Calcul concurrent d'une somme

Je vous propose ici un petit exercice basique de calcul concurrent de la somme d'une suite d'entiers en java RMI. La solution n'envisage que trois entiers mais elle est valable pour la somme de plusieurs entiers. Ce nombre dépend du nombre de clients connectés (nbre_Clients).
NB : Vous pourrez vous appuyer sur cet exercice pour implémenter le problème de deux joueurs.

Solution1 : Ici la concurrence d'accès est bloquante. Pour réaliser la somme de deux entiers il faut que deux clients se soient connectés pour transmettre un entier dans la ArrayList List. L'accès à cette liste est boquant et il est protégé par un synchronized (identique à un mutex). La méthode notify envoie le résultat de la somme à tous les clients.

// Serveur

```
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;

public class Serveur {

	public static void main(String[] args) {
		try {
			Integer port = Integer.parseInt(args[0]);
		String hote = args[1];


	LocateRegistry.createRegistry(port);
	Naming.rebind("rmi://" + hote + ":" + port +  "/sommebloquante", new
SommeBloquanteImpl());

	System.out.println("Server Calcul pret !");
	} catch (Exception e) {
	System.out.println("Server Calcul echec " + e);
	}
	}

		}
```

```
// Client
import java.rmi.Naming;
import java.util.InputMismatchException;
import java.util.Scanner;

public class Client {

	public static void main(String[] args) {
		Scanner sc = new Scanner (System.in) ;

		try {
			Integer port = Integer.parseInt(args[0]);
		String hote = args[1];

	SommeBloquanteInterface somme  =  (SommeBloquanteInterface)
Naming.lookup("rmi://" + hote + ":" + port +  "/sommebloquante");

while (true)
{
	int nb = 0;

	while (true)
	{System.out.println("nb=");
	try
	{ nb = sc.nextInt();
	break;
	}
	catch(InputMismatchException e)  {
		sc.nextLine();
	}		}

	System.out.println("La somme est :" + somme.add(nb));
	}
		}
catch(Exception e) {e.printStackTrace();}
		sc.close();
		System.exit(0);
	}}
```

```
import java.rmi.Remote;


// Interface de la somme concurrente
import java.rmi.RemoteException;

public interface SommeBloquanteInterface extends Remote{
	double add (double n) throws RemoteException;
	}

// Implémentation de la somme concurrente
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;

SommeBloquanteInterface {

	private static final long serialVersionUID = 1L;
	private static final int nb_Clients  = 3;
	private ArrayList<Double>  nombres = new ArrayList <> (nb_Clients);
	private double dernierRes =0;
	public SommeBloquanteImpl() throws RemoteException {
		super();
	}

	public double add(double nb)
		{
		Double res = addNombres (nb);
		if ( res ==null)
		{
		  try{
		 synchronized (nombres)
		   { nombres.wait(); }
	 } catch (InterruptedException e)
	{e.printStackTrace();
	return 0;
	}
		}
		return dernierRes;
		}
```

```
        private Double addNombres(double nb)
        {
        Double res = null;
        // semaphore sur la liste de nombre

        synchronized (nombres)
        {

        nombres.add(nb);
        if (nombres.size() >= nb_Clients)
        {
// res = nombres.stream().mapToDouble(n -> n).sum();
                double r = 0;
                for (int i = 0; i < nombres.size(); i++)
                        r = r+ nombres.get(i);
                res = new Double(r);
        dernierRes = res;
        nombres.clear();
        nombres.notifyAll ();
        }
        }
          return  res;  }
}
```