



**Faculté des Sciences de Tunis**

Département Informatique

# IF4 – Mini Projet Systèmes Répartis
# Calcul de produit matriciel en mode réparti

**Proposé par :** Monsieur Islem Denden

**Réalisé par :**

- Hammami Ahmed (IF4 A)
- Lassoued Motez Belleh (IF4 A)

**Année universitaire : 2021 – 2022**

# Rapport du mini projet :

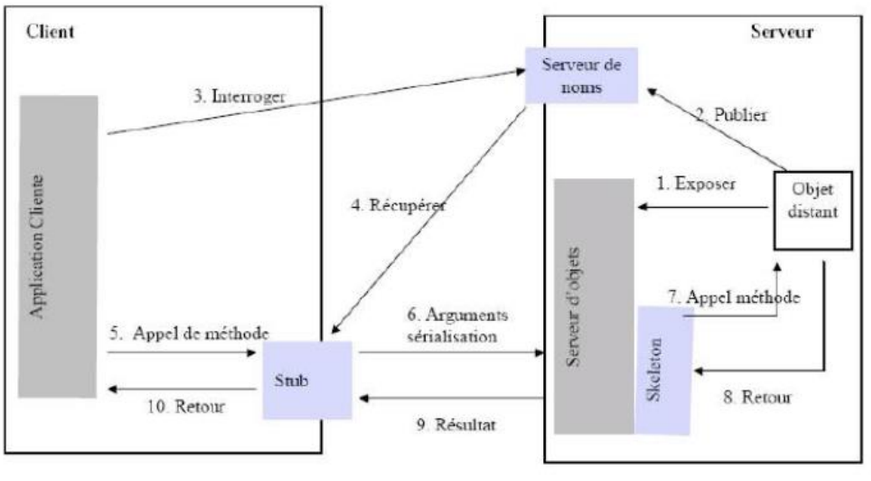
## Plan de ce mini projet :



## 1.Description du projet :

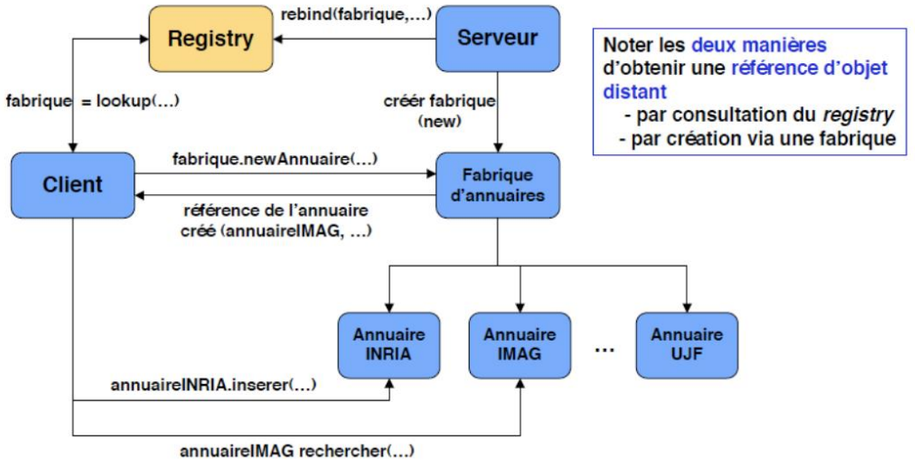
Ce projet consiste à développer un code Java pour un client et un serveur qui utiliseront le RMI afin de pouvoir communiquer ensemble. Le client va saisir deux matrices à multiplier. Il demandera par la suite au serveur de créer des objets RMI (au moins 4 objets) dont chacun prendra en charge une partie du calcul du produit matriciel au sein du serveur et renverra le résultat au client. Ce dernier devra consolider les données récupérées et les afficher à la fin de l'exécution.

## 2.Architecture des étapes d'un appel de méthode distante dans JAVA RMI :



# 3.Architecture de la fabrique d'objets (factory) :

Dans ce mini projet, nous avons fait recours à la création des objets RMI via des fabriques d'objets qui a cette architecture pour cet exemple :




# 4.Chargement dynamique :

- Les définitions de classes sont hébergées sur un serveur Web
- Les paramètres, les stubs sont envoyés au client via une connexion au serveur web
- Pour fonctionner, une application doit télécharger les fichiers de classe nécessaires

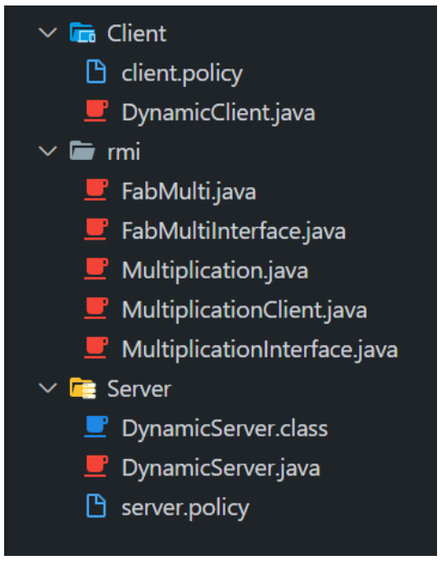
**Le Chargement dynamique nous donne l'opportunité à :**

- Éviter de disposer localement de toutes les définitions de classe.
- Les mêmes fichiers de classe (même version) sont partagés par tous les clients
- Charge une classe au besoin d'où il n'y a pas de gaspillage des ressources.

# 5.Algorithme d'un produit matriciel :

```
Pour i de 1 à taille faire
        Pour j de 1 à taille faire
                Pour k de 1 à taille faire
                        C[i,j]= C[i,j]+A[i,k]*B[k,j]
                Fin pour
        Fin pour
Fin pour
```

# 6.Répartition du projet :




Dans ce mini projet, nous allons utiliser le chargement dynamique comme nous avons dit précédemment c'est à dire que le client et le serveur communiquent à travers ce serveur web distant(rmi) :

- Tout d'abord on développe le code des interfaces et des classes qui correspondent aux :

**FabMulti :** Implémentation de la fabrique.
**FabMultiInterface :** Interface de la fabrique
**Multiplication :** Objet distant
**MultiplicationClient :** Client statique
**MultiplicationInterface :** Interface de l'objet distant

- Ces derniers sont mis sur un serveur web appelé dans notre projet (rmi).
- On développe le serveur dynamique :

**DynamicServer :** Implémentation du serveur dynamique.
**Server.policy:** Fichier de permission du serveur.

- Il nous reste que la partie Client qui va contenir

**DynamicClient:** Implémentation du client dynamique.
**Client.policy:** Fichier de permission du client.

# 7.Le code source des différentes entités :

**FabMultiInterface :** Interface de la fabrique

```java
import java.rmi.*;

public interface FabMultiInterface extends Remote {
  public MultiplicationInterface newMultiplication()
throws RemoteException;
}

```

**FabMulti :** Implémentation de la fabrique.

```java
import java.rmi.*;
import java.rmi.server.*;

public class FabMulti extends UnicastRemoteObject
implements FabMultiInterface {
  public FabMulti() throws RemoteException {}
  public MultiplicationInterface newMultiplication()
throws RemoteException {
    return new Multiplication();
  }
}

```

**MultiplicationInterface :** Interface de l'objet distant

```java
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface MultiplicationInterface extends Remote
{
  int[][] MultiplicationMatrice( int[][] m1, int[][] m2, int debI, int finI,
int debJ, int finJ, int comm) throws RemoteException;
}
```

**Multiplication :** Objet distant

```java
import java.rmi.*;
import java.rmi.server.*;
public class Multiplication extends
UnicastRemoteObject implements
MultiplicationInterface
{
       public Multiplication() throws RemoteException
       {
      super();
       }
  public  int[][] MultiplicationMatrice( int[][] A,
int[][] B, int debLig, int finLig, int debCol, int
finCol, int n)throws RemoteException {
    int [][]C = new int[finLig][finCol];
    for(int i=debLig; i<finLig; i++){
      for(int j=debCol; j<finCol; j++) {
        C[i][j] = 0;
        for (int k=0; k<n; k++) {
          C[i][j] += A[i][k] * B[k][j];
        }
      }

    }
    return C;
  }

}

```

**MultiplicationClient :** Client statique

```java
import java.rmi.registry.*;
import java.util.Scanner;

public class MultiplicationClient {

  //Affichage d'une matrice :
  public void afficher(int[][] M, int ligA, int colB) {
    for (int i = 0; i < ligA; i++) {
      for (int j = 0; j < colB; j++) {
        System.out.print(M[i][j] + "  ");
      }
      System.out.print("\n");
    }
  }

  //Remplissage d'une matrice à partir du clavier:
  public int[][] lire_matrice(int ligne, int colonne) {
    Scanner S = new Scanner(System.in);
    int i, j;
    int[][] M;


    M = new int[ligne][colonne];

    for (i = 0; i < M.length; i = i + 1) {
      for (j = 0; j < M[0].length; j = j + 1) {
        System.out.print(" [" + (i + 1) + "," + (j + 1) + "]:");
        M[i][j] = S.nextInt();
      }
    }
    return M;
  }
  //Création de la résultat à partir d'un produit par blocs
  public int[][] Rassamblage(int[][] a,int[][] b,int[][] c,int[][] d,int lig
,int col) {
        int[][] result = new int[lig][col];

        for (int i = 0; i < lig; i++) {
          for (int j = 0; j < col; j++) {
            if (i < lig / 2) {
              if (j < col / 2) {
                result[i][j] = a[i][j];
              } else {
                result[i][j] = b[i][j];
              }
            } else {
              if (j < col / 2) {
                result[i][j] = c[i][j];
              } else {
                result[i][j] = d[i][j];
              }
            }
          }
        }
        return result;
      }
  //programme principale
  public MultiplicationClient(String[] args) {
    try {

//Déclaration des matrices vides A et B et de leurs tailles (lignes, colonne
s)
      int[][] A;
      int[][] B;
      int ligA = 0, colA = 0, ligB = 0, colB = 0;
      //Saisie des tailles des matrices A et B
      Scanner S = new Scanner(System.in);

//La boucle while nous assure que les matrices ne sont pas nulles et que le
 nombre de colonne de A sont égaux aux nombres de lignes B pour faire la mul
tiplication
      while (ligA == 0 || colA == 0 || ligB == 0 || colB == 0 || colA !=
ligB) {
        System.out.print("Donner nombre de lignes de la matrice A: ");
        ligA = S.nextInt();
        System.out.print("Donner nombre de colonnes de la matrice A: ");
        colA = S.nextInt();
        System.out.print("Donner nombre de lignes de la matrice B: ");
        ligB = S.nextInt();
        System.out.print("Donner nombre de colonnes de la matrice A: ");
        colB = S.nextInt();
      }
      System.out.println("Remplir les élements de la matrice A");
      A = new int[ligA][colA];
      A = lire_matrice(ligA, colA);

      System.out.println("entrer les valeurs de la deuxieme matrice");
      B = new int[ligB][colB];
      B = lire_matrice(ligB, colB);

      //if(System.getSecurityManager() == null)
      //System.setSecurityManager(new RMISecurityManager());
      Registry reg = LocateRegistry.getRegistry("localhost", 1099);
      FabMultiInterface fabrique = (FabMultiInterface) reg.lookup("Fabrique"
);

      MultiplicationInterface Obj1;
      Obj1 = (MultiplicationInterface) fabrique.newMultiplication();
      MultiplicationInterface Obj2;
      Obj2 = (MultiplicationInterface) fabrique.newMultiplication();
      MultiplicationInterface Obj3;
      Obj3 = (MultiplicationInterface) fabrique.newMultiplication();
      MultiplicationInterface Obj4;
      Obj4 = (MultiplicationInterface) fabrique.newMultiplication();

      int[][] result1 = Obj1.MultiplicationMatrice(A,B,0,ligA/2,0,colB/2,
colA);
      int[][] result2 = Obj2.MultiplicationMatrice(A,B,0,ligA/2,colB/2,colB,
colA);
      int[][] result3 = Obj3.MultiplicationMatrice(A,B,ligA/2,ligA,0,colB/2,
colA);
      int[][] result4 = Obj4.MultiplicationMatrice(A,B,ligA/2,ligA,colB/2,
colB,colA);

      System.out.println(
        "Le resultat de multiplication des deux matrices est :"
      );
      //affichage de résultat
      afficher(Rassamblage(result1, result2, result3, result4, ligA, colB),
ligA,colB);
      //fermeture de Scanner
      S.close();
    } catch (Exception e) {
      System.out.println("Erreur d'acces a l'objet distant!!!");
      System.out.println(e.toString());
    }
  }
}
```

**DynamicServer :** Implémentation du serveur dynamique.

```java
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.RMIClassLoader;
import java.util.Properties;
public class DynamicServer {
  public static void main(String[] args) {
    try {
      System.setProperty("java.rmi.server.codebase", "file:../rmi/");

      if(System.getSecurityManager() == null){System.setSecurityManager(new
SecurityManager());}


          Registry registry = LocateRegistry.createRegistry(1099);
          System.out.println(
"Le serveur : Construction de l'implementation");
          System.out.println("L'objet Fabrique est  dans la RMIregistry");
    Properties p= System.getProperties();
    String url=p.getProperty("java.rmi.server.codebase");
    Class ClasseServeur = RMIClassLoader.loadClass(url,"FabMulti");

    registry.rebind("Fabrique",(Remote)ClasseServeur.getDeclaredConstructor
().newInstance());
          System.out.println ("Le serveur est pret.") ;
          System.out.println("Attente des invocations des clients ...");
        }
    catch (Exception e) {
          System.out.println("Erreur de liaison de l'objet Fabrique");
          System.out.println(e.toString());
          }
    }
}
```

**Server.policy:** Fichier de permission du serveur.

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

**DynamicClient:** Implémentation du client dynamique.

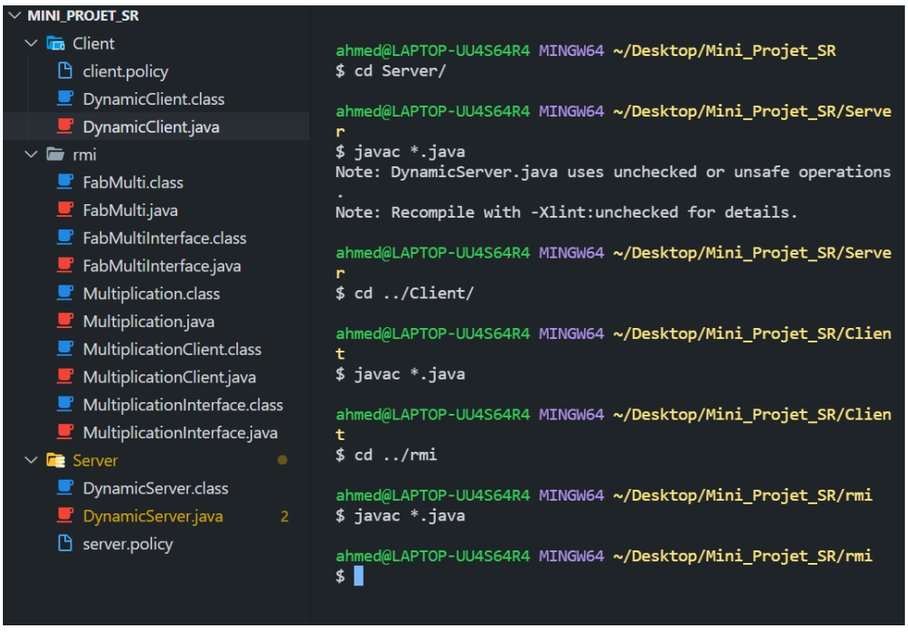
```
import java.lang.reflect.Constructor;
import java.rmi.server.RMIClassLoader;
import java.util.*;
import java.util.Properties;

public class DynamicClient {

  public DynamicClient(String[] args) throws Exception {
    System.setProperty("java.rmi.server.codebase", "file:../rmi/");
    Properties p = System.getProperties();
    String url = p.getProperty("java.rmi.server.codebase");
    Class ClasseClient = RMIClassLoader.loadClass(url,
"MultiplicationClient");

    Constructor[] C = ClasseClient.getConstructors();
    C[0].newInstance(new Object[] { args });
  }

  public static void main(String[] args) {
    System.setSecurityManager(new SecurityManager());
    try {
      DynamicClient cli = new DynamicClient(args);
    } catch (Exception e) {
      System.out.println(e.toString());
    }
  }
}
```

**Client.policy:** Fichier de permission du client.

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

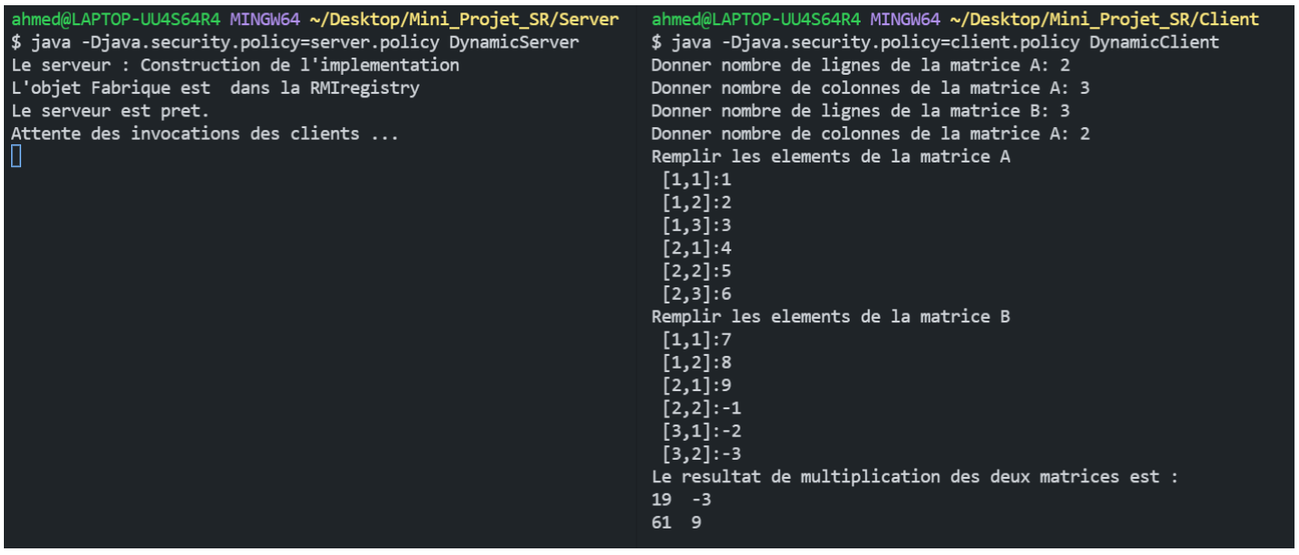
# 8.Exemples d’exécution :

On fait la compilation de tout les fichiers avac javac et on génère les fichiers .class



```
MINI_PROJET_SR
  Client
    client.policy
    DynamicClient.class
    DynamicClient.java
  rmi
    FabMulti.class
    FabMulti.java
    FabMultiInterface.class
    FabMultiInterface.java
    Multiplication.class
    Multiplication.java
    MultiplicationClient.class
    MultiplicationClient.java
    MultiplicationInterface.class
    MultiplicationInterface.java
  Server
    DynamicServer.class
    DynamicServer.java 2
    server.policy

ahmed@LAPTOP-UU4S64R4 MINGW64 ~/Desktop/Mini_Projet_SR
$ cd Server/

ahmed@LAPTOP-UU4S64R4 MINGW64 ~/Desktop/Mini_Projet_SR/Serve
r
$ javac *.java
Note: DynamicServer.java uses unchecked or unsafe operations
.
Note: Recompile with -Xlint:unchecked for details.

ahmed@LAPTOP-UU4S64R4 MINGW64 ~/Desktop/Mini_Projet_SR/Serve
r
$ cd ../Client/

ahmed@LAPTOP-UU4S64R4 MINGW64 ~/Desktop/Mini_Projet_SR/Clien
t
$ javac *.java

ahmed@LAPTOP-UU4S64R4 MINGW64 ~/Desktop/Mini_Projet_SR/Clien
t
$ cd ../rmi

ahmed@LAPTOP-UU4S64R4 MINGW64 ~/Desktop/Mini_Projet_SR/rmi
$ javac *.java

ahmed@LAPTOP-UU4S64R4 MINGW64 ~/Desktop/Mini_Projet_SR/rmi
$
```

On fait l’exécution du serveur et puis l’exécution du client et la capture ci-dessous illustre un petit exemple de produit matriciel (à gauche le serveur et à droite le client)




$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \times \begin{pmatrix} 7 & 8 \\ 9 & -1 \\ -2 & -3 \end{pmatrix} = \begin{pmatrix} 1\times 7+2\times 9-3\times 2 & 1\times 8-2\times 1-3\times 3 \\ 4\times 7+5\times 9-2\times 6 & 4\times 8-5\times 1-6\times 3 \end{pmatrix}$$

$$= \begin{pmatrix} 19 & -3 \\ 61 & 9 \end{pmatrix}$$