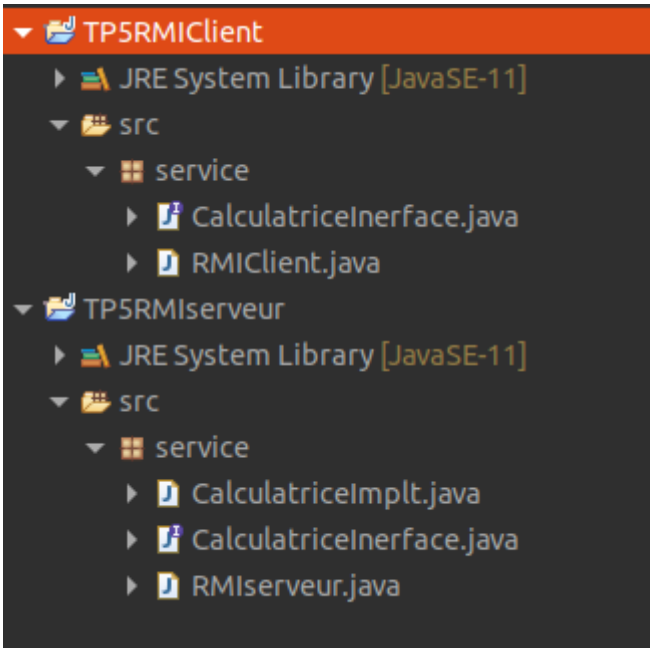
TP 5
Correction

La structure des fichiers des projets créés à l'aide d'Eclipse tout au long de ce TP est illustrée ci-dessous :



1. Côté serveur :

- CalculatriceInerface.java

CalculatriceInerface.java

```java
package service;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface CalculatriceInerface extends Remote{
    public double somme(double a,double b)throws RemoteException;
    public double soustraction(double a,double b)throws RemoteException;
    public double multiplication(double a,double b)throws RemoteException;
    public double division(double a,double b)throws RemoteException;
}
```

❖ CalculatriceImplt.java

CalculatriceImplt.java

```java
package service;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
@SuppressWarnings("serial")
public class CalculatriceImplt extends UnicastRemoteObject implements CalculatriceInerface{

    protected CalculatriceImplt() throws RemoteException {
        super();
    }

    @Override
    public double somme(double a, double b) throws RemoteException {
        return a+b;
    }
    @Override
    public double soustraction(double a, double b) throws RemoteException {
        return a-b;
    }
    @Override
    public double multiplication(double a, double b) throws RemoteException {
        return a*b;
    }
    @Override
    public double division(double a, double b) throws RemoteException {
        return a/b;
            }
```

❖ RMIserveur.java

RMIserveur.java

```java
package service;

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

public class RMIserveur {

    public static void main(String[] args) {

        try {

            LocateRegistry.createRegistry(1099);
            CalculatriceImplt objetDistant = new CalculatriceImplt();
            Naming.rebind("rmi://localhost:1099/calculatrice", objetDistant);
            System.out.println(objetDistant.toString());
            System.out.println("Le serveur est prêt !");

        } catch (Exception e) {
            System.out.println("Échec de l'exécution du serveur !" +e);
        }
    }

}
```

2. Côté client :

- ❖ CalculatriceInerface.java

CalculatriceInerface.java

```java
package service;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface CalculatriceInerface extends Remote{
    public double somme(double a,double b)throws RemoteException;
    public double soustraction(double a,double b)throws RemoteException;
    public double multiplication(double a,double b)throws RemoteException;
    public double division(double a,double b)throws RemoteException;
}
```

- ❖ RMIClient.java

- Lire une saisie clavier avec la classe **Scanner**
  - ➔ La variable nommée **operation** contient la chaine saisie par l'utilisateur.

```java
package service;

import java.rmi.Naming;
import service.CalculatriceInerface;
import java.util.Scanner;

public class RMIClient {

    public static void main(String[] args) {
        @SuppressWarnings("resource")
        Scanner saisirParClavier = new Scanner(System.in);

        try{
            CalculatriceInerface stub = (CalculatriceInerface) Naming.lookup("rmi://localhost:1099/calculatrice");
            System.out.println("Le client est connecté au serveur,saisissez l'opération comme suit : x opérateur y , exemple 5+5.");
            String operation  = saisirParClavier.nextLine();
```

- Il faut d'abord vérifier si **operation** contient un opérateur arithmétique (+, -, / ou *), puis le sauvegarder dans une variable nommée **operateur**.
  - ➔ La méthode **contains()** est utilisée pour rechercher une sous-chaîne dans une chaîne donnée. Si la sous-chaîne est trouvée dans la chaîne spécifiée, elle renvoie **True** ; sinon, il renvoie **False**.
  - ➔ Syntax: **Chaine.contains(SousChaine)**

```java
            char operateur;
            if (operation.contains("+")){operateur='+';}
            else if(operation.contains("-")){operateur='-';}
            else if(operation.contains("*")){operateur='*';}
            else {{operateur='/';}}
```

- Selon l'opérateur (la variable **operateur)** entré au clavier appelez la fonction correspondante.

```java
            switch(operateur)
            {
                case '+':
                {
                    System.out.println("Opération: "+operation);
                    System.out.print("Résultat : ");
                    System.out.println(stub.somme(Double.parseDouble(operation.substring(0,operation.indexOf(operateur))),
                            Double.parseDouble(operation.substring(operation.indexOf(operateur)+1))));
                    break;
                }
                   case '-':
                {
                    System.out.println("Opération: "+operation);
                    System.out.print("Résultat : ");
                    System.out.println(stub.soustraction(Double.parseDouble(operation.substring(0,operation.indexOf(operateur))),
                            Double.parseDouble(operation.substring(operation.indexOf(operateur)+1))));
                    break;
                }
```

```
                case '*':
            {
                System.out.println("Opération: "+operation);
                System.out.print("Résultat : ");
                System.out.println(stub.multiplication(Double.parseDouble(operation.substring(0,operation.indexOf(operateur))),
                        Double.parseDouble(operation.substring(operation.indexOf(operateur)+1))));
                break;
            }
               case '/':
            {
                System.out.println("Opération: "+operation);
                System.out.print("Résultat : ");
                System.out.println(stub.division(Double.parseDouble(operation.substring(0,operation.indexOf(operateur))),
                        Double.parseDouble(operation.substring(operation.indexOf(operateur)+1))));
                break;
            }
        }
    }catch (Exception e) {
        System.out.println("Opération incorrect--> "+e);
    }
  }
}
```

- ➔ System.out.println(stub.somme(Double.parseDouble(operation.substring(0,operation.indexOf(operateur))),Double.parseDouble(operation.substring(operation.indexOf(operateur)+1))));
- ➔ Double.parseDouble(String) : est une méthode de la classe Java **Double** intégrée à Java qui renvoie un nouveau double initialisé à la valeur représentée par la string spécifiée, elle permet donc de convertir la première partie de l'opération (**la variable operation** de type String **)** en **Double.**
- ➔ la première partie de l'opération est récupéré par operation.substring(0,operation.indexOf(operateur))
- ➔ La méthode **substring(int start, int end)** de la classe **String** renvoie une nouvelle chaîne qui est ***une sous-chaîne*** d'une chaîne donnée. La méthode Java String substring() est utilisée pour obtenir la sous-chaîne d'une chaîne donnée en fonction des index passés.
- ➔ Les index pour la première partie sont : 0 et operation.indexOf(operateur)
- ➔ operation.indexOf(operateur) :La méthode indexOf() renvoie la position de la première occurrence du ou des caractères spécifiés (un seul caractère **operateur** dans notre cas) dans une chaîne (la chiane **operation** dans notre exemple).
- ➔ la deuxième partie de l'opération est récupéré par: operation.substring(operation.indexOf(operateur)+1), <--> operation.substring(in start) ***la sous-chaîne*** commence par le caractère à l'index spécifié(indexOf(operateur)+1)et s'étend jusqu'à la fin de cette chaîne (**operation**) si **"int end**" n'est pas fourni comme argument.

- Exemple d'exécution:

```
Le client est connecté au serveur,saisissez l'opération comme suit : x opérateur y , exemple 5+5.
125.500/2.25
Opération: 125.500/2.25
Résultat : 55.77777777777778
```

```
Le client est connecté au serveur,saisissez l'opération comme suit : x opérateur y , exemple 5+5.
54**7
Opération: 54**7
Résultat : Opération incorrect--> java.lang.NumberFormatException: For input string: "*7"
```