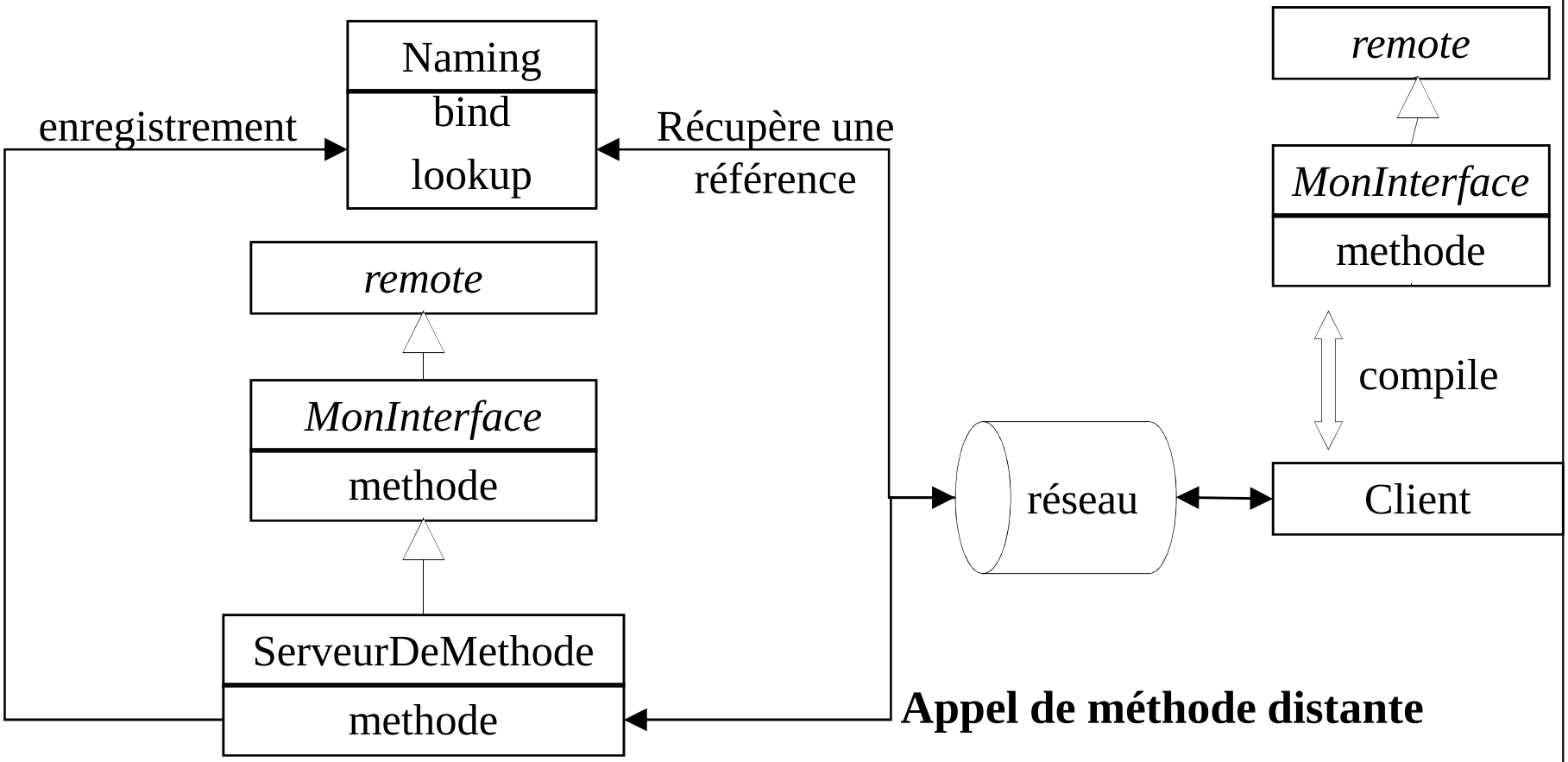
Remote Method Invocation

Remote Method Invocation

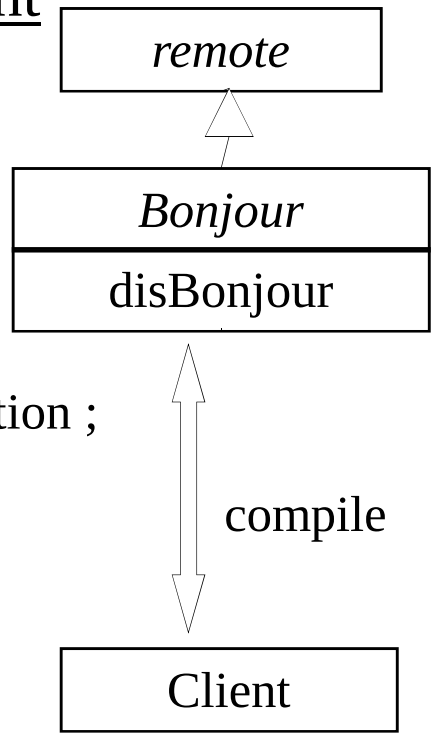
- Java permet de réaliser un appel de méthodes à distance (Remote Method Invocation) :



Remote Method Invocation



Remote Method Invocation coté client



- L'interface distante :

```
package paquetageInterface;
public interface Bonjour extends java.rmi.Remote{
        public String disBonjour() throws java.rmi.RemoteException ;
}
```

- Le programme client (classe Client):

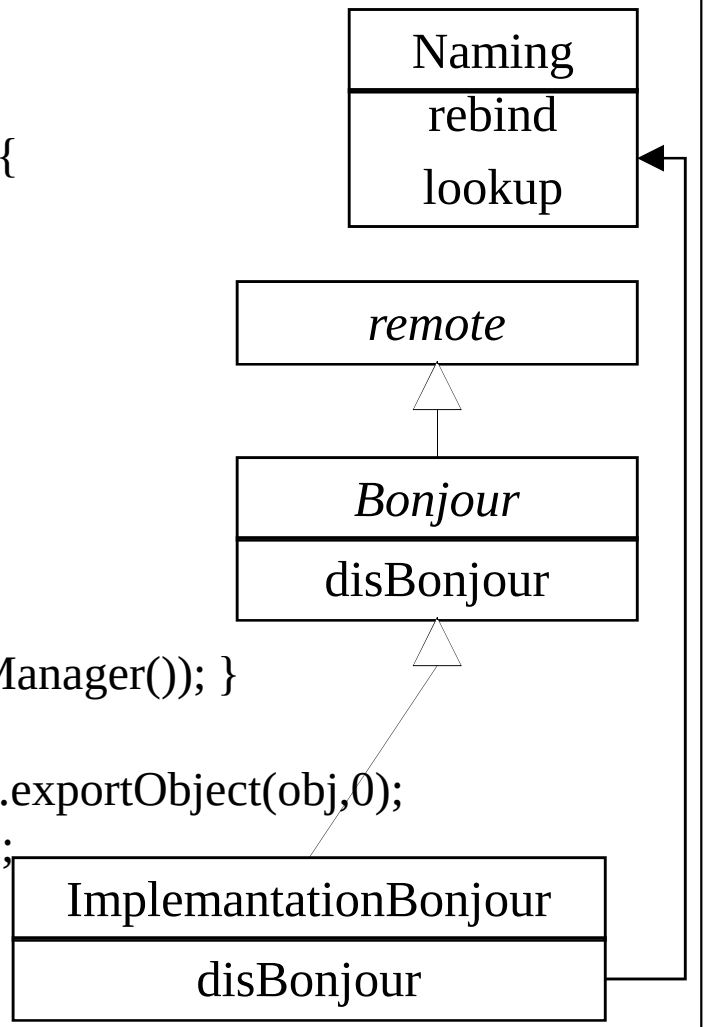
```
package paquetageClient;
import paquetageInterface.Bonjour;
import java.rmi.registry.*;
…
try{
        Registry registry = LocateRegistry.getRegistry( "host name", 2001 );
        Bonjour stub = (Bonjour) registry.lookup( "Monsieur" );
        String reponse = stub.disBonjour();
} catch( Exception e ){
}
```

Remote Method Invocation coté serveur

```
package paquetageServeur;
import java.rmi.serveur.UnicastRemoteObject;
import java.rmi.registry.*;
import paquetageInterface;

public class ImplementationBonjour implements Bonjour {
   public ImplementationBonjour () { super(); }
   public String disBonjour() {
            return "Heu... bonjour !";
   }

   public static void main(String args[]) {
      try {
            if (System.getSecurityManager() == null) {
                  System.setSecurityManager(new SecurityManager()); }
            Bonjour obj = new ImplementationBonjour();
            Bonjour stub = (Bonjour) UnicastRemoteObject.exportObject(obj,0);
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("Monsieur", stub);
      } catch (Exception e) { e.printStackTrace(); }
   }
}
```



Utilisation de RMI

- Les étapes pour utiliser RMI :
  - codage de l'interface ;
  - création d'un fichier jar pour l'interface ;
  - codage du serveur ;
  - compilation côté serveur ;
  - démarrage de l'annuaire d'objets ;
  - création d'un fichier déclarant la politique de sécurité utilisée ;
  - démarrage du serveur ;

  - donner physiquement le fichier jar contenant l'interface au développeur du client ;
  - codage du client ;
  - compilation côté client ;
  - création d'un fichier déclarant la politique de sécurité utilisée ;
  - démarrage du client.

- Pour plus de détails :

  http://java.sun.com/docs/books/tutorial/rmi/overview.html

Les plus et les moins de RMI

- Les plus :
  - l'enregistrement d'un seul objet dans le service de nommage suffit rendre accessible plusieurs objets du serveur : il suffit que des méthodes de l'objet enregistré retournent des objets présent sur le serveur ;
  - une méthode sur le serveur peut appeler une méthode d'un client si ce dernier à été transmis comme argument d'une méthode du serveur !
  - si le serveur est derrière un pare-feu qui ne laisse passer que HTTP, les appels de méthode sous automatiquement déroutées par le démon HTTP du serveur ;
- les moins :
  - si le client est derrière un pare-feu, il lui faudra un démon HTTP pour que le déroutage des requête passe par HTTP ;
  - dans une méthode appelée à distance, il est impossible d'utiliser une quelconque méthode de l'appelant => utiliser des threads ;
  - Un objet retourné par une méthode distante puis renvoyé à l'émetteur sera considéré comme distant par l'émetteur !

Communiquer avec un serveur web via HTTP

La classe HTTPURLConnection

- Un programme Java qui communique avec un serveur web :

```
URL url = new URL( "http://www..." );
HttpURLConnection connection = (HttpURLConnection)url.openConnection();
connection.setUseCaches( true );          // utilise des fichiers temporaires côté client
connection.setIfModifiedSince( ... );     // n'accède qu'aux fichiers récents
connection.connect();
int responseCode = connection.getResponseCode();
if( responseCode == HttpURLConnection.HTTP_NOT_FOUND ){
   ...
} else if( responseCode!=HttpURLConnection.HTTP_OK ){
  long fileLength = connection.getContentLength();
  InputStream inputStream = connection.getInputStream();
  ...          // TO DO : une boucle pour télécharger le fichier à l'URL
   connection.disconnect();
   ...
}
```

Les applets

Les applets : principe

- Les applets sont des programme Java téléchargés à partir d'un serveur WEB et qui s'exécutent dans le browser d'un client :

Fichier.html

```
<APPLET
CODEBASE=http://machin.truc.fr/bidon
code=appletBidon.class width=200 height=100>
</APPLET>
```

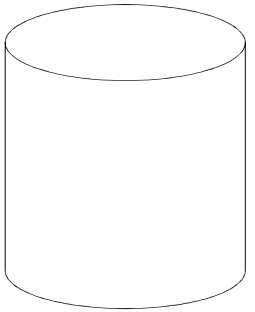
Browser du client

Demande la page Fichier.html

Télécharge la page Fichier.html

Demande la class appletBidon

Télécharge la classe appletBidon



Serveur WEB

appletBidon.class

Détail du téléchargement d'une applets

Fichier.html

```
<APPLET CODEBASE=http://machin.truc.fr/bidon
code=appletBidon.class width=200 height=100></APPLET>
```

Le browser :

- réserve une surface rectangulaire de 200*100 pixels
- se connecte au port 80 du serveur http://machin.truc.fr/bidon
- demande le fichier appletBidon.class (GET /bidon/appletBidon.class HTTP/1.0)

le serveur :

- transmet le fichier

le browser :

- stocke le fichier dans un tableau d 'octets

la machine virtuelle Java du browser :

- vérifie le byte-code
- charge la classe correspondante
- si la classe requiert une autre classe Java, celle ci est recherché dans le CLASSPATH du client, puis sur le serveur

Les applets et la sécurité

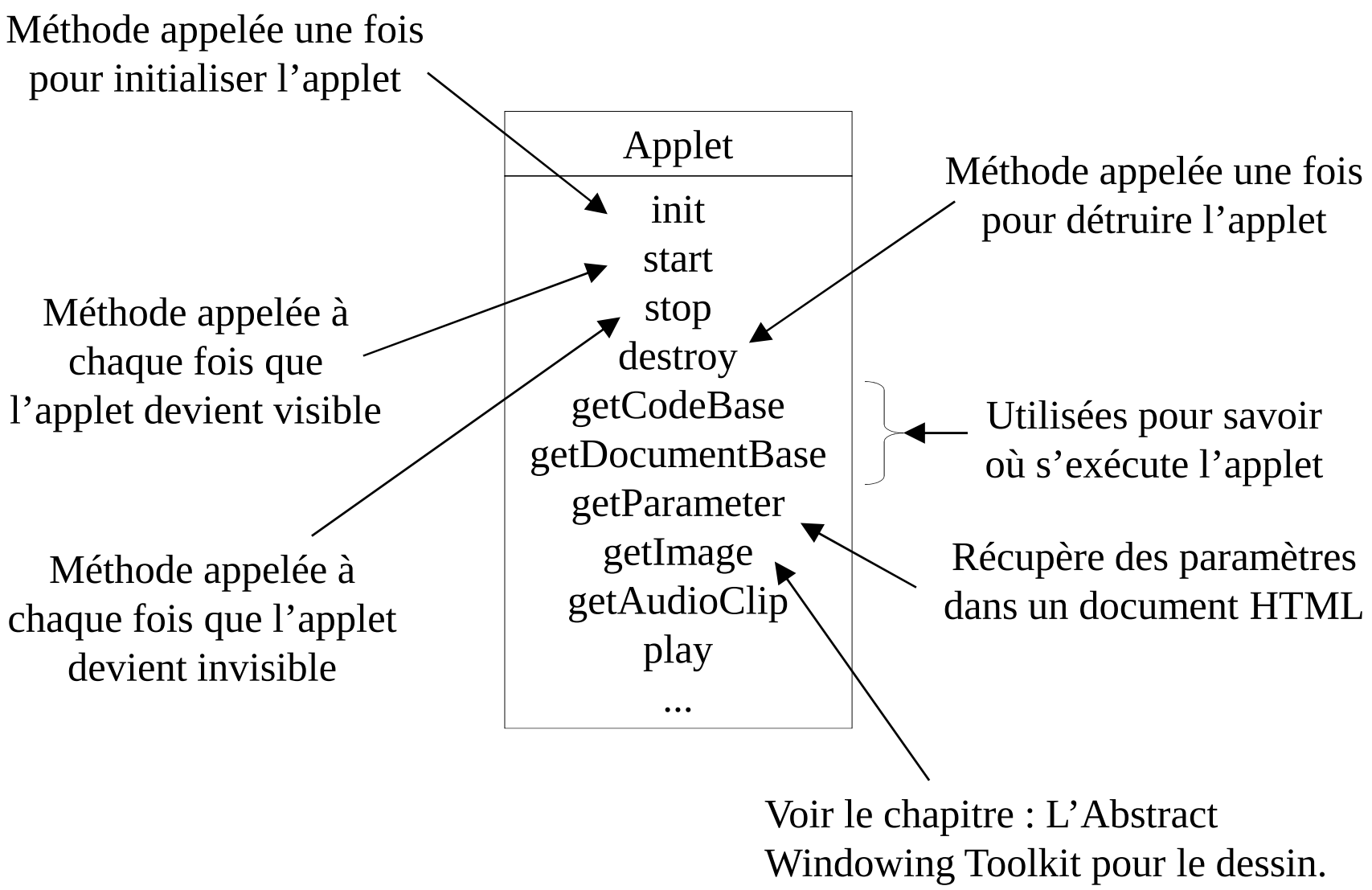
Les applets peuvent s'exécuter en toute sécurité dans un browser car elles ne peuvent pas :

- accéder aux système de fichier du client ;
- appeler des programmes sur la machine cliente : System.exec() ou Runtime.exec();
- redéfinir les classes : ClassLoader, SecurityManager, SocketImplFactory, …
- avec la version 1.2, il est possible d'assouplir la sécurité avec des applets signées.

Ce qu'il est parfois possible de faire :

- ouvrir une socket vers le serveur d'où provient l'applet (attention au firewall) ;
- appeler une méthode distante via RMI.

La classe Applet



Récupérer des paramètres d'un fichier HTML

Fichier .html

```
<applet code="Clock" width=50 height=50>
<param name=Color value="blue">
</applet>
```

```
Sring param = getParameter("Color") ;
System.out.println( param ) ;            // à l'écran : blue
```

Sample Applet

```
package appletapp;

import java.awt.*;
import javax.swing.*;

public class SampleApplet extends JApplet {

    //============================================================== main
    public static void main(String[] args) {
        //... Create an initialize the applet.
        JApplet theApplet = new SampleApplet();
        //theApplet.init();         // Needed if overridden in applet
        //theApplet.start();        // Needed if overridden in applet

        //... Create a window (JFrame) and make applet the content pane.
        JFrame window = new JFrame("Sample Applet and Application");
        window.setContentPane(theApplet);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.pack();              // Arrange the components.
        //System.out.println(theApplet.getSize());
        window.setVisible(true);    // Make the window visible.
    }

    //============================================== Applet constructor
    public SampleApplet() {
        add(new JLabel("This is both an Applet and Application!"));
    }
}
```