Serveur

```java
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;

public class Server {

    private static String message;

    // Afficher le buffer
            public static void afficherBuffer(int taille, byte[] buffer) {
                String strBuffer = new String(buffer);
                System.out.println(strBuffer.substring(0, taille));
            }
    public static void main (String[] args) {
        try {

            // Un buffer pour la lecture et l'ecriture
            byte[] buffer = new byte[8192];

            // Scanner pour l'ecriture du serveur
            Scanner scan = new Scanner(System.in);

            // Créer le serveur
            ServerSocket server = new ServerSocket(8080);
            // Attendre la connection d'un client au serveur
            Socket client = server.accept();

            // Récupérer l'input pour lire et l'output pour écrire
            OutputStream out = client.getOutputStream();
            InputStream in = client.getInputStream();

            // Ecouter les messages du client
            Thread listen = new Thread(new Runnable() {
                @Override
                public void run() {
                    byte[] buffer = new byte[8192];
                    while(!server.isClosed()) {
                        try {
                            int byteRead = in.read(buffer);
                            afficherBuffer(byteRead, buffer);
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                }

            });
            listen.start();
             // Envoyer les message au client
             while(!server.isClosed()) {
                 message = scan.nextLine();
                 buffer = message.getBytes();
                 out.write(buffer, 0, message.length());
             }

             client.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Client

```java
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.util.Scanner;

public class Client {

        // Afficher le buffer
        public static void afficherBuffer(int taille, byte[] buffer) {
            String strBuffer = new String(buffer);
            System.out.println(strBuffer.substring(0, taille));
        }

    public static void main (String[] args) {
        byte[] buffer = new byte[8192];
        Scanner scan = new Scanner(System.in);
        try {
            Socket client = new Socket("localhost", 8080);
            InputStream in = client.getInputStream();
            OutputStream out = client.getOutputStream();

            Thread listen = new Thread(new Runnable() {
                @Override
                public void run() {
                    byte[] buffer = new byte[8192];
                    while (client.isConnected()) {
                        try {
                            int byteRead = in.read(buffer);
                            System.out.println(new String(buffer));
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                }
            });
            listen.start();

            while (client.isConnected()) {
                String message = scan.nextLine();
                buffer = message.getBytes();
                out.write(buffer, 0, message.length());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}
```

Exemple permet la communication entre le serveur et le client (mini chat)

1- On crée l'interface **Product,java** qui hérite de Remote et dont toutes les méthode jettent l'exception **RemoteException** et contient les spécifications des méthodes,

```java
import java.rmi.Remote;

public interface Product extends Remote {
    // Lets us define API.

    public String getname() throws RemoteException;
    public String getDescription() throws RemoteException;
    public double getPrice() throws RemoteException;

}
```

2- On crée la classe **ProductImpl,java** qui hérite de **UnicastRemoteObjet** et implémente l'interface **Product,java,**

```java
public class ProductImpl implements Product {

// Attributes of product
public String name;
public String description;
public double price;


public ProductImpl(String newName, String newDescription, double newPrice) throws RemoteE
    this.name = newName;
    this.description = newDescription;
    this.price = newPrice;
}
public String getName() throws RemoteException {
    return this.name;
}

public String getDescription() throws RemoteException {
    return this.description;
}

public double getPrice() throws RemoteException {
    return this.price;
}
```

3- On crée la classe **Serveur,Java** qui contient le **main** du serveur ou bien on peut ajouter le **main** dans **ProductImpl,java** directement,

```java
import java.rmi.server.UnicastRemoteObject;
import java.rmi.registry.LocateRegistry;
import java.rmi.registy.Registry;

public class Server {
    public static void main(String [] args ) {
        try {

            // Set hostname for the server using javaProperty.
            System.setProperty("java.rmi.server.hostname", "127.0.0.")

            // Lets create product object.
            ProductImpl p1 = new ProductImpl("Laptop", "lenovo laptop", 1240.5);
            ProductImpl p2 = new ProductImpl("Mobile", "Mi mobile", 240.1);
            ProductImpl p3 = new ProductImpl("Power Charger", "Lenovo charger", 240.1);
            ProductImpl p4 = new ProductImpl("MoterBike", "Yamaha Biker", 38000.24);

            // Export p1, p2, p3 and p4 object using UnicastRemoteObject class
            Product stub1 = (Product) UnicastRemoteObject.exportObject(p1,0);
            Product stub2 = (Product) UnicastRemoteObject.exportObject(p2,0);
            Product stub3 = (Product) UnicastRemoteObject.exportObject(p3,0);
            Product stub4 = (Product) UnicastRemoteObject.exportObject(p4,0);

            // Register the exported class in RMI registry with some name,
            // Client will use that name to get the reference of those exported object.

// Get the registry to register the object.
Registry registry = LocateRegistry.getRegistry("127.0.0.1", 9100);

registry.rebind("l", stub1);
registry.rebind("m", stub2);
registry.rebind("c", stub3);
registry.rebind("b", stub4);
```

Objet réparti = extends from Remote

4- On crée la classe **Client,java** qui fait l'invocation des méthodes,

```java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class Client {
    public static void main(String [] args) {
        try {
// Locate the registry.
Registry registry = LocateRegistry.getRegistry("127.0.0.1", 9100);

// Get the reference of exported object from RMI Registry.
Product laptop = (Product) registry.lookup("l");
Product mobile = (Product) registry.lookup("m");
Product charger = (Product) registry.lookup("c");
Product bike = (Product) registry.lookup("b");

// Now we can invoke the method of the referenced objects.
System.out.println("The name of the laptop is " + laptop.getName());
System.out.println("The description is " + laptop.getDescription());
System.out.println("The price is " + laptop.getPrice());

System.out.println(mobile.getName());
System.out.println(charger.getName());
System.out.println(bike.getName());

} catch(Exception e) {
    System.out.println("Client side error..." + e);
}
```

file Hello.java

```
public interface Hello extends java.rmi.Remote {
  public void sayHello()
        throws java.rmi.RemoteException;
}
```

file HelloImpl.java

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject
                       implements Hello {
    String message;

   // Constructor implementation
   public HelloImpl(String msg) throws java.rmi.RemoteException {
        message = msg;
   }
  // Implementation of the remote method
   public void sayHello() throws java.rmi.RemoteException {
         System.out.println(message);
   }

      ...
```

Impleme
of t
server

file HelloImpl.java

```
public static void main(String args[]) {
   int port;    String URL;

   try {
     Integer I = new Integer(args[0]); port = I.intValue();
   } catch (Exception ex) {
     System.out.println(" Please enter: java HelloImpl <port>"); return;
   }

   try {
      // Launching the naming service - rmiregistry - within the JVM
      Registry registry = LocateRegistry.createRegistry(port);

      // Create an instance of the server object
      Hello obj = new HelloImpl();

      // compute the URL of the server
      URL = "//"+InetAddress.getLocalHost().getHostName()+":"+
                          port+"/my_server";
      Naming.rebind(URL, obj);
   } catch (Exception exc) { ...}
}
```

file HelloClient.java

```
import java.rmi.*;

public class HelloClient {
  public static void main(String args[]) {
    try {
      // get the stub of the server object from the rmiregistry
      Hello obj = (Hello) Naming.lookup("//my_machine/my_server");
      // Invocation of a method on the remote object
      obj.sayHello();
    } catch (Exception exc) { ... }
  }
}
```

Impleme
of t
client