

Applications Réparties :

Rapport de RPC



RPC REMOTE PROCEDURE CALL

Sous la supervision de
MR. AMAMOU Ahmed

CHERRADI Ibrahim
EL AMRANI Yassine

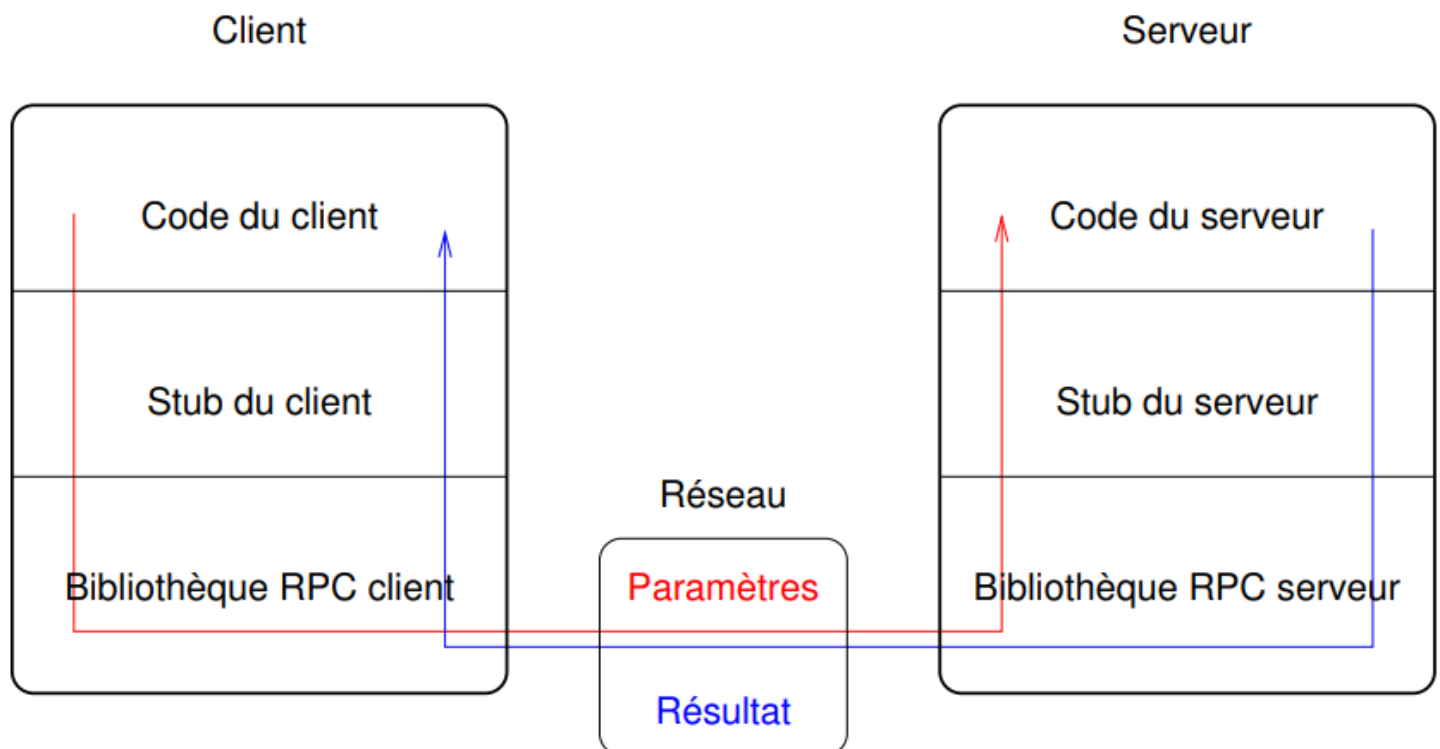
SOMMAIRE

1. ***Introduction***
2. ***Partie Théorique***
3. ***Partie Pratique***
4. ***Conclusion***

Introduction :

Les Appels de Procédure à Distance (RPC) représentent un protocole fondamental dans le monde de l'informatique moderne. Il permet à un programme de solliciter un service auprès d'un autre programme situé sur un ordinateur distant, sans nécessiter une connaissance approfondie des détails du réseau. Imaginez-le comme une sorte de télécommunication entre deux entités logicielles, où l'une demande un service et l'autre le fournit, le tout à travers le réseau.

Schéma de principe :



Partie Théorique

Le RPC, ou Remote Procedure Call (appel de procédure à distance), est un protocole de communication utilisé dans les systèmes distribués. Il est basé sur le modèle client-serveur, où le programme client fait une demande de service à un programme serveur distant. Voici quelques détails supplémentaires sur le fonctionnement du RPC :

- ***Modèle Client-Serveur :***

Dans le cadre du RPC, le programme demandeur est le client, tandis que le programme qui fournit le service est le serveur. Le client envoie une requête au serveur pour exécuter une procédure spécifique.

- ***Nature Synchrone :***

L'opération RPC est synchrone, ce qui signifie que le programme client est suspendu (bloqué) jusqu'à ce que les résultats de la procédure à distance soient retournés par le serveur. Cela garantit que les résultats sont disponibles avant que le programme client ne continue son exécution.

- ***Exécution Simultanée :***

Bien que les opérations RPC soient synchrone par nature, il est possible d'exécuter simultanément plusieurs RPC en utilisant des processus légers ou des threads partageant le même espace d'adressage. Cela permet à un programme client d'envoyer plusieurs requêtes à différents serveurs ou de traiter d'autres tâches pendant qu'il attend les réponses des appels RPC précédents.

Partie Pratique

Objectif :

Comprendre les concepts de base des RPC en mettant en place un système simple de procédures à distance avec Java RMI.

Étapes :

1. Création de l'interface distante :

- Définissez une interface qui déclare les méthodes que le client peut appeler à distance.

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface MonServiceRPCInterface extends Remote {  
    int addition(int a, int b) throws RemoteException;  
    int multiplication(int a, int b) throws RemoteException; }
```

2. Implémentation du serveur :

- Implémentez la classe du serveur qui étend UnicastRemoteObject et implémente l'interface distante.

```
import java.rmi.RemoteException;  
import java.rmi.server.UnicastRemoteObject;  
  
public class MonServiceRPCServeur extends UnicastRemoteObject  
implements MonServiceRPCInterface {  
    protected MonServiceRPCServeur() throws RemoteException {  
        super();  
    }  
  
    @Override  
    public int addition(int a, int b) throws RemoteException {  
        return a + b;  
    }  
  
    @Override  
    public int multiplication(int a, int b) throws RemoteException {  
        return a * b;  
    }  
  
    public static void main(String[] args) {  
        // Code pour démarrer le serveur RMI ici  
    }  
}
```

3. Implémentation du client :

- Créez la classe du client qui utilise l'interface distante.

```
import java.rmi.registry.LocateRegistry;  
import java.rmi.registry.Registry;  
  
public class MonServiceRPCClient {  
    public static void main(String[] args) {  
        try {  
            // Code pour récupérer la référence distante et appeler les méthodes du serveur ici  
        }  
    }  
}
```

Partie Pratique

```
} catch (Exception e) {
    e.printStackTrace();
}
}
```

4. Enregistrement du service :

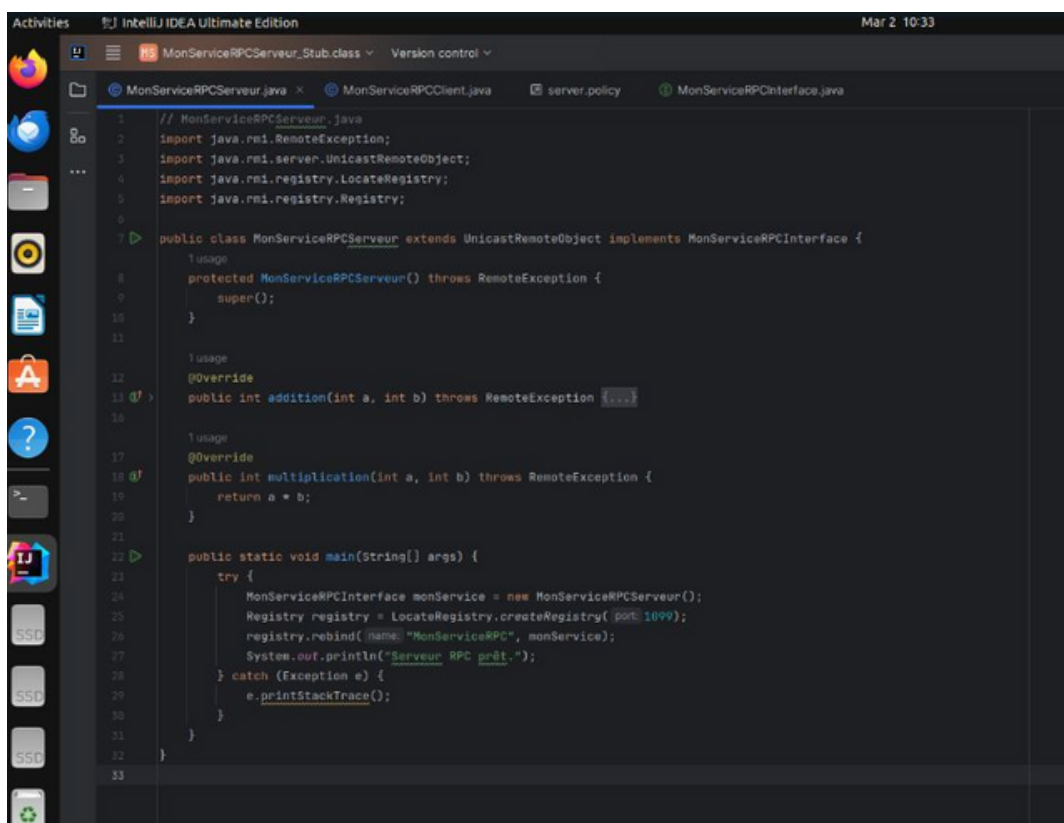
- o Dans le main du serveur, enregistrez le service auprès du registre RMI.

```
try {
    MonServiceRPCInterface monService = new MonServiceRPCServeur();
    Registry registry = LocateRegistry.createRegistry(1099);
    registry.rebind("MonServiceRPC", monService);
    System.out.println("Serveur RPC prêt.");
} catch (Exception e) {
    e.printStackTrace();
}
```

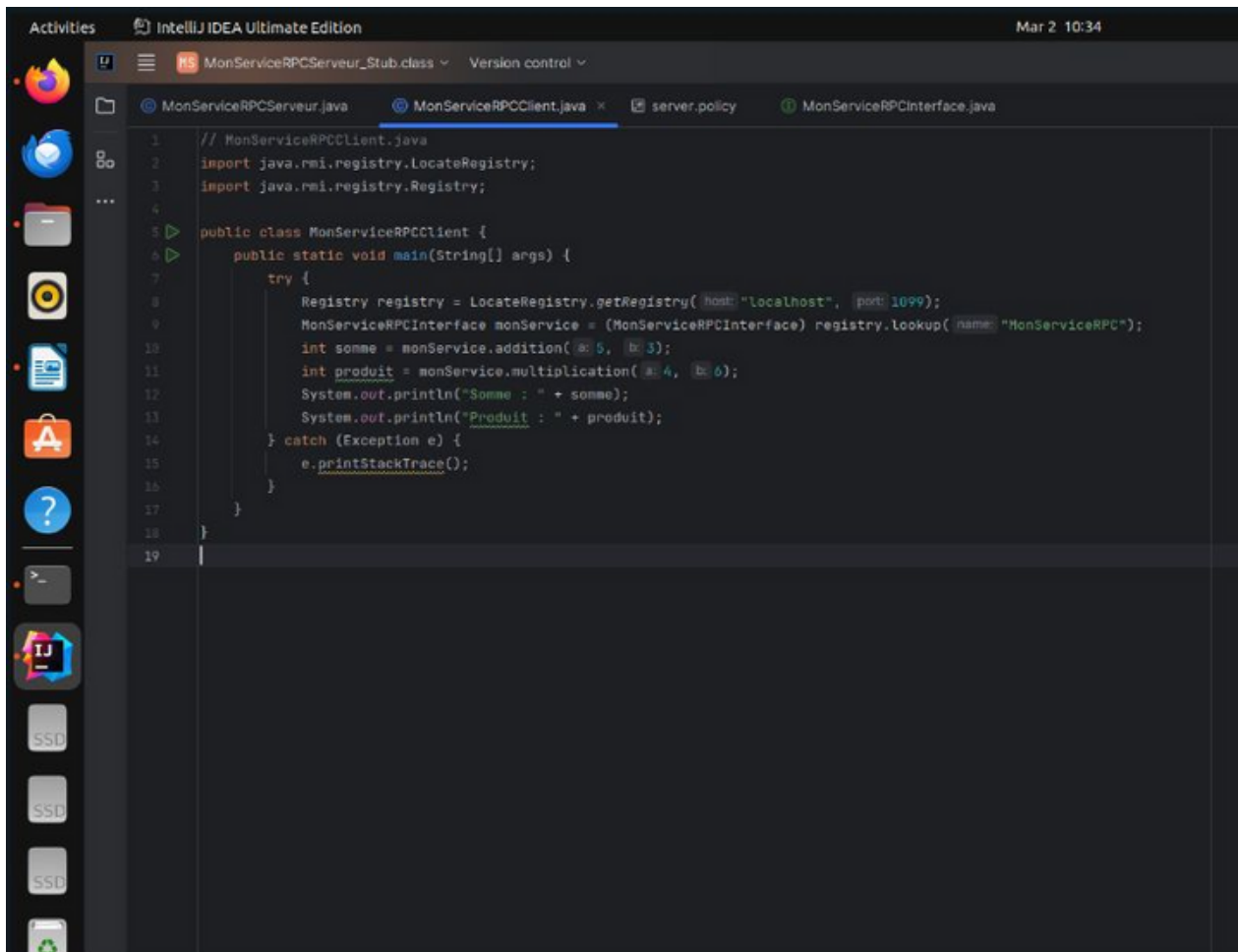
5. Appel du service depuis le client :

- o Dans le main du client, récupérez la référence distante du service et appelez les méthodes.

```
Registry registry = LocateRegistry.getRegistry("localhost", 1099);
MonServiceRPCInterface monService = (MonServiceRPCInterface)
registry.lookup("MonServiceRPC");
int somme = monService.addition(5, 3);
int produit = monService.multiplication(4, 6);
System.out.println("Somme : " + somme);
System.out.println("Produit : " + produit);
```

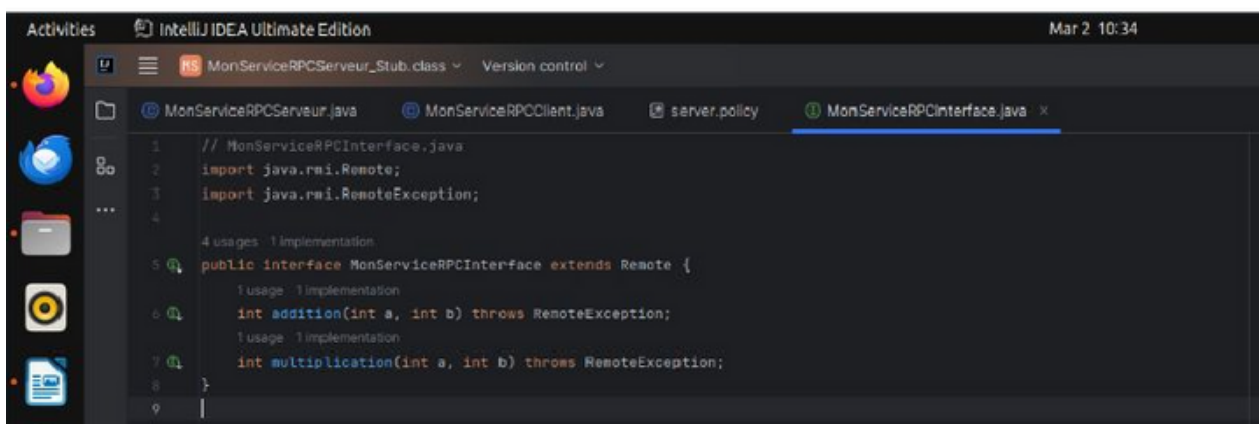


Partie Pratique



The screenshot shows the IntelliJ IDEA Ultimate Edition interface. The main editor displays the `MonServiceRPCClient.java` file. The code imports `java.rmi.registry LocateRegistry` and `java.rmi.registry Registry`. It defines a `MonServiceRPCClient` class with a `main` method. Inside `main`, it uses `LocateRegistry.getRegistry` to connect to a registry on `localhost` at port `1099`. It then looks up `MonServiceRPCInterface` and performs `addition` and `multiplication` operations. The results are printed to the console. A try-catch block handles any exceptions.

```
1 // MonServiceRPCClient.java
2 import java.rmi.registry.LocateRegistry;
3 import java.rmi.registry.Registry;
4
5 public class MonServiceRPCClient {
6     public static void main(String[] args) {
7         try {
8             Registry registry = LocateRegistry.getRegistry("localhost", 1099);
9             MonServiceRPCInterface monService = (MonServiceRPCInterface) registry.lookup("MonServiceRPC");
10             int somme = monService.addition(5, 3);
11             int produit = monService.multiplication(4, 6);
12             System.out.println("Somme : " + somme);
13             System.out.println("Produit : " + produit);
14         } catch (Exception e) {
15             e.printStackTrace();
16         }
17     }
18 }
19
```



The screenshot shows the IntelliJ IDEA Ultimate Edition interface. The main editor displays the `MonServiceRPCInterface.java` file. The code imports `java.rmi Remote` and `java.rmi RemoteException`. It defines a `MonServiceRPCInterface` interface that extends `Remote`. It contains two methods: `addition` and `multiplication`, both of which throw `RemoteException`.

```
1 // MonServiceRPCInterface.java
2 import java.rmi.Remote;
3 import java.rmi.RemoteException;
4
5 public interface MonServiceRPCInterface extends Remote {
6     int addition(int a, int b) throws RemoteException;
7     int multiplication(int a, int b) throws RemoteException;
8 }
9
```


Partie Pratique

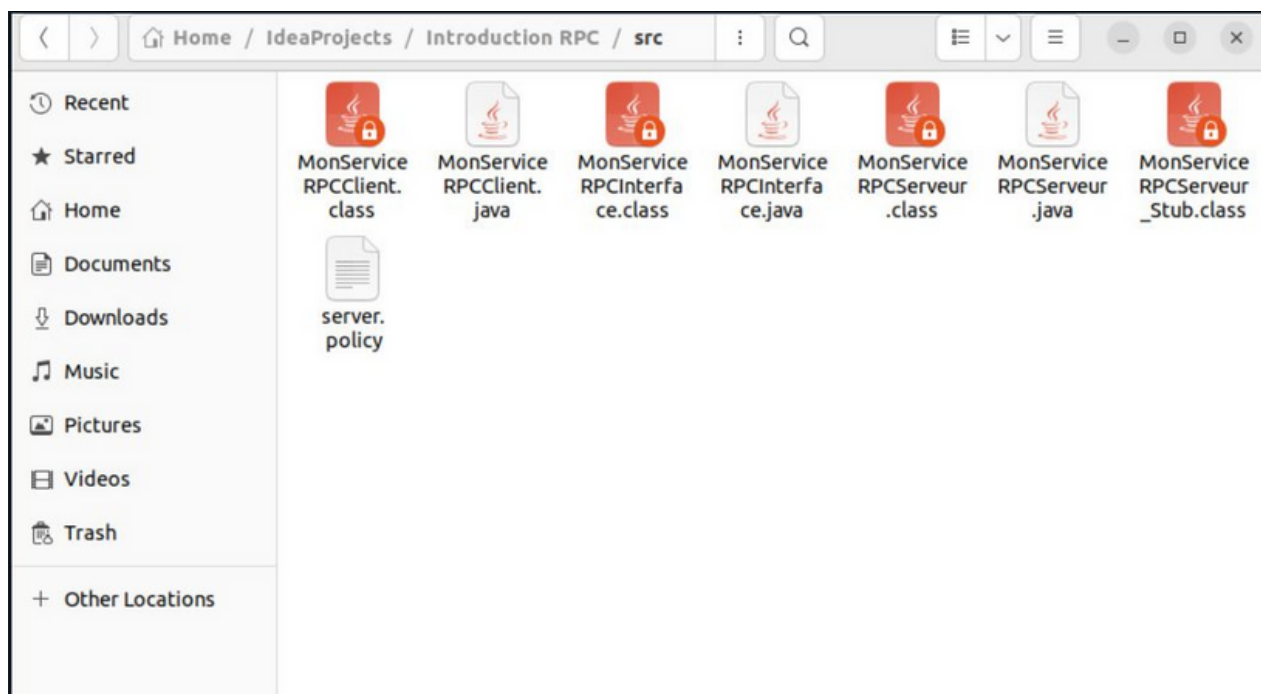
Déploiement :

1. Compilation du code :

- Ouvrez un terminal dans le répertoire de vos fichiers source Java.
- Utilisez la commande javac pour compiler les fichiers source.

```
javac MonServiceRPCInterface.java MonServiceRPCServeur.java MonServiceRPCClient.java
```

```
root@Yassine:/home/elamrani_yassine/IdeaProjects/Introduction RPC/src# javac MonServiceRPCInterface.java MonServiceRPCServeur.java MonServiceRPCClient.java
```



- Cela générera les fichiers .class correspondants.

2. Création du fichier stub :

- Utilisez la commande rmic pour générer le fichier stub du serveur.

```
rmic MonServiceRPCServeur
```

```
root@Yassine:/home/elamrani_yassine/IdeaProjects/Introduction RPC/src# rmic MonServiceRPCServeur
Warning: generation and use of skeletons and static stubs for JRPC
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
```

- Ceci créera un fichier MonServiceRPCServeur_Stub.class.
-


**MonService
RPCServeur
_Stub.class**

Partie Pratique

3. Démarrage du registre RMI :

- Dans le même répertoire, ouvrez un autre terminal et exécutez le registre RMI.

`rmiregistry &`

- Le `&` permet d'exécuter le registre en arrière-

```
root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src x root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src x root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src
[1]+ Exit 1 rmiregistry
root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src# rmiregistry &
[1] 7576
root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src# WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by sun.rmi.registry.RegistryImpl
WARNING: Please consider reporting this to the maintainers of sun.rmi.registry.RegistryImpl
WARNING: System::setSecurityManager will be removed in a future release
```

4. Exécution du serveur :

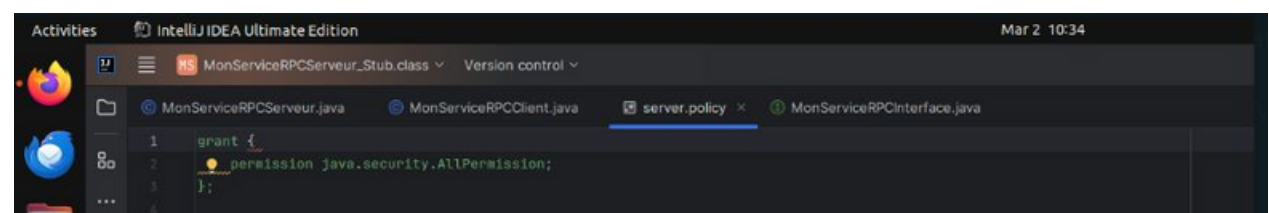
- Toujours dans le premier terminal, lancez votre serveur en spécifiant la politique de sécurité.

`java -Djava.security.policy=server.policy MonServiceRPCServeur`

```
root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src x root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src x root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src
root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src# javac MonServiceRPCInterface.java MonServiceRPCServeur.java MonServiceRPCClient.java
root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src# rmiregistry &
Warning: generation and use of skeletons and static stubs for JRMPI
Warning: is deprecated, skeletons are unnecessary, and static stubs have
Warning: been superseded by dynamically generated stubs. Users are
Warning: encouraged to migrate away from using rmic to generate skeletons and static
Warning: stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src# sudo lsof -i :1099
root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src# java -Djava.security.policy=server.policy MonServiceRPCServeur
java.rmi.server.ExportException: Port already in use: 1099; nested exception is:
java.net.BindException: Address already in use (Bind failed)
at java.rmi.sun.rmi.transport.tcp.TCPTransport.listen(TCPTransport.java:335)
at java.rmi.sun.rmi.transport.tcp.TCPTransport.exportObject(TCPTransport.java:243)
at java.rmi.sun.rmi.transport.tcp.TCPEndpoint.exportObject(TCPEndpoint.java:412)
at java.rmi.sun.rmi.transport.LLRef.exportObject(LLRef.java:147)
at java.rmi.sun.rmi.server.UnicastServerRef.exportObject(UnicastServerRef.java:234)
at java.rmi.sun.rmi.registry.RegistryImpl.setup(RegistryImpl.java:220)
at java.rmi.sun.rmi.registry.RegistryImpl.<init>(RegistryImpl.java:205)
at java.rmi/java.rmi.registry.LocateRegistry.createRegistry(LocateRegistry.java:203)
at MonServiceRPCServeur.main(MonServiceRPCServeur.java:25)
Caused by: java.net.BindException: Address already in use (Bind failed)
at java.base/java.net.PlainSocketImpl.socketBind(Native Method)
at java.base/java.net.AbstractPlainSocketImpl.bind(AbstractPlainSocketImpl.java:452)
at java.base/java.net.ServerSocket.bind(ServerSocket.java:395)
at java.base/java.net.ServerSocket.<init>(ServerSocket.java:257)
at java.base/java.net.ServerSocket.<init>(ServerSocket.java:149)
at java.rmi.sun.rmi.transport.tcp.TCPDLSocketFactory.createServerSocket(TCPDLSocketFactory.java:45)
at java.rmi.sun.rmi.transport.tcp.TCPDLSocketFactory.newServerSocket(TCPDLSocketFactory.java:670)
at java.rmi.sun.rmi.transport.tcp.TCPTransport.listen(TCPTransport.java:324)
... 8 more
^Croot@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src# sudo lsof -i :1099
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
rmireglist 7576 root 7u IPv6 75149 0t0 TCP *:rmireglist (LISTEN)
root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src# kill 7576
root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src# java -Djava.security.policy=server.policy MonServiceRPCServeur
Serveur RPC prêt.
```

- Assurez-vous d'avoir un fichier `server.policy` configuré avec les autorisations nécessaires. Un exemple simple pourrait être :

```
grant {
    permission java.security.AllPermission;
};
```



Partie Pratique



5. Exécution du client :

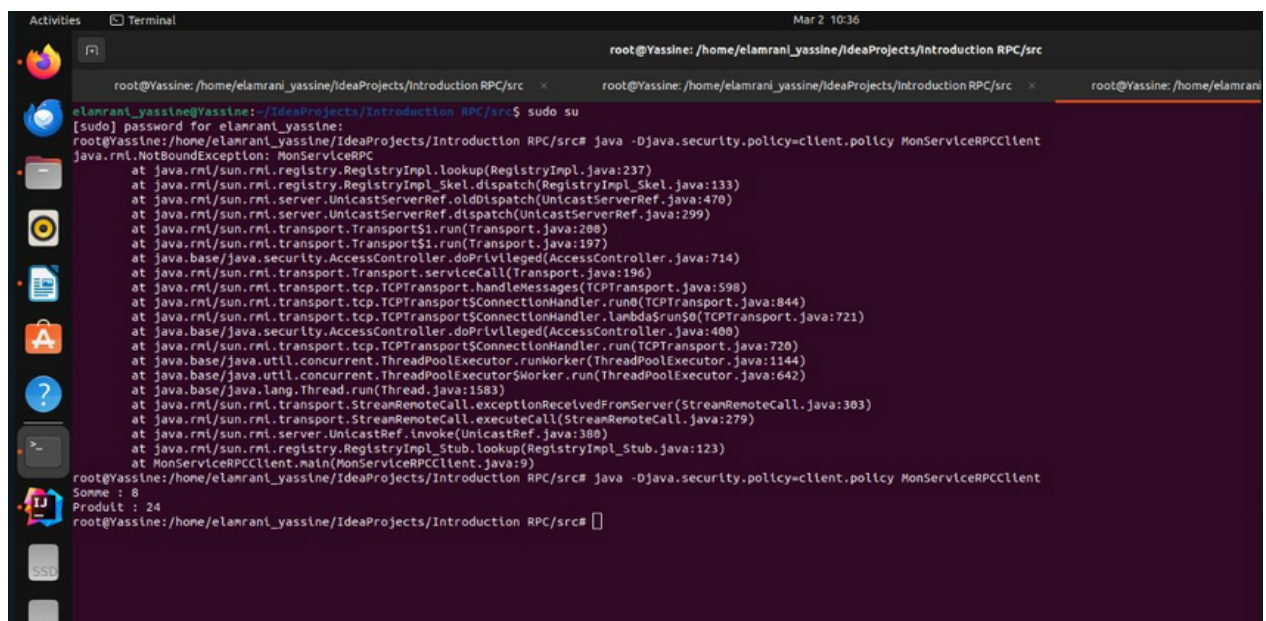
- Ouvrez un troisième terminal et démarrez votre client.

`java -Djava.security.policy=client.policy MonServiceRPCClient`

- Assurez-vous que le fichier client.policy est également configuré correctement.

Un

exemple simple de client.policy pourrait être similaire à celui du serveur.



```
root@Yassine: /home/elanrani_yassine/IdeaProjects/Introduction RPC/src$ sudo su
[sudo] password for elanrani_yassine:
root@Yassine: /home/elanrani_yassine/IdeaProjects/Introduction RPC/src# java -Djava.security.policy=client.policy MonServiceRPCClient
java.rmi.NotBoundException: MonServiceRPC
    at java.rmi.sun.rmi.registry.RegistryImpl.lookup(RegistryImpl.java:237)
    at java.rmi.sun.rmi.registry.RegistryImpl_Skel.dispatch(RegistryImpl_Skel.java:133)
    at java.rmi.sun.rmi.server.UnicastServerRef.oldDispatch(UnicastServerRef.java:478)
    at java.rmi.sun.rmi.server.UnicastServerRef.dispatch(UnicastServerRef.java:299)
    at java.rmi.sun.rmi.transport.Transport$1.run(Transport.java:288)
    at java.rmi.sun.rmi.transport.Transport$1.run(Transport.java:197)
    at java.base/java.security.AccessController.doPrivileged(AccessController.java:714)
    at java.rmi.sun.rmi.transport.Transport.serviceCall(Transport.java:196)
    at java.rmi.sun.rmi.transport.tcp.TCPTransport.handleMessages(TCPTransport.java:598)
    at java.rmi.sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run0(TCPTransport.java:844)
    at java.rmi.sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.lambda$run$0(TCPTransport.java:721)
    at java.base/java.security.AccessController.doPrivileged(AccessController.java:400)
    at java.rmi.sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run(TCPTransport.java:720)
    at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1144)
    at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:642)
    at java.rmi.sun.rmi.transport.StreamRemoteCall.exceptionReceivedFromServer(StreamRemoteCall.java:303)
    at java.rmi.sun.rmi.transport.StreamRemoteCall.executeCall(StreamRemoteCall.java:279)
    at java.rmi.sun.rmi.server.UnicastRef.invoke(UnicastRef.java:380)
    at java.rmi.sun.rmi.registry.RegistryImpl_Stub.lookup(RegistryImpl_Stub.java:123)
    at MonServiceRPCClient.main(MonServiceRPCClient.java:9)
root@Yassine: /home/elanrani_yassine/IdeaProjects/Introduction RPC/src# java -Djava.security.policy=client.policy MonServiceRPCClient
Somme : 8
Produit : 24
root@Yassine: /home/elanrani_yassine/IdeaProjects/Introduction RPC/src#
```

Partie Pratique

```

MonServiceRPCClient.java
1 import java.rmi.registry.LocateRegistry;
2 import java.rmi.registry.Registry;
3
4 public class MonServiceRPCClient {
5     public static void main(String[] args) {
6         if (args.length != 2) {
7             System.out.println("Usage: MonServiceRPCClient <nombre1> <nombre2>");
8             System.exit(1);
9         }
10
11         try {
12             String serverAddress = args[0]; // L'adresse IP du serveur
13             int nombre1 = Integer.parseInt(args[1]);
14             int nombre2 = Integer.parseInt(args[2]);
15
16             Registry registry = LocateRegistry.getRegistry(serverAddress, 1099);
17             MonServiceRPCInterface monService = (MonServiceRPCInterface)
18                 registry.lookup("MonServiceRPC");
19
20             int somme = monService.addition(nombre1, nombre2);
21             System.out.println("Somme : " + somme);
22
23             int produit = monService.multiplication(nombre1, nombre2);
24             System.out.println("Produit : " + produit);
25         } catch (Exception e) {
26             e.printStackTrace();
27         }
28     }
29 }

```

```

MonServiceRPCInterface.java
1 import java.rmi.Remote;
2 import java.rmi.RemoteException;
3
4 public interface MonServiceRPCInterface extends Remote {
5     int addition(int a, int b) throws RemoteException;
6     int multiplication(int a, int b) throws RemoteException;
7 }

```

```

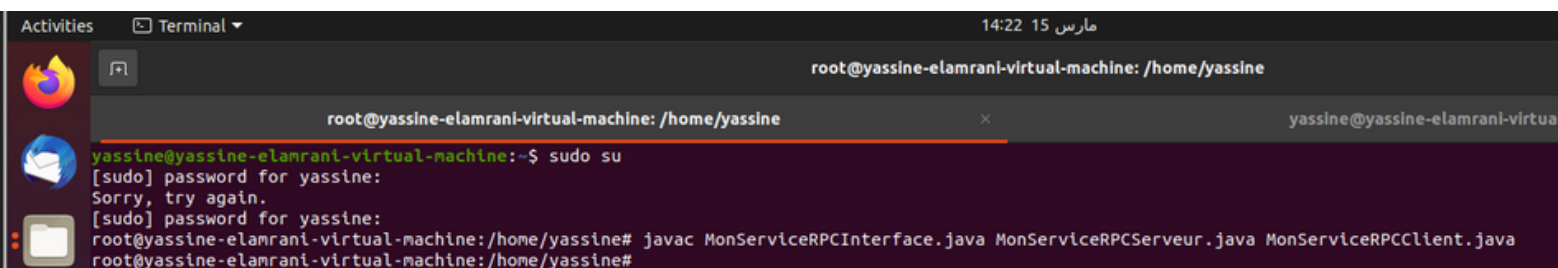
MonServiceRPCServeur.java
1 import java.rmi.RemoteException;
2 import java.rmi.server.UnicastRemoteObject;
3
4 public class MonServiceRPCServeur extends UnicastRemoteObject implements MonServiceRPCInterface {
5     protected MonServiceRPCServeur() throws RemoteException {
6         super();
7     }
8
9     @Override
10    public int addition(int a, int b) throws RemoteException {
11        return a + b;
12    }
13
14    @Override
15    public int multiplication(int a, int b) throws RemoteException {
16        return a * b;
17    }
18
19    public static void main(String[] args) {
20        try {
21            MonServiceRPCServeur monService = new MonServiceRPCServeur();
22            java.rmi.registry.LocateRegistry.createRegistry(1099).rebind("MonServiceRPC",
23                monService);
24            System.out.println("Serveur RMI prêt.");
25        } catch (Exception e) {
26            e.printStackTrace();
27        }
28    }
29 }

```


Partie Pratique

Compilation :

```
javac MonServiceRPCInterface.java MonServiceRPCServeur.java  
MonServiceRPCClient.java
```

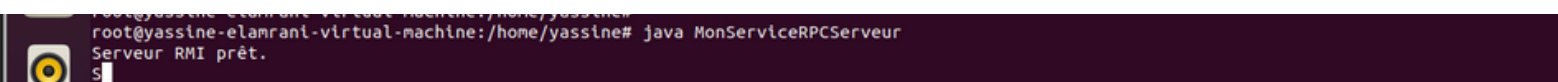


A terminal window titled 'Terminal' with a date and time of 14:22 15 مارس. The prompt is 'root@yassine-elamrani-virtual-machine: /home/yassine'. The user enters 'sudo su' and provides the password 'yassine'. Then, the user enters 'javac MonServiceRPCInterface.java MonServiceRPCServeur.java MonServiceRPCClient.java' and the compilation is successful.

```
root@yassine-elamrani-virtual-machine: /home/yassine  
yassine@yassine-elamrani-virtual-machine:~$ sudo su  
[sudo] password for yassine:  
Sorry, try again.  
[sudo] password for yassine:  
root@yassine-elamrani-virtual-machine: /home/yassine# javac MonServiceRPCInterface.java MonServiceRPCServeur.java MonServiceRPCClient.java  
root@yassine-elamrani-virtual-machine: /home/yassine#
```

Exécution du serveur :

```
java MonServiceRPCServeur
```

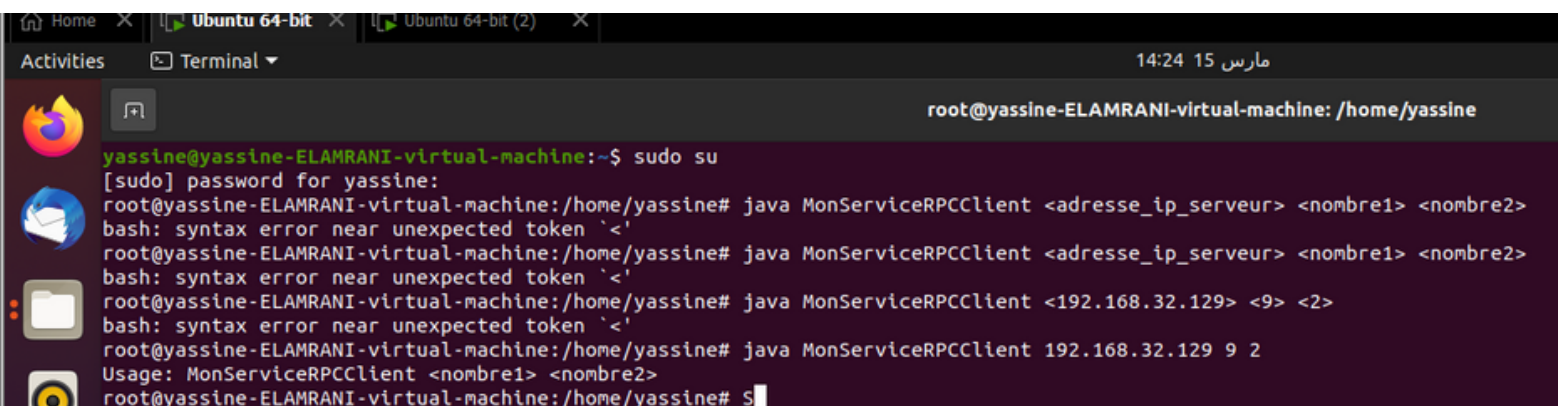


A terminal window showing the execution of the server. The prompt is 'root@yassine-elamrani-virtual-machine: /home/yassine#'. The user enters 'java MonServiceRPCServeur' and the output is 'Serveur RMI prêt.'

```
root@yassine-elamrani-virtual-machine: /home/yassine# java MonServiceRPCServeur  
Serveur RMI prêt.
```

Exécution du client :

```
java MonServiceRPCClient <adresse_ip_serveur> <nombre1> <nombre2>
```



A terminal window showing the execution of the client. The prompt is 'root@yassine-ELAMRANI-virtual-machine: /home/yassine#'. The user enters 'sudo su' and provides the password 'yassine'. Then, the user enters 'java MonServiceRPCClient <adresse_ip_serveur> <nombre1> <nombre2>' three times, each time getting a 'bash: syntax error near unexpected token `<'' error. Finally, the user enters 'java MonServiceRPCClient 192.168.32.129 9 2' and the output is 'Usage: MonServiceRPCClient <nombre1> <nombre2>'. The user then enters 'S'.

```
root@yassine-ELAMRANI-virtual-machine: /home/yassine# sudo su  
[sudo] password for yassine:  
root@yassine-ELAMRANI-virtual-machine: /home/yassine# java MonServiceRPCClient <adresse_ip_serveur> <nombre1> <nombre2>  
bash: syntax error near unexpected token `<'  
root@yassine-ELAMRANI-virtual-machine: /home/yassine# java MonServiceRPCClient <adresse_ip_serveur> <nombre1> <nombre2>  
bash: syntax error near unexpected token `<'  
root@yassine-ELAMRANI-virtual-machine: /home/yassine# java MonServiceRPCClient <192.168.32.129> <9> <2>  
bash: syntax error near unexpected token `<'  
root@yassine-ELAMRANI-virtual-machine: /home/yassine# java MonServiceRPCClient 192.168.32.129 9 2  
Usage: MonServiceRPCClient <nombre1> <nombre2>  
root@yassine-ELAMRANI-virtual-machine: /home/yassine# S
```