

---

***Applications Réparties :***  
***Rapport sur***  
***"Système d'Enchères Java RMI : Plateforme***  
***Interactive d'Enchères en Ligne"***

---



**RPC REMOTE PROCEDURE CALL**

Sous la supervision de  
***MR. AMAMOU Ahmed***

**LAOULIDI LAHKIM Ahmed**  
**EL AMRANI Yassine**

---

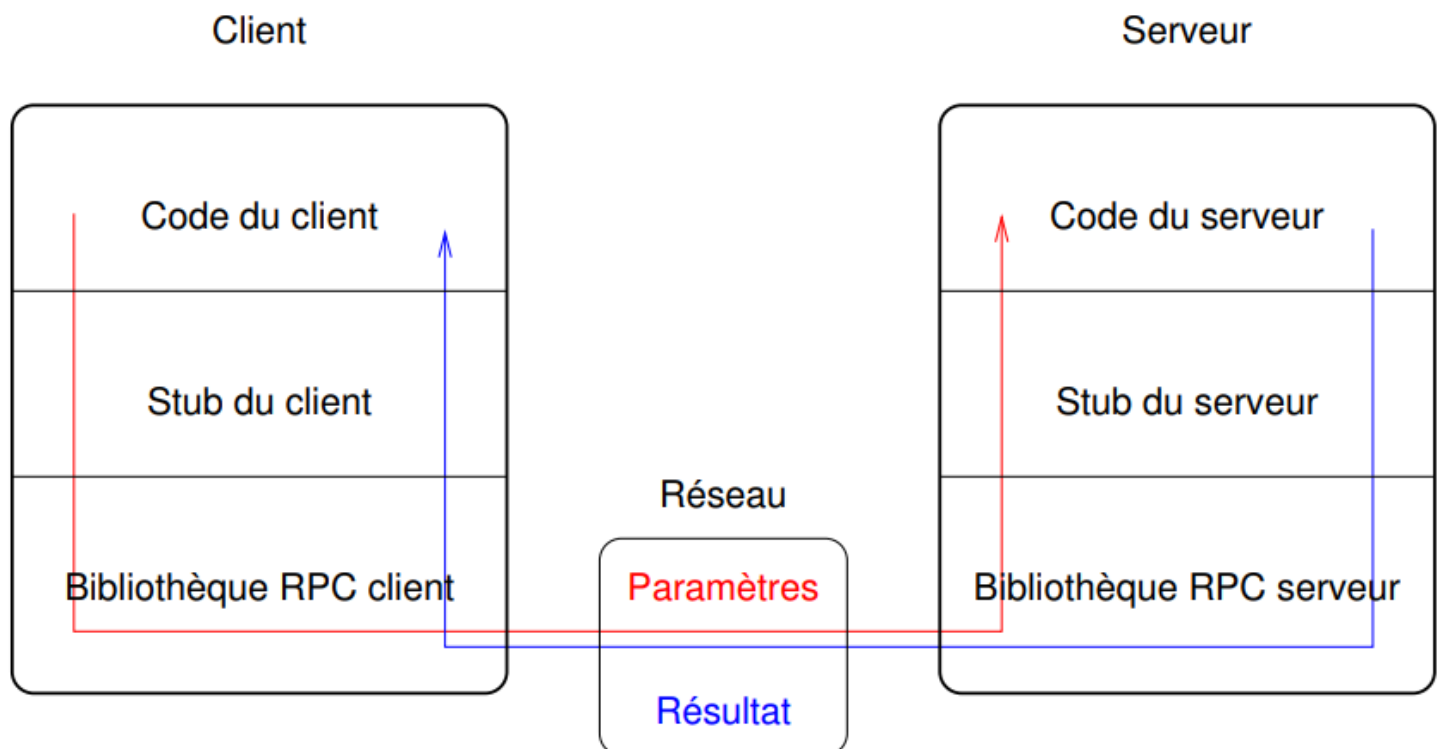
## SOMMAIRE

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 :**

Le code fourni implémente un système d'enchères en ligne à l'aide de Java RMI (Remote Method Invocation). L'objectif principal est de fournir une plateforme permettant aux utilisateurs d'enchérir sur des produits spécifiques et de déterminer le gagnant de chaque enchère. Ce système vise à créer une expérience interactive et sécurisée pour les enchères en ligne.

### • **FONCTIONNEMENT DU SERVEUR :**

Le serveur est représenté par la classe MonServiceRPCServeur. Voici son fonctionnement détaillé :

#### **1. Initialisation :**

Le serveur est initialisé avec un produit spécifique et son prix initial. Cette information est stockée localement dans le serveur pour référence.

#### **2. Enregistrement :**

Le serveur est enregistré dans le registre RMI, rendant ses méthodes accessibles aux clients via des appels distants.

#### **3. Méthodes :**

- `afficher()` : Cette méthode permet de mettre à jour les informations sur le produit en cours d'enchère, telles que son nom et son prix initial.

- `enchere()` : Les clients utilisent cette méthode pour soumettre leurs enchères sur le produit en cours. Si l'enchère est supérieure au prix actuel, elle est acceptée et le prix actuel est mis à jour. De plus, le nom de l'utilisateur est enregistré en tant que meilleur offreur.

- `getPrixActuel()` : Cette méthode renvoie le prix actuel de l'enchère en cours.

- `getGagnant()` : Cette méthode renvoie le nom de l'enchérisseur ayant soumis l'offre la plus élevée.

- `getListeProduits()` : Cette méthode renvoie la liste des produits disponibles pour l'enchère, avec leurs prix initiaux.

## Partie Pratique

```

MonServiceRPCInterface.java  MonServiceRPCServeur.java  MonServiceRPCClient.java  MonServiceRPCClient2.java

1  import java.rmi.RemoteException;
2  import java.rmi.server.UnicastRemoteObject;
3  import java.util.HashMap;
4  import java.util.Map;
5  import java.util.concurrent.ExecutorService;
6  import java.util.concurrent.Executors;
7  import java.util.concurrent.TimeUnit;
8  import java.util.concurrent.atomic.AtomicInteger;
9
10 public class MonServiceRPCServeur extends UnicastRemoteObject implements MonServiceRPCInterface {
11     // Implémentation du serveur pour le service RPC
12
13     private String produit; 2 usages
14     private int prixInitial; 2 usages
15     private int prixActuel; 6 usages
16     private AtomicInteger userCounter; // Compteur pour générer des noms d'utilisateur uniques 2 usages
17     private String meilleurOffreur; // Nom de l'enchérisseur le plus offrant 4 usages
18     private Map<String, Integer> produits; // Liste des produits disponibles 3 usages
19     private ExecutorService executorService; 2 usages
20
21     // Constructeur du serveur
22     protected MonServiceRPCServeur(String produit, int prixInitial) throws RemoteException { 1 usage
23         super();
24         this.produit = produit;
25         this.prixInitial = prixInitial;
26         this.prixActuel = prixInitial;
27         this.userCounter = new AtomicInteger(1); // Initialiser le compteur d'utilisateur
28         this.meilleurOffreur = null; // Initialiser meilleurOffreur
29         this.produits = new HashMap<>();
30         this.produits.put(produit, prixInitial); // Ajouter le produit initial à la liste des produits disponibles
31         this.executorService = Executors.newSingleThreadExecutor();
32     }
33
34     // Méthode pour afficher le produit et son prix initial
35     @Override no usages
36     public void afficher(String produit, int prixInitial) throws RemoteException {
37         System.out.println("Produit : " + produit + ", Prix initial : " + prixInitial);
38         this.produit = produit;
39         this.prixInitial = prixInitial;
40         this.prixActuel = prixInitial;
41     }
42
43     // Méthode pour entrer le PRID (identifiant produit)
44     @Override no usages
45     public void entrerPRID(String utilisateur, int prid) throws RemoteException {
46         // Pas besoin d'implémenter cette méthode pour l'instant
47     }
48
49     // Méthode pour enchérir sur le produit
50     @Override 2 usages
51     public synchronized boolean enchere(String utilisateur, int prix) throws RemoteException {
52         if (prix > prixActuel) {
53             prixActuel = prix;
54             utilisateur = "utilisateur" + userCounter.getAndIncrement(); // Attribuer un nom d'utilisateur unique
55             meilleurOffreur = utilisateur; // Mettre à jour meilleurOffreur
56             executorService.submit(() -> {
57                 try {
58                     // Attendre 30 secondes avant d'annoncer le gagnant
59                     TimeUnit.SECONDS.sleep(30);
60                     annoncerGagnant();
61                 } catch (InterruptedException e) {
62                     e.printStackTrace();
63                 }
64             });
65         }
66     }

```



## Partie Pratique

```
63         }
64     });
65     System.out.println("Attendez : 30 secondes jusqu'à ce que d'autres clients soumettent leurs offres.");
66     return true;
67 }
68 return false;
69 }
70
71 // Méthode privée pour annoncer le gagnant
72 private synchronized void annoncerGagnant() { 1 usage
73     System.out.println("L'offre gagnante est de : " + prixActuel + " euros par " + meilleurOffreur);
74     // Logique supplémentaire pour notifier tous les clients de l'offre gagnante, si nécessaire
75 }
76
77 // Méthode pour récupérer le prix actuel
78 @Override 2 usages
79 public synchronized int getPrixActuel() throws RemoteException {
80     return prixActuel;
81 }
82
83 // Méthode pour récupérer le gagnant
84 @Override 2 usages
85 public synchronized String getGagnant() throws RemoteException {
86     return meilleurOffreur;
87 }
88
89 // Méthode pour récupérer la liste des produits disponibles
90 @Override 2 usages
91 public synchronized Map<String, Integer> getListeProduits() throws RemoteException {
92     return produits;
93 }
94
95 // Méthode principale pour exécuter le serveur
96 public static void main(String[] args) {
97     try {
98         // Création du serveur avec un produit et un prix initial
99         MonServiceRPCInterface monService = new MonServiceRPCServeur( produit: "test", prixInitial: 250);
100
101         // Démarrage du registre RMI sur le port 1099
102         java.rmi.registry.Registry registry = java.rmi.registry.LocateRegistry.createRegistry( port: 1099);
103
104         // Enregistrement du serveur dans le registre RMI
105         registry.rebind( name: "MonServiceRPC", monService);
106
107         System.out.println("Serveur RPC prêt.");
108     } catch (Exception e) {
109         e.printStackTrace();
110     }
111 }
112 }
113 }
```

## Partie Pratique

### • FONCTIONNEMENT DU CLIENT :

Le client est représenté par la classe MonServiceRPCClient. Voici son fonctionnement détaillé :

#### 1. Connexion au Serveur :

Le client établit une connexion avec le serveur en récupérant le registre RMI à partir de l'adresse locale et du port spécifié.

#### 2. Affichage des Produits :

Le client récupère la liste des produits disponibles à partir du serveur et les affiche avec leurs prix initiaux, permettant ainsi aux utilisateurs de choisir le produit sur lequel ils souhaitent enchérir.

#### 3. Choix et Enchère :

- Le client choisit un produit parmi ceux disponibles et entre son enchère. Avant de soumettre l'enchère, le client vérifie que son offre est supérieure au prix actuel de l'enchère.

#### 4. Attente :

Après avoir soumis son enchère, le client attend pendant une période spécifiée (30 secondes dans ce cas) pour permettre à d'autres utilisateurs de soumettre leurs offres.

#### 5. Affichage du Gagnant :

Une fois la période d'enchère terminée, le client récupère l'offre gagnante et affiche le prix final ainsi que le nom de l'enchérisseur gagnant.

```
1 import java.rmi.registry.LocateRegistry;
2 import java.rmi.registry.Registry;
3 import java.util.Map;
4 import java.util.Scanner;
5
6 public class MonServiceRPCClient {
7     // Classe pour le client du service RPC
8
9     public static void main(String[] args) {
10         try {
11             // Récupération du registre RMI à partir de l'adresse locale et du port 1099
12             Registry registry = LocateRegistry.getRegistry(host: "localhost", port: 1099);
13             MonServiceRPCInterface monService = (MonServiceRPCInterface) registry.lookup(name: "MonServiceRPC");
14
15             Scanner scanner = new Scanner(System.in);
```



## Partie Pratique

```

17 // Récupérer la liste des produits disponibles sur le serveur
18 Map<String, Integer> produits = monService.getListeProduits();
19
20 // Afficher les produits disponibles avec leurs prix initiaux
21 System.out.println("Produits disponibles pour l'enchère :");
22 for (Map.Entry<String, Integer> entry : produits.entrySet()) {
23     System.out.println(entry.getKey() + " : " + entry.getValue() + " euros");
24 }
25
26 // Demander à l'utilisateur de choisir un produit
27 System.out.print("Entrez le nom du produit sur lequel vous souhaitez enchérir : ");
28 String produitChoisi = scanner.nextLine();
29
30 // Vérifier si le produit choisi existe dans la liste des produits disponibles
31 if (produits.containsKey(produitChoisi)) {
32     int prixInitial = produits.get(produitChoisi);
33
34     boolean enchereAcceptee = false;
35     while (!enchereAcceptee) {
36         System.out.print("Entrez votre enchère : ");
37         int prixEnchere = scanner.nextInt();
38         enchereAcceptee = monService.enchere( utilisateur: "Utilisateur 1", prixEnchere);
39         if (!enchereAcceptee) {
40             System.out.println("Votre enchère doit être supérieure au prix actuel.");
41         }
42     }
43
44     // Affichage du message d'attente
45     System.out.println("Veuillez patienter : 30 secondes pour que d'autres clients soumettent leurs offres.");
46
47     // Attendre pendant 30 secondes
48     Thread.sleep( millis: 30000);
49
50     // Maintenant, nous récupérons l'offre gagnante du serveur
51     int prixActuel = monService.getPrixActuel();
52     String gagnant = monService.getGagnant();
53     System.out.println("L'offre gagnante est de : " + prixActuel + " euros par " + gagnant);
54 } else {
55     System.out.println("Le produit choisi n'est pas disponible pour l'enchère.");
56 }
57
58 scanner.close(); // Fermer le scanner après utilisation
59 } catch (Exception e) {
60     e.printStackTrace();
61 }
62 }

```

## Partie Pratique

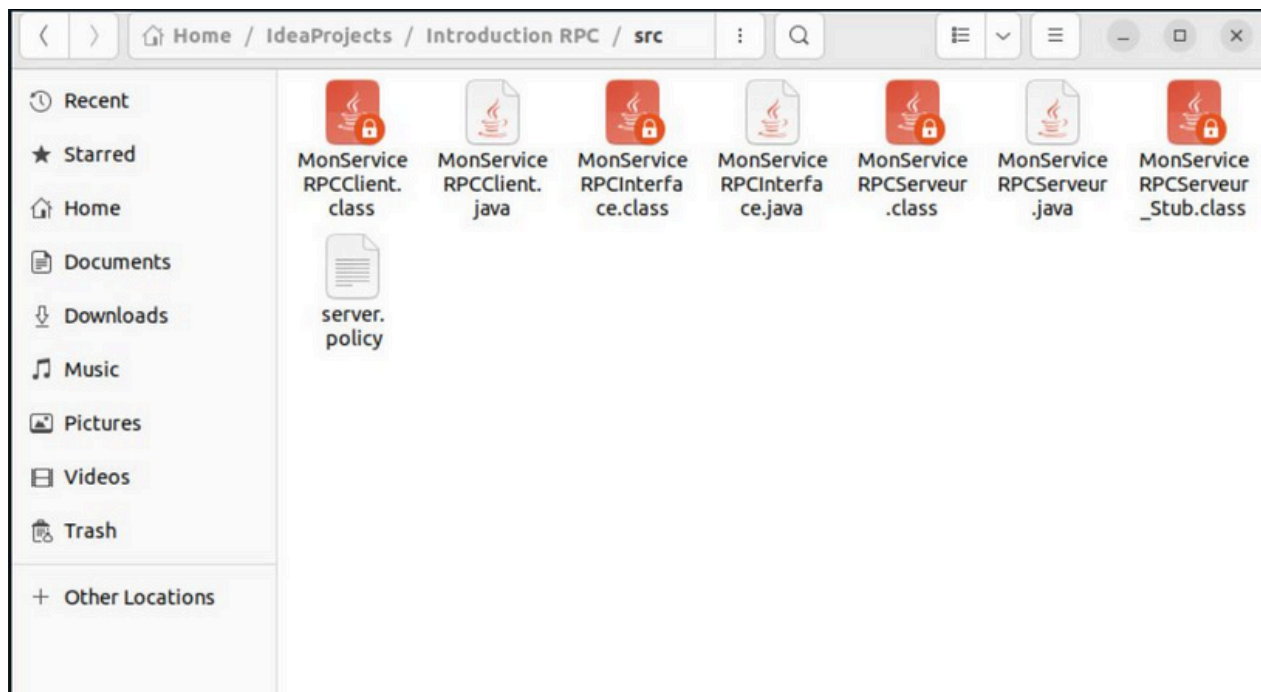
### Déploiement :

#### 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 JRMPI
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**

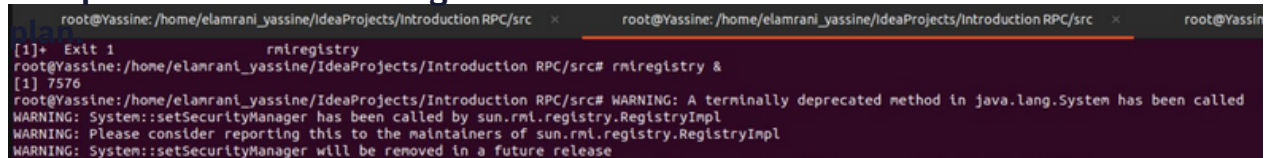
## Partie Pratique

### 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-**

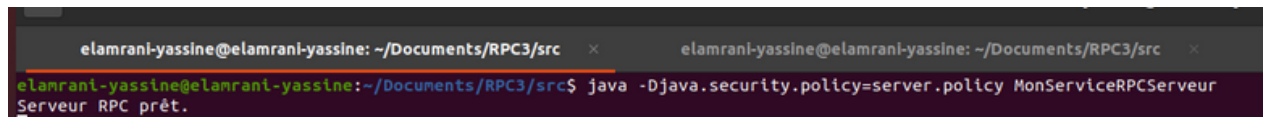


```
root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src x root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src x root@Yassin
[1]+ Exit 1 rmiregistry
root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction RPC/src# rmiregistry &
[1] 7576
root@Yassine: /home/elamrani_yassine/IdeaProjects/Introduction 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.

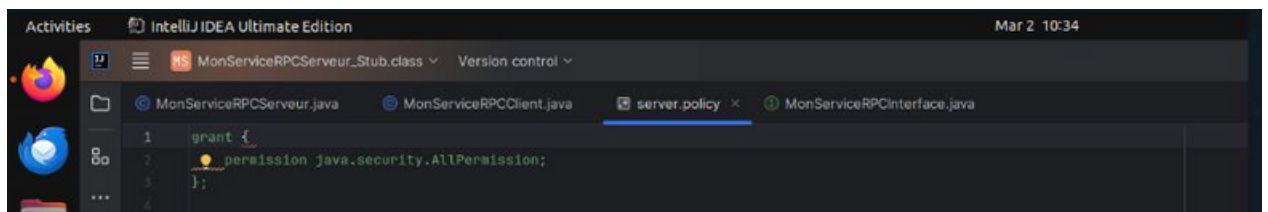
`java -Djava.security.policy=server.policy MonServiceRPCServeur`



```
elamrani-yassine@elamrani-yassine: ~/Documents/RPC3/src x elamrani-yassine@elamrani-yassine: ~/Documents/RPC3/src x
elamrani-yassine@elamrani-yassine: ~/Documents/RPC3/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 clients :

- ☐ Ouvrez un troisième terminal et démarrez votre client.

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

- ☐ premier cleint

```
elanrani-yassine@elanrani-yassine:~/Documents/RPC3/src$ java -Djava.security.policy=client.policy MonServiceRPCClient
Produits disponibles pour l'enchère :
test : 250 euros
Entrez le nom du produit sur lequel vous souhaitez enchérir : test
Entrez votre enchère : 270
Veuillez patienter : 30 secondes pour que d'autres clients soumettent leurs offres.
L'offre gagnante est de : 280 euros par utilisateur2
elanrani-yassine@elanrani-yassine:~/Documents/RPC3/src$
```

### 6. Exécution du clients :

- ☐ Ouvrez un catriem terminal et démarrez votre client.

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

- ☐ deusiemcleint

```
elanrani-yassine@elanrani-yassine:~/Documents/RPC3/src$ java -Djava.security.policy=client.policy MonServiceRPCClient
Produits disponibles pour l'enchère :
test : 250 euros
Entrez le nom du produit sur lequel vous souhaitez enchérir : test
Entrez votre enchère : 280
Veuillez patienter : 30 secondes pour que d'autres clients soumettent leurs offres.
L'offre gagnante est de : 280 euros par utilisateur2
elanrani-yassine@elanrani-yassine:~/Documents/RPC3/src$
```

### 7. Affichage du Gagnant :

- ☐ Une fois la période d'enchère terminée, le client récupère l'offre gagnante et affiche le prix final ainsi que le nom de l'enchérisseur gagnant.

```
elanrani-yassine@elanrani-yassine:~/Documents/RPC3/src$ java -Djava.security.policy=server.policy MonServiceRPCServeur
Serveur RPC prêt.
Attendez : 30 secondes jusqu'à ce que d'autres clients soumettent leurs offres.
Attendez : 30 secondes jusqu'à ce que d'autres clients soumettent leurs offres.
L'offre gagnante est de : 280 euros par utilisateur2
L'offre gagnante est de : 280 euros par utilisateur2
```