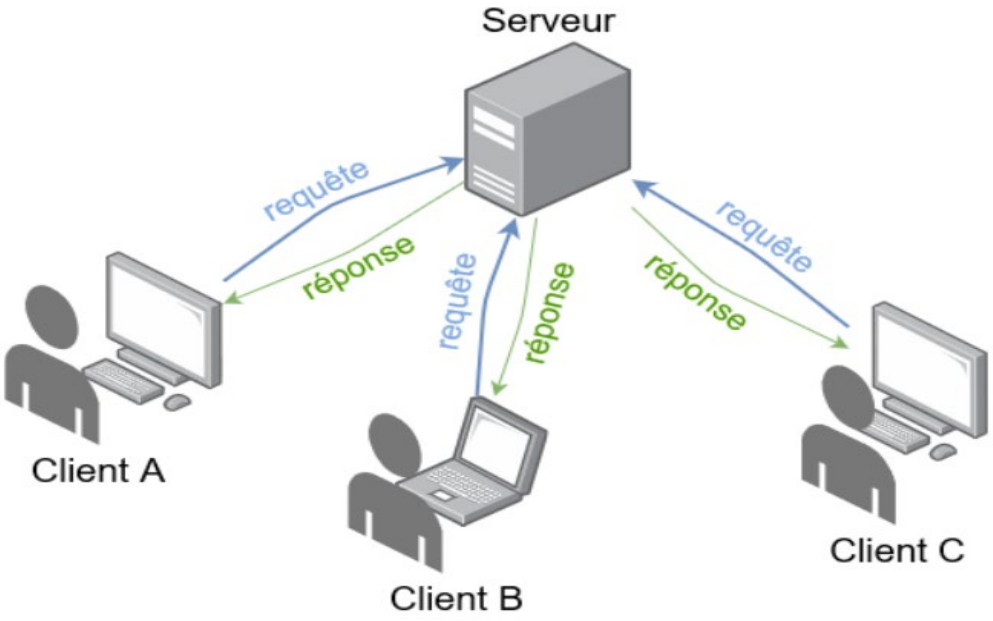
المدرسة العليا للتكنولوجيا
جامعة الحسن الثاني الدار البيضاء
Ecole Supérieure de Te hnologie
Université Hassan II de Casablanca

RAPPORT :

APPLICATION RESEAU D'ENCHERES

EN JAVA SOCKETS



Encadre par :

Dr M.ouzzif

Realise par :

El arrouchi fatima ezzahra

Introduction

Le développement d'applications utilisant des technologies réseau, comme les sockets, est crucial dans les environnements nécessitant une communication rapide et fiable entre clients et serveurs. Ce projet, a pour objectif de concevoir une application d’enchères en ligne reposant sur une architecture client-serveur basée sur des sockets. Cette approche garantit une gestion réactive et efficace des enchères en temps réel, tout en permettant l’intégration de fonctionnalités telles que les mises à jour dynamiques, indispensables pour ce type de service. L’objectif principal est de créer une plateforme permettant aux utilisateurs de participer à des enchères en ligne en soumettant des offres et en suivant les enchères en direct. La solution devra répondre à des exigences élevées en termes de fiabilité, de performance et de sécurité des échanges de données, tout en assurant une synchronisation précise des informations pour une expérience utilisateur optimale.

Chapitre 1

L’Architecture de l'Application :

○ Description de l'architecture client-serveur :

L'architecture client-serveur est un modèle de communication où des clients (utilisateurs ou applications) envoient des requêtes à un serveur centralisé, qui traite ces requêtes et renvoie les réponses appropriées. Dans le cadre de notre application d'enchères en ligne, cette architecture permet une gestion centralisée des enchères, assurant cohérence et synchronisation entre tous les participants.

Composants principaux :

- **Serveur :** Il centralise la logique de l'application, gère les enchères, maintient l'état actuel des offres et coordonne les interactions entre les différents clients. Le serveur est responsable de la gestion des connexions, de l'authentification des utilisateurs et de la diffusion des mises à jour en temps réel.
- **Clients :** Ce sont les interfaces utilisateur permettant aux participants de se connecter au serveur, de visualiser les enchères en cours, de placer des offres et de recevoir des mises à jour en temps réel. Les clients envoient des requêtes au serveur et affichent les réponses reçues.

○ Rôle des sockets dans l'application:

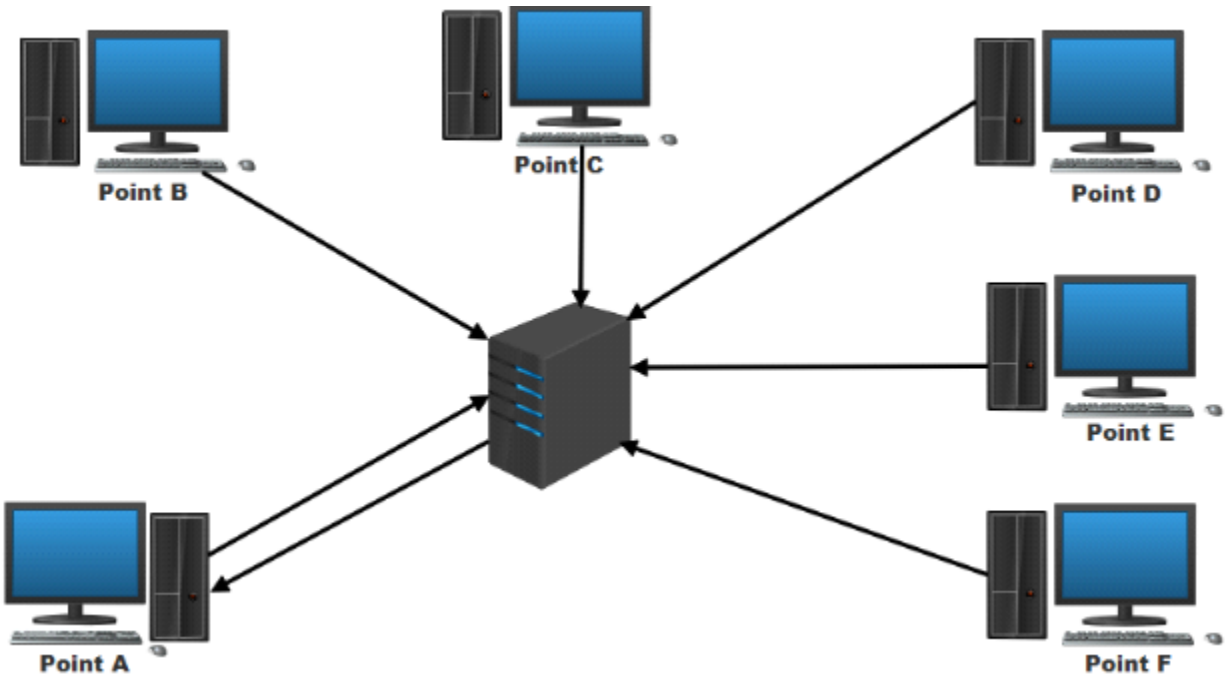
Les sockets sont des points de communication permettant l'échange de données entre le client et le serveur sur un réseau. Ils jouent un rôle crucial dans notre application en assurant une communication bidirectionnelle en temps réel, essentielle pour le fonctionnement fluide des enchères.

Fonctionnement des sockets :

- **Côté serveur :** Le serveur crée un socket d'écoute sur un port spécifique et attend les connexions entrantes des clients. Lorsqu'un client se connecte, le serveur accepte la connexion et établit un canal de communication dédié avec ce client.
- **Côté client :** Le client initialise une connexion vers le serveur en créant un socket et en se connectant au port sur lequel le serveur écoute. Une fois la

connexion établie, le client peut envoyer des requêtes et recevoir des réponses via ce socket.

- **Diagramme de l'architecture :**



Cette image met en évidence la centralisation du serveur, qui interagit directement avec les clients et la base de données, assurant la cohérence et la synchronisation des informations à travers l'application.

Chapitre 2

Spécifications Fonctionnelles et Techniques :

○ Fonctionnalités principales :

L'application d'enchères en ligne offre un éventail varié et complet de fonctionnalités essentielles, destinées à garantir une expérience utilisateur de qualité, en satisfaisant leurs exigences d'interaction, de gestion et de participation dans un cadre en ligne dynamique et intuitif :

- **Gestion des enchères en temps réel :**

Les utilisateurs ont la possibilité de prendre part à des enchères en temps réel avec des mises à jour instantanément pour tous les participants.

- **Établissement et administration des comptes utilisateurs :**

Chaque utilisateur peut créer un compte personnel, gérer ses informations, et accéder à l'historique de ses enchères.

- **System de notification:**

Des notifications en temps réel sont envoyées aux utilisateurs pour les informer des nouvelles enchères ou des mises concurrentes.

- **Historique et statistiques** :

Les utilisateurs peuvent consulter un résumé de leurs activités passées, notamment les enchères gagnées et perdues.

○ Contraintes et exigences réseau :

Afin d'assurer la performance, la fiabilité, et la réactivité de l'application, un ensemble précis de contraintes et d'exigences réseau a été soigneusement étudié et intégré. Ces exigences ont pour but de garantir un fonctionnement optimal même en cas de forte affluence, tout en permettant des échanges fluides, sûrs et efficaces entre les divers utilisateurs et le serveur central.

- **Bande passante :** L'application nécessite une bande passante adéquate pour traiter les échanges en temps réel, surtout durant les périodes d'intense activité.

- **Latence :** La latence doit être minimisée pour garantir que les mises et les notifications soient instantanées.
- **Disponibilité :** L'application doit être accessible 24h/24 et 7j/7.
- **Sécurité des données :** Des mécanismes de chiffrement sont nécessaires pour protéger les données sensibles des utilisateurs et des transactions.
- **Scalabilité :** L'application doit pouvoir gérer un nombre croissant d'utilisateurs et d'objets.

○ Protocoles utilisés (TCP/UDP, etc.) :

L'application se base sur une sélection stricte de protocoles de communication pour assurer des interactions fiables, efficaces et conformes aux besoins d'une application réseau contemporaine. Ces protocoles sont sélectionnés en tenant compte de leurs avantages spécifiques afin de satisfaire les divers besoins des fonctionnalités de l'application, qu'il s'agisse de sécurité, de rapidité ou de fiabilité des échanges :

- **Protocole TCP (Transmission Control Protocol)** :

  Ce protocole est essentiel dans l'infrastructure de l'application en garantissant des échanges essentiels. Il sert à la gestion des paris, à l'authentification des utilisateurs, et à d'autres interactions requérant une grande fiabilité. Grâce à son système de gestion du flux et de retransmission des paquets, TCP assure non seulement une livraison fiable et en ordre des messages, mais aussi l'identification et la correction des erreurs. Ceci en fait une option parfaite pour préserver l'intégrité des échanges entre le client et le serveur, même dans des situations réseau compliquées ou surchargées.

- **Protocole UDP (User Datagramme Protocol) :**

  Ce protocole a été sélectionné pour les fonctions requérant une réponse instantanée, comme les alertes en temps réel, où la vitesse est plus importante que la garantie totale de la transmission des informations. À la différence de TCP, UDP ne dispose pas de dispositifs de contrôle de flux ou de retransmission, ce qui diminue énormément la latence. Bien que cela puisse provoquer une perte de paquets dans des conditions réseau difficiles, l'avantage est sa capacité à transmettre rapidement des données, assurant une

expérience utilisateur fluide pour les mises à jour en temps réel et les alertes importantes.

- **Web Sockets** :

  Mise en place pour favoriser une communication continue bidirectionnelle entre le client et le serveur, ces protocoles occupent une place centrale dans l'architecture de l'application. Ils assurent une connexion ouverte et continue, évitant ainsi les échanges répétitifs requis avec des requêtes HTTP traditionnelles. Cela assure des actualisations instantanées pour les enchères, fournissant une expérience utilisateur fluide et dynamique. Par exemple, les participants d'une enchère peuvent immédiatement apercevoir les nouvelles offres ou modifications de statut, sans nécessiter de recharger la page ou d'attendre une réponse décalée du serveur.

Avantages et Inconvénients des Applications d'Enchères :

1) Avantages des Applications d'Enchère

Accessibilité et portée étendue :

Les applications d'enchères permettent une participation facile et flexible depuis n'importe quel endroit disposant d'une connexion Internet.

Contrairement aux enchères physiques nécessitant une présence sur place, les enchères en ligne éliminent les contraintes géographiques et temporelles.

Les participants peuvent enchérir depuis leur domicile, leur bureau ou même en déplacement.

Transparence dans les transactions:

Les enchères en ligne garantissent une visibilité des mises en temps réel, offrant ainsi une expérience équitable et honnête.

Les utilisateurs peuvent voir instantanément les montants proposés par d'autres participants.

Cela renforce leur confiance dans le processus, car il n'y a pas de place pour des pratiques malhonnêtes, comme des enchères cachées ou biaisées.

Optimisation des ventes pour le vendeur :

Les enchères permettent de maximiser les profits en obtenant le meilleur prix possible pour les produits.

Le format compétitif des enchères incite les acheteurs à surenchérir, augmentant ainsi progressivement le prix final. Cela est particulièrement avantageux pour des produits rares ou très demandés.

Interaction en temps réel :

Les utilisateurs peuvent participer activement et suivre les enchères en direct, ce qui rend l'expérience engageante et dynamique.

L'aspect en temps réel des enchères crée une sensation d'immédiateté et de suspense, encourageant les participants à rester engagés jusqu'à la fin.

Automatisation et simplicité :

Les applications d'enchères automatisent les processus, réduisant la complexité et le besoin d'intervention humaine.

Les systèmes calculent automatiquement les enchères les plus élevées, envoient des notifications aux utilisateurs et clôturent les enchères au moment prévu. Cela simplifie la gestion pour le vendeur et les participants.

Flexibilité des formats d'enchères :

Les plateformes d'enchères peuvent proposer une variété de formats pour répondre à des besoins spécifiques.

Chaque type d'enchère peut être adapté pour différents types de produits ou objectifs commerciaux. Par exemple :

Enchères classiques : Les utilisateurs surenchérissent jusqu'à ce qu'un gagnant soit déterminé.

Enchères inversées : Utilisées pour des services ou des contrats, où les fournisseurs proposent des prix décroissants.

Enchères silencieuses : Les utilisateurs soumettent leurs offres sans voir celles des autres, augmentant le suspense et la compétition.

2) Inconvénients des Applications d'Enchères:

Risques de Sécurité:

Sans mesures de sécurité adéquates (comme l'authentification ou le chiffrement), les données des utilisateurs et les transactions peuvent être vulnérables aux attaques telles que le piratage ou les intrusions.

Problèmes potentiels : Les attaques comme le vol de données (phishing), les interceptions de communications (MITM) ou les injections malveillantes (SQL injection) peuvent compromettre l'intégrité de l'application.

Dépendance à Internet:

Les utilisateurs doivent disposer d'une connexion stable pour participer, ce qui peut limiter l'accès dans les zones à faible connectivité.

Problèmes potentiels : Une connexion instable ou une panne d'Internet peut empêcher les utilisateurs de participer ou entraîner des déconnexions pendant une enchère cruciale.

Problèmes de scalabilité:

Avec un grand nombre d'utilisateurs, les applications peuvent rencontrer des problèmes de performance, tels que des temps de latence ou des interruptions de service.

Problèmes potentiels : Le serveur peut être surchargé, entraînant des temps de réponse lents ou des plantages.

Comportement frauduleux:

Certains utilisateurs peuvent exploiter les failles du système pour obtenir des avantages injustes.

Problèmes potentiels : Des faux comptes ou des "enchérisseurs fantômes" peuvent manipuler les prix pour créer une fausse impression de demande.

Expérience utilisateur limitée sans interface graphique :

Dans les applications simples basées sur des sockets et sur des lignes de commande peut ne pas convenir à tous les utilisateurs, en particulier les non-technophiles.

Problèmes potentiels : Les utilisateurs peuvent trouver l'application difficile à comprendre ou à utiliser sans une interface graphique intuitive

Complexité de la mise en œuvre de certaines fonctionnalités avancées :

Certaines fonctionnalités modernes, comme les paiements en ligne ou les notifications en temps réel, sont complexes à intégrer.

Problèmes potentiels : Ces fonctionnalités nécessitent des efforts supplémentaires en termes de développement et de maintenance.

Concurrence intense:

De nombreuses plateformes d'enchères existent déjà, ce qui rend difficile de se démarquer et d'attirer une base d'utilisateurs fidèle.

Chapitre 3

Implementation:

Outils et langages de programmation:

Outils utilisés :

- **Java SE Development Kit (JDK)** : Utilisé pour compiler et exécuter les programmes Java.
- **IDE (Integrated Development Environment)** : IntelliJ IDEA, faciliter le développement.

Langages de programmation :

- **Java** : Le langage principal pour implémenter la logique côté client et serveur, grâce à ses API robustes pour la gestion des sockets.
- **Protocoles réseau** : L'application utilise le protocole TCP/IP, ce qui garantit une connexion fiable entre le client et le serveur.

Gestion des sockets côté client et serveur :

L'application utilise les **sockets** pour permettre la communication entre le serveur et les clients. Voici comment cela fonctionne :

Côté Serveur :

Le serveur écoute les connexions entrantes via un ServerSocket sur un port spécifique (ici, le port **12345**). Lorsqu'un client se connecte, le serveur accepte la connexion et lui attribue un thread séparé pour gérer ses interactions. Cela permet une gestion simultanée de plusieurs clients.

Côté Client :

Le client se connecte au serveur en utilisant un objet Socket. Une fois connecté, il peut envoyer et recevoir des messages via les flux d'entrée/sortie (InputStream et OutputStream).

Gestion des sessions et des connexions multiples

Gestion des sessions :

Chaque client est traité comme une session unique par le serveur. Une session commence lorsqu'un client se connecte et se termine lorsqu'il se déconnecte. Le serveur gère la session en maintenant des références aux flux d'entrée/sortie du client.

Lorsqu'un client envoie un message (comme une enchère), le serveur analyse ce message et effectue les mises à jour nécessaires (par exemple, mettre à jour le montant le plus élevé et l'enchérisseur).

Gestion des connexions multiples :

Le serveur utilise une approche **multi-threading** pour gérer plusieurs connexions en parallèle. Chaque client est géré dans un thread séparé, permettant au serveur de rester réactif, même si plusieurs clients interagissent simultanément.

Exemple de gestion multi-thread :

```
while (true) {

    Socket clientSocket = serverSocket.accept(); // Accepte une connexion new
    ClientHandler(clientSocket).start(); // Lance un thread pour gérer ce client
```

Diffusion des messages :

Pour informer tous les clients des nouvelles enchères, le serveur conserve une liste des flux de sortie (PrintWriter) de chaque client. Lorsqu'un événement se produit (nouvelle enchère), le serveur diffuse ce message à tous les clients connectés.

Exemple de diffusion :

```
private void broadcast(String message) {
        for (PrintWriter writer : clientOutputs) {
         writer.println(message); }
 }
```

Synchronisation des enchères :

Le serveur vérifie que chaque enchère soumise est supérieure à l'enchère actuelle avant de l'accepter. Si une enchère est valide, elle est mise à jour et diffusée à tous les clients. Sinon, le client reçoit un message d'erreur.

AuctionServer.java:

Déclaration des variables globales

```
private static int highestBid = 0;
    private static String highestBidder = "";
    private static List<PrintWriter> clientOutputs = new ArrayList<>();
```

- **highestBid** : Garde une trace de l'enchère la plus élevée.
- **highestBidder** : Stocke le nom de la personne qui a placé cette enchère.
- **clientOutputs** : Permet de diffuser des messages à tous les clients connectés. Chaque client a un flux de sortie (PrintWriter) qui lui est associé.

Méthode principale (main)

```
 public static void main(String[] args) throws Exception {

        System.out.println("Le serveur d'enchères est en cours d'exécution...");
        ServerSocket serverSocket = new ServerSocket(12345);

        while (true) {
            Socket clientSocket = serverSocket.accept();
            new ClientHandler(clientSocket).start();
        }
    }
```

- **ServerSocket** : Ouvre un socket serveur sur le port 12345 pour écouter les connexions des clients.
- **serverSocket.accept()** : Attend qu'un client se connecte.
- **ClientHandler** : Une classe interne qui gère chaque client dans un thread séparé, permettant au serveur de traiter plusieurs connexions simultanément.
- **new ClientHandler(clientSocket).start()** : Démarre un nouveau thread pour gérer un client spécifique, ce qui évite que le serveur soit bloqué par une seule connexion.

Classe interne "ClientHandler":

```
static class ClientHandler extends Thread {
        private Socket socket;
        private PrintWriter out;
        private BufferedReader in;

        public ClientHandler(Socket socket) throws IOException {
            this.socket = socket;
            out = new PrintWriter(socket.getOutputStream(), true);
            in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            clientOutputs.add(out);
        }
```

- **socket** : Représente la connexion entre le serveur et ce client spécifique.
- **out** : Permet au serveur d'envoyer des messages au client.
- **in** : Permet au serveur de lire les messages envoyés par le client.
- **BufferedReader :** permet de lire efficacement des données ligne par ligne.
- **InputStreamReader :** convertit les données brutes d'un flux (bytes) en texte lisible (caractères).
- **getInputStream() :** récupère les données envoyées par le client via le socket.
- **clientOutputs.add(out)** : Stocke le flux de sortie pour permettre la diffusion des messages à ce client plus tard.

Méthode run:

```
public void run() {
    try {
        out.println("Bienvenue à l'enchère! Le prix de départ est de " + highestBid);
        String name = in.readLine();
        out.println("Bienvenue " + name + "!");

        String bid;
        while ((bid = in.readLine()) != null) {
            int newBid = Integer.parseInt(bid);
            if (newBid > highestBid) {
                highestBid = newBid;
                highestBidder = name;
                broadcast("Nouvelle enchère de " + highestBid + " par " + highestBidder);
            } else {
                out.println("Votre enchère doit être supérieure à " + highestBid);
            }
        }
    } catch (IOException e) {
        System.out.println("Erreur : " + e.getMessage());
    }
}
```

Message de bienvenue :

- Le serveur informe le client du prix de départ.
- Le client est invité à fournir son nom, qui est lu avec in.readLine().

Lecture des enchères :

- Une boucle lit les enchères envoyées par le client.
- Si l'enchère est supérieure à highestBid, elle devient la nouvelle enchère la plus élevée, et son enchérisseur devient le highestBidder.
- Le serveur diffuse un message à tous les clients pour informer de la nouvelle enchère.
- Si l'enchère est inférieure ou égale à highestBid, le client est informé que son enchère est invalide.

Gestion des erreurs :

- Si une erreur survient (par exemple, si le client se déconnecte), un message d'erreur est affiché sur le serveur.

Méthode broadcast

```
private void broadcast(String message) {
            for (PrintWriter writer : clientOutputs) {
                writer.println(message);
            }
        }
```

- Cette méthode envoie un message à tous les clients connectés.
- Elle est utilisée pour diffuser des informations importantes, comme la nouvelle enchère la plus élevée.

AuctionClient.java:

```
public class AuctionClient {
    Run | Debug
    public static void main(String[] args) throws Exception {
        Socket socket = new Socket(host:"localhost", port:12345);
        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), autoFlush:true);
        BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));

        System.out.println(in.readLine());  // Message de bienvenue
        System.out.print(s:"Entrez votre nom : ");
        String name = userInput.readLine();
        out.println(name);
```

- **Création d'un Socket** : Le client établit une connexion avec le serveur via un Socket. Ce socket se connecte à l'adresse **localhost** (c'est-à-dire l'ordinateur local où le serveur est en cours d'exécution) et au **port 12345**, qui est le port où le serveur écoute les connexions entrantes. Une fois la connexion établie, le client peut envoyer et recevoir des données via ce socket.
- **in (BufferedReader)** : Ce flux permet au client de lire les messages envoyés par le serveur. Le socket.getInputStream() récupère les données brutes envoyées par le serveur, et **InputStreamReader** les convertit en caractères lisibles par l'humain. Ensuite, **BufferedReader** permet de lire ces données ligne par ligne, ce qui est plus pratique pour gérer des échanges de texte.
- **out (PrintWriter)** : Ce flux permet au client d'envoyer des messages au serveur. Le socket.getOutputStream() récupère le flux de sortie du socket, et **PrintWriter** permet d'écrire des données texte dans ce flux. Le second paramètre (true) active l'auto-flush, c'est-à-dire que chaque fois qu'on appelle println(), les données sont envoyées immédiatement au serveur sans nécessiter un appel explicite à flush().

- **userInput (BufferedReader)** : Ce flux permet au client de lire les entrées de l'utilisateur depuis la console. **System.in** est le flux d'entrée standard pour lire les saisies de l'utilisateur à partir du clavier. Encore une fois, **BufferedReader** permet de lire les données de manière plus efficace.
- **Lecture du message de bienvenue** : Le client utilise **in.readLine()** pour lire une ligne envoyée par le serveur.

```
        new Thread(() -> {
            try {
                String serverMessage;
                while ((serverMessage = in.readLine()) != null) {
                    System.out.println(serverMessage);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }).start();

        String userBid;
        while ((userBid = userInput.readLine()) != null) {
            out.println(userBid);
        }
    }
}
```

- **Création d'un thread** : Le client crée un nouveau **Thread** qui permet d'écouter en permanence les messages envoyés par le serveur, sans bloquer l'exécution principale du programme. Le code qui suit est encapsulé dans une **lambda expression** (fonction anonyme) qui sera exécutée dans un nouveau thread.
- **Lecture des messages du serveur** :

  - **in.readLine()** : Cette méthode lit une ligne de texte envoyée par le serveur. Elle attend que le serveur envoie un message. Si ce message est disponible, il est récupéré et stocké dans la variable **serverMessage**.
  - **System.out.println(serverMessage)** : Chaque message reçu du serveur est ensuite affiché dans la console du client.

- **Boucle infinie** :
    - Le programme continue à écouter les messages du serveur tant que **in.readLine()** ne retourne pas null. Cela signifie que la connexion est toujours ouverte et que le serveur envoie des messages.
- **Gestion des erreurs** : En cas d'exception (par exemple, si la connexion est fermée ou si une erreur de lecture se produit), l'exception est capturée par le bloc catch, et l'erreur est imprimée dans la console avec **e.printStackTrace().**
- **Envoi de l'enchère au serveur** :
    - L'enchère est envoyée au serveur avec **out.println(userBid)**. Le **PrintWriter** avec auto-flush permet l'envoi immédiat des données.
- **Boucle infinie** :
    - La boucle continue tant que l'utilisateur saisit des enchères (si **userInput.readLine()** ne retourne pas null).
    - L'utilisateur peut continuer à saisir et envoyer des enchères au serveur sans interruption, jusqu'à ce qu'il ferme l'entrée.