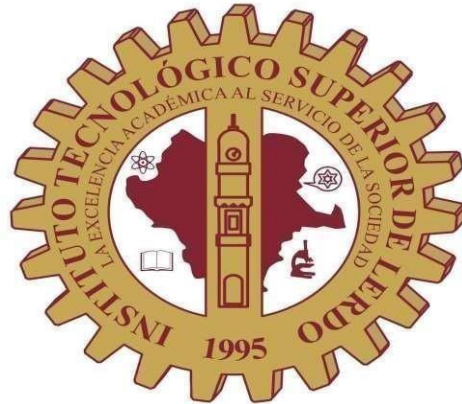


Instituto Tecnológico Superior de Lerdo



Nombre de la Materia:

Programación en Ambiente Cliente/Servidor

Profesor:

Jesús Salas Marín

Semestre:

VII

Proyecto Corte 3 RMI

Nombre del Alumno:

Jaime Francisco Moreno Lujan 182310299

Fecha de Entrega:

10/11/21

Introducción

RMI se caracteriza por la facilidad de su uso en la programación por estar específicamente diseñado para Java. A través de RMI, un programa Java puede exportar un objeto, con lo que dicho objeto estará accesible a través de la red y el programa permanece a la espera de peticiones en un puerto TCP. A partir de ese momento, un cliente puede conectarse e invocar los métodos proporcionados por el objeto. La invocación se compone de los siguientes pasos:

- Encapsulado (marshalling) de los parámetros (utilizando la funcionalidad de serialización de Java).
- Invocación del método (del cliente sobre el servidor). El invocador se queda esperando una respuesta.
- Al terminar la ejecución, el servidor serializa el valor de retorno (si lo hay) y lo envía al cliente.
- El código cliente recibe la respuesta y continúa como si la invocación hubiera sido local.

Toda aplicación RMI normalmente se descompone en 2 partes:

Un servidor, que crea algunos objetos remotos, crea referencias para hacerlos accesibles, y espera a que el cliente los invoque.

Un cliente, que obtiene una referencia a objetos remotos en el servidor, y los invoca.

Para este proyecto vamos a crear un cliente, un servidor y una interfaz entre el cliente y el servidor.

Objetivo

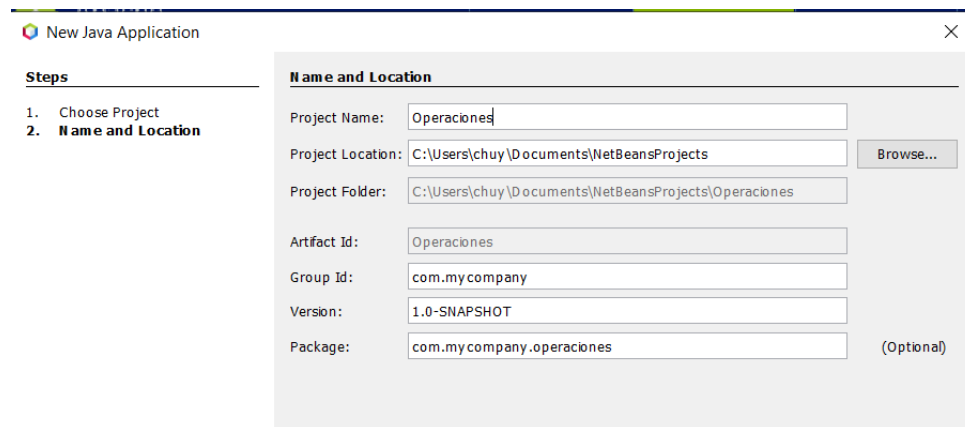
Crear un sistema aplicación RMI para poder acceder a los métodos necesarios para realizar aritmética con fracciones.

Proporcione métodos públicos que realicen cada una de las siguientes operaciones:

- a) Suma de dos números racionales.
- b) Resta de dos números racionales.
- c) Multiplicación de dos números racionales.
- d) División de dos números racionales.
- e) Mostrar números racionales en la forma a / b .
- f) Mostrar números racionales en formato de punto flotante.

Desarrollo

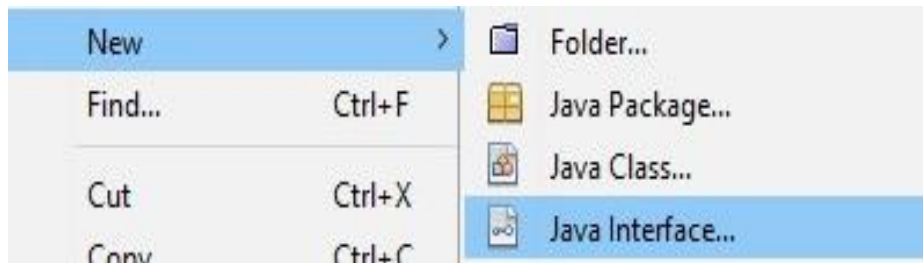
1. Crea un nuevo proyecto vacío en NetBeans, lo llamaremos Operaciones ya que la finalidad es reflejar las operaciones con fracciones mediante RMI



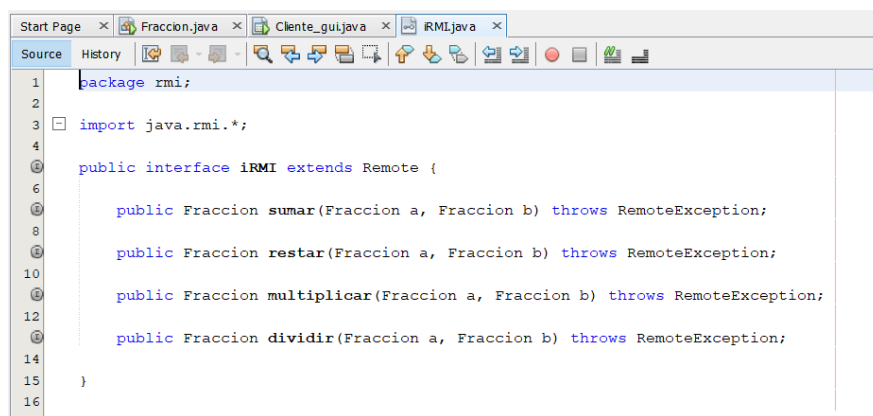
2. Agrega un nuevo paquete con el nombre rmi



3. Crea, en el nuevo paquete rmi, un nuevo archivo de tipo **Java Interface**, lo nombraremos iRMI

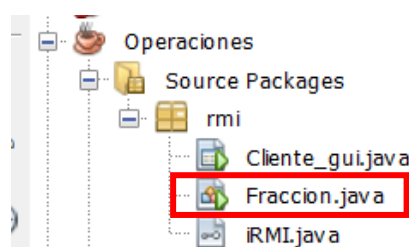


4. Dentro de el agregaremos el código donde haremos uso de los métodos que después codificaremos. Importante entender que esta es la parte fundamental del uso de RMI.



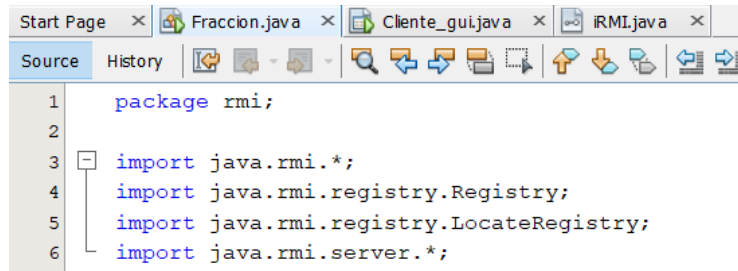
Lo que se muestra son 4 metodos que corresponden a las operaciones que nos pide el proyecto que realicemos. Se omite el punto e ya que por consecuencia, las fracciones ya se muestran del modo en que se pide en este inciso. Por otra parte, el punto f lo mostraremos de una forma simple en donde utilizaremos los parámetros que se ingresan desde la aplicación ya que se trata de un método menos laborioso que nos podemos ahorrar.

5. Agrega una clase con el nombre de Fracción



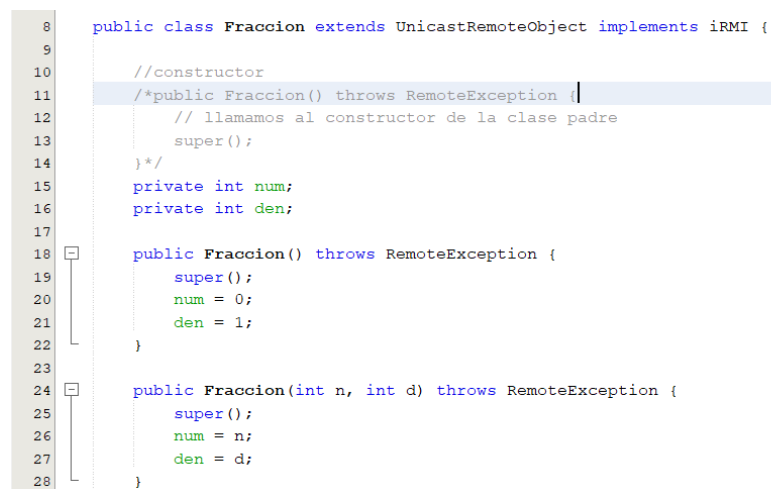
Codificamos:

Empezamos por la parte de las librerías rmi que nos ayudaran a usarlo dentro de todo el proyecto que estaremos realizando. Se puede omitir las ultimas 3 puesto que con la primera estamos importando todo lo que contiene rmi pero para mejores prácticas y para conocer específicamente lo que estamos utilizando, lo hacemos.



```
1 package rmi;
2
3 import java.rmi.*;
4 import java.rmi.registry.Registry;
5 import java.rmi.registry.LocateRegistry;
6 import java.rmi.server.*;
```

A partir de la línea 8 empezamos por codificar la clase Fracción que nos permitirá hacer uso de los métodos que creamos en ella. Esta línea se crea por default puesto que la creamos unos pasos atrás, lo que resta y que si tenemos que hacer son los constructores que definirán los parámetros que estaremos usando, que en este caso solo son 2, el numerador (num) y denominador (den). Las fracciones están formadas por estar dos partes, por eso la importancia de su implementación y definición. No olvidar que tenemos que llamar la conexión remota mediante RMI en cada constructor y método que se crea.



```
8 public class Fraccion extends UnicastRemoteObject implements iRMI {
9
10     //constructor
11     /*public Fraccion() throws RemoteException {
12         // llamamos al constructor de la clase padre
13         super();
14     }*/
15     private int num;
16     private int den;
17
18     public Fraccion() throws RemoteException {
19         super();
20         num = 0;
21         den = 1;
22     }
23
24     public Fraccion(int n, int d) throws RemoteException {
25         super();
26         num = n;
27         den = d;
28     }
29 }
```

Ahora vemos algunos de los métodos que estamos creando para la invocación mediante RMI que nos arrojará el resultado que nos pide el proyecto. Empezamos con el método sumar que de forma lineal estamos haciendo la operación correspondiente a la suma de fracciones, a continuación se muestran las formulas tradicionales para las operaciones correspondientes.

$\frac{a}{c} + \frac{b}{c} = \frac{a+b}{c}$	Suma de Fracciones homogéneas
$\frac{a}{c} + \frac{b}{d} = \frac{ad+bc}{cd}$	Suma de Fracciones heterogéneas
$\frac{a}{c} - \frac{b}{c} = \frac{a-b}{c}$	Resta de Fracciones homogéneas
$\frac{a}{c} - \frac{b}{d} = \frac{ad-bc}{cd}$	Resta de Fracciones heterogéneas
$\frac{a}{c} \cdot \frac{b}{d} = \frac{ab}{cd}$	Multipliación de Fracciones
$\frac{a}{c} : \frac{b}{d} = \frac{a}{c} \cdot \frac{d}{b} = \frac{ad}{cb}$	División de Fracciones

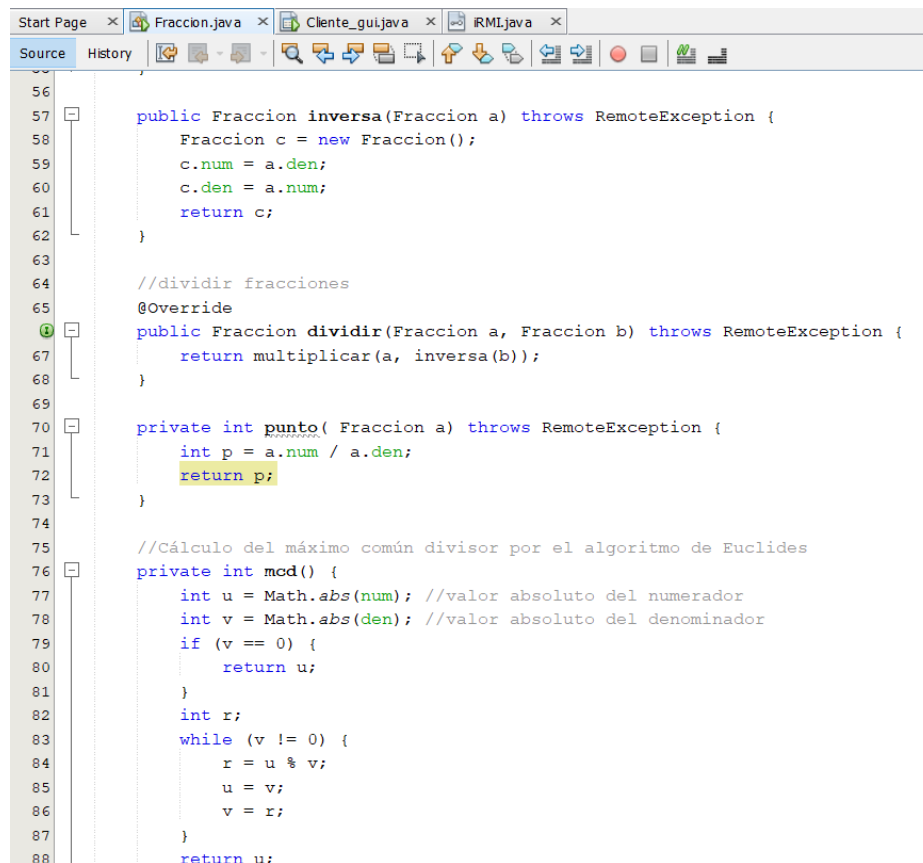
Cada método como ya se mencionó hace uso de la librería que hace conexión remota para el uso de RMI. @Override también es parte de la conexión que hacemos desde iRMI el cual está haciendo uso de estos métodos desde el archivo propio y que el servidor utilizara para la implementación de estos mismos métodos que ahora se están codificando. De no estar implementados des iRMI no será necesario agregar esta línea, pues no se está utilizando de manera remota. NetBeans nos da la ayuda en caso de olvidar esta parte y da la opción de hacerlo por nosotros.

```

31  @Override
32  ① public Fraccion sumar(Fraccion a, Fraccion b) throws RemoteException {
33      Fraccion c = new Fraccion();
34      c.num = a.num * b.den + b.num * a.den;
35      c.den = a.den * b.den;
36      return c;
37  }
38
39  //restar fracciones
40  @Override
41  ② public Fraccion restar(Fraccion a, Fraccion b) throws RemoteException {
42      Fraccion c = new Fraccion();
43      c.num = a.num * b.den - b.num * a.den;
44      c.den = a.den * b.den;
45      return c;
46  }
47
48  //multiplicar fracciones
49  @Override
50  ③ public Fraccion multiplicar(Fraccion a, Fraccion b) throws RemoteException {
51      Fraccion c = new Fraccion();
52      c.num = a.num * b.num;
53      c.den = a.den * b.den;
54      return c;
55  }

```

En la línea 57 se crea un método que será necesario para la división de la fracción, puesto que en la operación tradicional se hace un cruce entre numerador y denominador y este método nos ayuda a la realización de la operación dividir. Si nos fijamos, en este método inversa, no se hace uso de `@Override` por lo que comentamos anteriormente, no está siendo usado directamente por iRMI, de igual manera el método `mdc` que nos ayuda a sacar el máximo común divisor.



```
56
57 public Fraccion inversa(Fraccion a) throws RemoteException {
58     Fraccion c = new Fraccion();
59     c.num = a.den;
60     c.den = a.num;
61     return c;
62 }
63
64 //dividir fracciones
65 @Override
66 public Fraccion dividir(Fraccion a, Fraccion b) throws RemoteException {
67     return multiplicar(a, inversa(b));
68 }
69
70 private int punto(Fraccion a) throws RemoteException {
71     int p = a.num / a.den;
72     return p;
73 }
74
75 //Cálculo del máximo común divisor por el algoritmo de Euclides
76 private int mod() {
77     int u = Math.abs(num); //valor absoluto del numerador
78     int v = Math.abs(den); //valor absoluto del denominador
79     if (v == 0) {
80         return u;
81     }
82     int r;
83     while (v != 0) {
84         r = u % v;
85         u = v;
86         v = r;
87     }
88     return u;
```

También es importante mencionar algunas de las operaciones esta creando una nueva fracción (Fracción c) que nos permitirá devolver (`return c`) el resultado de la operación en sí.

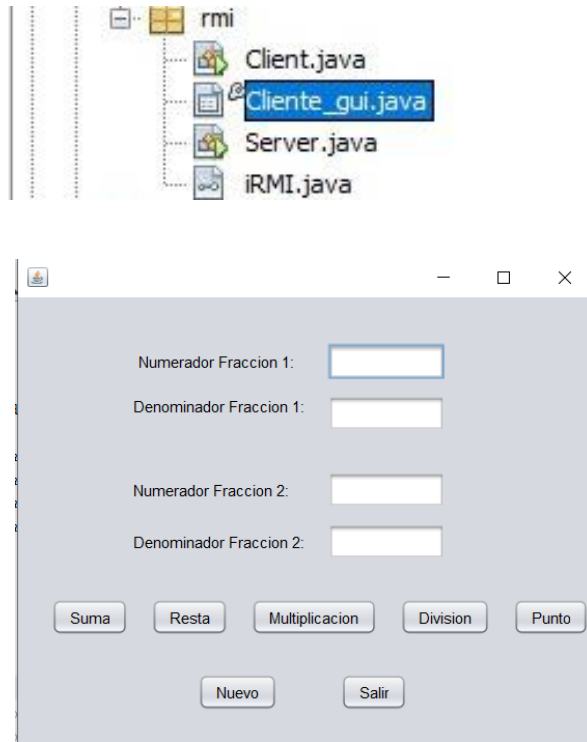
Por último, tenemos el método simplificar que es el que nos mostrara la fracción que se ingresara de una manera cruda en donde se pedirá el numerador y denominador por separado y que este método nos lo regresara de la manera numerador/denominador que conocemos comúnmente. Este método hace uso del método mcd que ya mencionamos anteriormente.

toString se utiliza para convertir a String (es decir, a una cadena de texto) cualquier objeto Java.

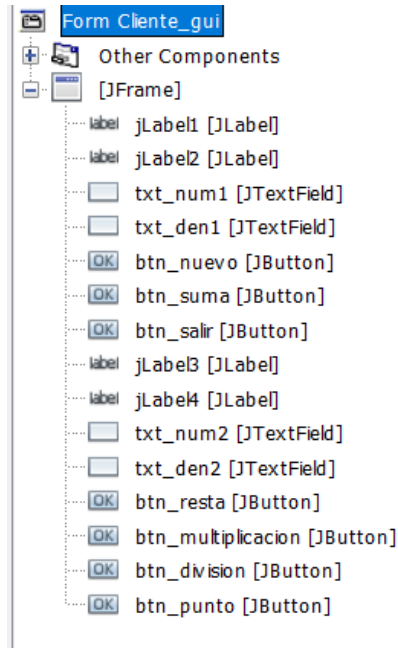
Para finalizar la clase, dentro del main hacemos todo lo que corresponde al servidor, como lo es el puerto, la manera en que se utilizara mediante un nombre y el mensaje que nos arroja para saber que el servicio esta activo. Todo esto tiene una estructura que utiliza las librerías que importamos desde el principio y que son fundamentales para el funcionamiento del sistema.

```
91 //método para simplificar fracciones
92 private void simplificar() {
93     int n = mcd(); //se calcula el mcd de la fracción
94     num = num / n;
95     den = den / n;
96 }
97
98 @Override
99 public String toString() {
100     simplificar();
101     return num + "/" + den;
102 }
103
104 /*
105  Se ejecuta el servidor desde aquí: main
106  Se crea primero el registro en un puerto específico vinculándolo
107  con el servidor.
108  Este vínculo debe tener un nombre que en este caso es "mates".
109
110  *Cuando el cliente llame al método remoto y haga la invocación
111  al registro, también deberá usar el mismo nombre y puerto.
112 */
113 public static void main(String[] args) {
114     try {
115         Registry registro = LocateRegistry.createRegistry(9999);
116         registro.rebind("mates", new Fraccion());
117         System.out.println("Servidor activo");
118     } catch (RemoteException ex) {
119         System.out.println(ex.getMessage());
120     }
121 }
```


6. Ahora hagamos un cliente visual.



Los nombres de variable que se utilizan deben ser específicos para cada una de las cajas de texto, como los botones, esto por que se utilizara para codificar el uso de estos mismo y para su función dentro del sistema. De aquí lo importate de la POO

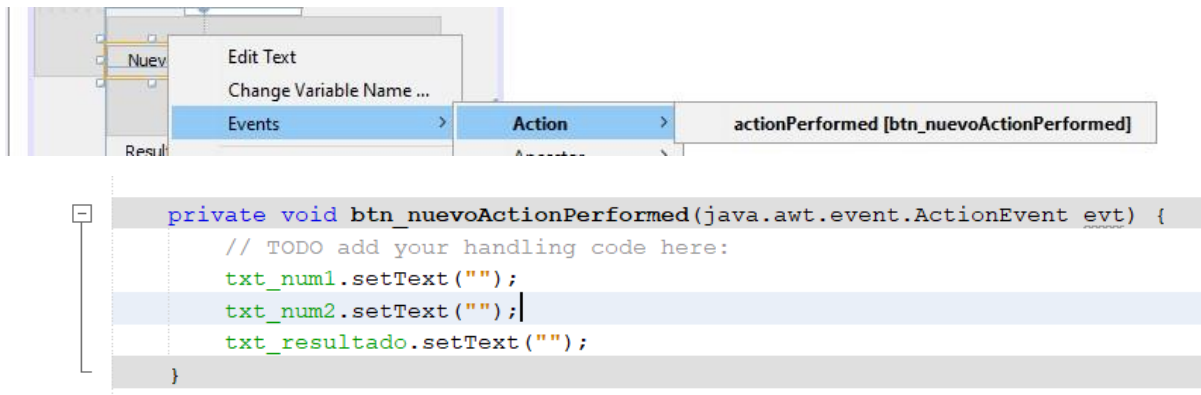


7. Codificamos la interfaz:

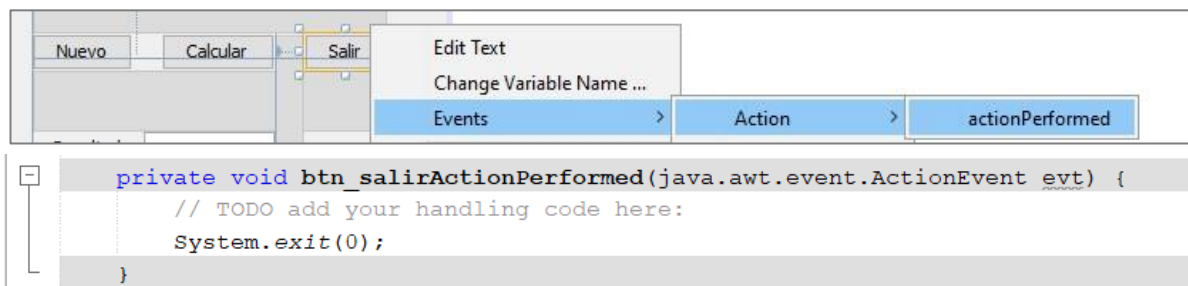
Las librerías a importar son las siguientes:

```
import java.rmi.NotBoundException;  
import java.rmi.RemoteException;  
import java.rmi.registry.*;  
import javax.swing.JOptionPane;
```

8.- Agrega el siguiente código al botón “Nuevo”. Son indicaciones simples para la funcionalidad de dicho botón. Aun no es trascendente dentro de lo que es RMI



9.- Agrega ahora el código para el botón “Salir”



10.- A partir de aquí entramos en los botones importantes, si recordamos, utilizaremos un botón para cada operación y cada operación está dentro de un método, los cuales utilizaremos mediante RMI y aquí es donde viene la codificación detallada ya que tendremos que concatenar y llenar los datos de la aplicación mediante los parámetros que se utilizan en las fracciones.

Empezando por la conexión y puerto necesarios para la aplicación, pasamos a la declaración de las variables que utilizaremos, en este caso son 4 ya que necesitamos numerador y denominador de la primera fracción y numerador y denominador para la segunda fracción, por lógica tendrán nombres diferentes (a, b, c, d). Cada una de las variables también corresponde a una caja de texto que tiene su nombre específico del que ya hablamos y que es importante en esta parte, donde ya la estamos utilizando para su funcionalidad. La manera de introducir es como se muestra.

Pasamos a la creación de los objetos Fracción que se crean de esa manera y en donde damos como parámetros, los datos ingresados en las cajas de texto que se utilizan mediante las variables ya mencionadas.

Ahora para la invocación de los métodos lo hacemos guardando dentro de la variable suma (que es una Fracción) el resultado que nos devuelve el método sumar. Este método sumar se concatena mediante la interfaz iRMI que ya se estableció en la línea 200. Si vemos el método sumar, nos pide 2 parámetros que son las dos fracciones ya creadas con sus atributos necesarios.

Para finalizar la codificación del botón sumar, mostramos los resultados con una ventana con el mensaje del resultado de la suma como se indica en la línea 217 y donde así mismo utilizamos las variables y los objetos ya creados

```
195 private void btn_sumaActionPerformed(java.awt.event.ActionEvent evt) {  
196     // TODO add your handling code here:  
197     try {  
198         Registry registro = LocateRegistry.getRegistry("127.0.0.1", 9999);  
199  
200         iRMI interfaz = (iRMI) registro.lookup("mates");  
201  
202         Fraccion suma;  
203  
204         int a, b, c, d;  
205  
206         a = Integer.parseInt(txt_num1.getText());  
207         b = Integer.parseInt(txt_den1.getText());  
208         c = Integer.parseInt(txt_num2.getText());  
209         d = Integer.parseInt(txt_den2.getText());  
210  
211         Fraccion f1 = new Fraccion (a, b); // Fracción 1/4  
212         Fraccion f2 = new Fraccion(b, c); // Fracción 1/2  
213  
214         suma = interfaz.sumar(f1, f2);  
215  
216         //System.out.println("La suma de:" + f1 + " + " + f2 + " es = a: " + suma);  
217         JOptionPane.showMessageDialog(this, "La suma de:" + f1 + " + " + f2 + " es = a: " + suma);  
218  
219     } catch (RemoteException | NotBoundException | NumberFormatException rex) {  
220         JOptionPane.showMessageDialog(this, rex.getMessage());  
221     }  
222 }
```

De la misma manera codificamos como corresponde cada botón

El botón Resta queda como se muestra

```
224 private void btn_restaActionPerformed(java.awt.event.ActionEvent evt) {  
225     try {  
226         Registry registro = LocateRegistry.getRegistry("127.0.0.1", 9999);  
227         iRMI interfaz = (iRMI) registro.lookup("mates");  
228         Fraccion resta;  
229         int a, b, c, d;  
230         a = Integer.parseInt(txt_num1.getText());  
231         b = Integer.parseInt(txt_den1.getText());  
232         c = Integer.parseInt(txt_num2.getText());  
233         d = Integer.parseInt(txt_den2.getText());  
234         Fraccion f1 = new Fraccion(a, b); // Fracción 1/4  
235         Fraccion f2 = new Fraccion(c, d); // Fracción 1/2  
236         resta = interfaz.restar(f1, f2);  
237         System.out.println("La resta de: " + f1 + " - " + f2 + " es = a: " + resta);  
238     } catch (RemoteException | NotBoundException | NumberFormatException rex) {  
239         JOptionPane.showMessageDialog(this, rex.getMessage());  
240     }  
241 }  
242
```

Botón Multiplicar

```
251 private void btn_multiplicacionActionPerformed(java.awt.event.ActionEvent evt) {  
252     try {  
253         Registry registro = LocateRegistry.getRegistry("127.0.0.1", 9999);  
254         iRMI interfaz = (iRMI) registro.lookup("mates");  
255         Fraccion multiplicacion;  
256         int a, b, c, d;  
257         a = Integer.parseInt(txt_num1.getText());  
258         b = Integer.parseInt(txt_den1.getText());  
259         c = Integer.parseInt(txt_num2.getText());  
260         d = Integer.parseInt(txt_den2.getText());  
261         Fraccion f1 = new Fraccion(a, b); // Fracción 1/4  
262         Fraccion f2 = new Fraccion(c, d); // Fracción 1/2  
263         multiplicacion = interfaz.multiplicar(f1, f2);  
264         System.out.println("La multiplicacion de: " + f1 + " * " + f2 + " es = a: " + multiplicacion);  
265         //JOptionPane.showMessageDialog(this, "La multiplicacion de: " + f1 + " * " + f2 + " es = a: " + multiplicacion);  
266     } catch (RemoteException | NotBoundException | NumberFormatException rex) {  
267         JOptionPane.showMessageDialog(this, rex.getMessage());  
268     }  
269 }  
270
```

Botón División

```
279 private void btn_divisionActionPerformed(java.awt.event.ActionEvent evt) {  
280     try {  
281         Registry registro = LocateRegistry.getRegistry("127.0.0.1", 9999);  
282  
283         iRMI interfaz = (iRMI) registro.lookup("mates");  
284  
285         Fraccion division;  
286  
287         int a, b, c, d;  
288  
289         a = Integer.parseInt(txt_num1.getText());  
290         b = Integer.parseInt(txt_den1.getText());  
291         c = Integer.parseInt(txt_num2.getText());  
292         d = Integer.parseInt(txt_den2.getText());  
293  
294         Fraccion f1 = new Fraccion(a, b); // Fracción 1/4  
295         Fraccion f2 = new Fraccion(c, d); // Fracción 1/2  
296  
297         division = interfaz.dividir(f1, f2);  
298  
299         System.out.println("La division de: " + f1 + " ÷ " + f2 + " es = a: " + division);  
300     } catch (RemoteException | NotBoundException | NumberFormatException rex) {  
301         JOptionPane.showMessageDialog(this, rex.getMessage());  
302     }  
303 }
```

El botón Punto nos da el resultado de una manera mas directa sin un método puesto que los datos necesarios nos lo dan la misma aplicación. Para esta operación solo basta hacer una división simple entre numerador y denominador. La operación la podemos establecer con los operadores necesarios y mostramos los resultados mediante variable que creamos para devolver el resultado.

```
305 private void btn_puntoActionPerformed(java.awt.event.ActionEvent evt) {  
306     try {  
307         Registry registro = LocateRegistry.getRegistry("127.0.0.1", 9999);  
308  
309         iRMI interfaz = (iRMI) registro.lookup("mates");  
310  
311         float punto1;  
312         float punto2;  
313  
314         int a, b, c, d;  
315  
316         a = Integer.parseInt(txt_num1.getText());  
317         b = Integer.parseInt(txt_den1.getText());  
318         c = Integer.parseInt(txt_num2.getText());  
319         d = Integer.parseInt(txt_den2.getText());  
320  
321         Fraccion f1 = new Fraccion(a, b); // Fracción 1/4  
322         Fraccion f2 = new Fraccion(c, d); // Fracción 1/2  
323  
324         punto1 = (a / b);  
325         punto2 = (c / d);  
326  
327         //System.out.println("La fraccion "+f1+" en punto flotante es igual a: "+ punto1);  
328         //System.out.println("La fraccion "+f2+" en punto flotante es igual a: "+ punto2);  
329  
330         JOptionPane.showMessageDialog(this, "La fraccion "+f1+" en punto flotante es igual a: "+ punto1);  
331         JOptionPane.showMessageDialog(this, "La fraccion "+f2+" en punto flotante es igual a: "+ punto2);  
332  
333     } catch (RemoteException | NotBoundException | NumberFormatException rex) {  
334         JOptionPane.showMessageDialog(this, rex.getMessage());  
335     }  
336 }
```

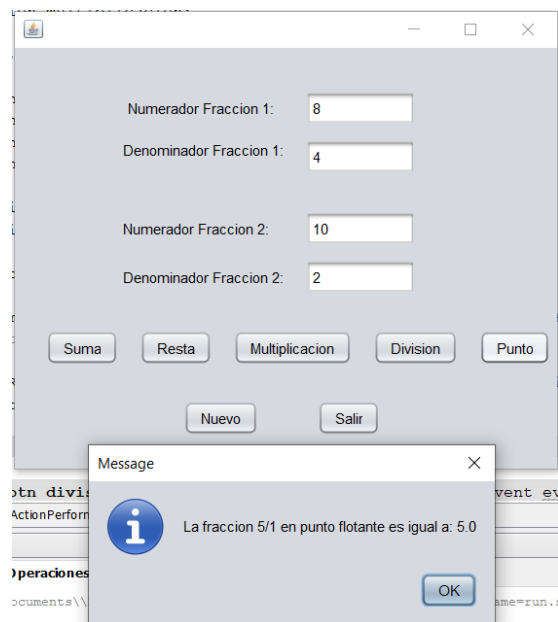
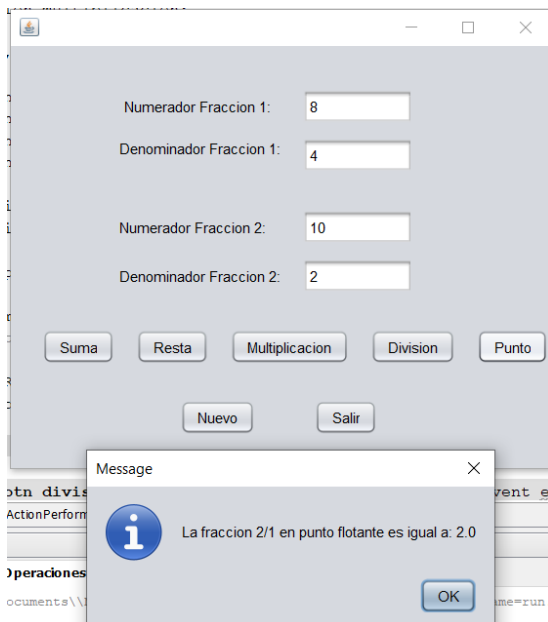
Pasamos a las pruebas, lo primero es ejecutar el archivo Fraccion.java que a su vez es el servidor y la clase que contiene los metodos a usar. Nos mostrara el mensaje de “Servidor activo” que dejamos para establecer que todo esta funcionando.

```
Output - Operaciones (run-singl... x
ant -f C:\Users\chuy\Documents\NetBeansProjects\Operaciones -Dnb.internal.action.name=run.single -Djava
init:
Deleting: C:\Users\chuy\Documents\NetBeansProjects\Operaciones\build\build-jar.properties
deps-jar:
Updating property file: C:\Users\chuy\Documents\NetBeansProjects\Operaciones\build\build-jar.properties
Compiling 1 source file to C:\Users\chuy\Documents\NetBeansProjects\Operaciones\build\classes
compile-single:
run-single:
Servidor activo
```

Ejecutamos el archivo Cliente_gui.java que es la aplicación en si donde se utiliza RMI y los metodos de la clase Fraccion.

Nos muestra la ventana con el fomrulario creado e ingresamos los datos que nos pide.

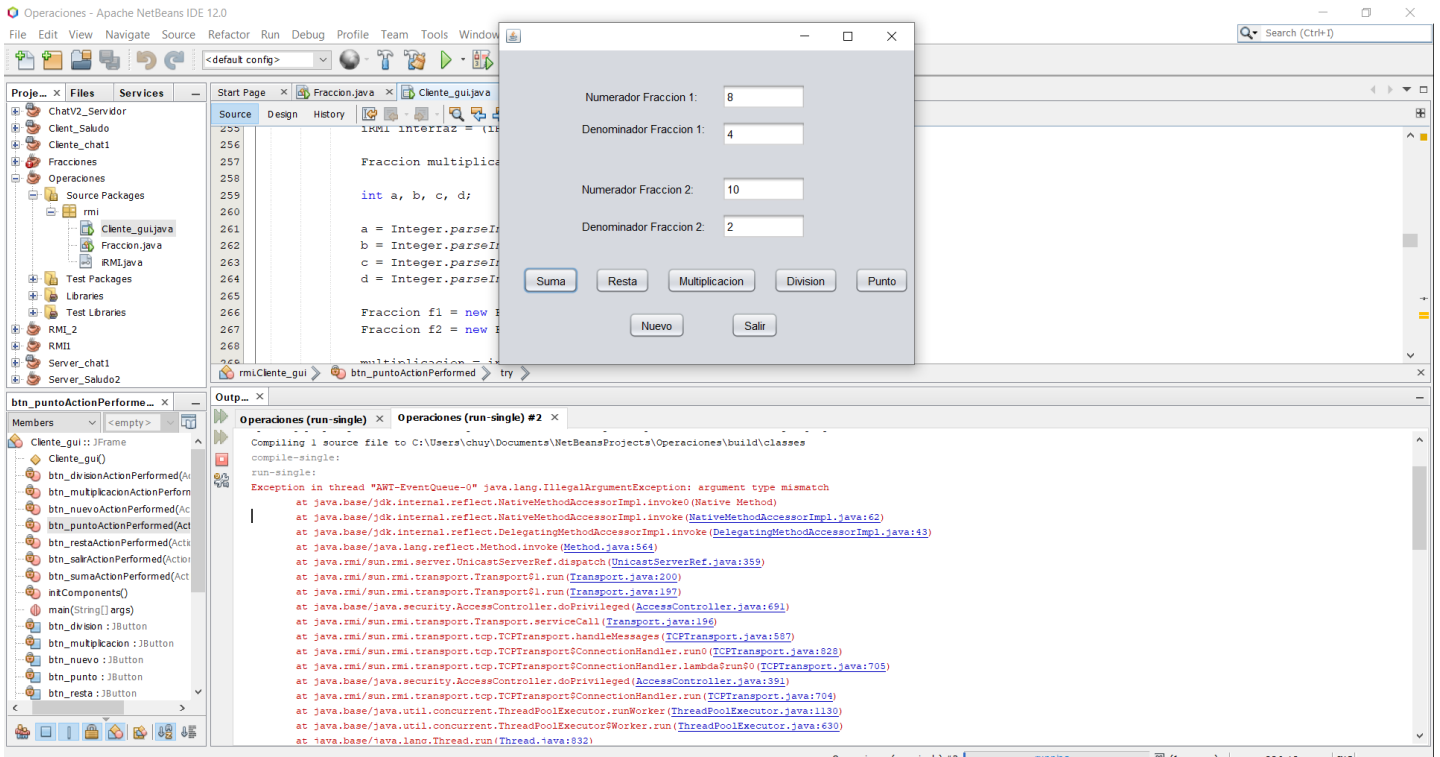
El primer boton y por pruebas antes ya hechas es el de punto, puesto que no utiliza metodo alguno mas que el de simplificar mediante mcd que nos da un equivalente de las fracciones y nos devuelve un resultado en punto flotante.



En cuanto a los botones correspondientes a las operaciones, nos arrojo errores que estan fuera de lo codificado, por lo que los resultados esperados no son lo que se requerian.

Una manera de probarlo es sin el RMI pero perderia el onjetivo.

Si vemos el codigo del cliente, es evidente que no existe error alguno por lo que se concluye que los metodos y la clase en si, no tiene algun problema



Conclusion:

La practica y el tema son claros y aunque los resultados no fueron los deseados, se entiende que el protocolo no es muy utilizado en la actualidad aunque sigue siendo de gran interes puesto que se ven protocolos para acceder a metodos independientes establecidos mediante un servidor. La conexi3n entre cliente y servidor lo pudimos hacer evidente en practicas anteriores de manera visual y en red LAN por lo que independientemente del resultado de este proyecto, fue muy interesante el desarrollo del tema en si en la practica.

