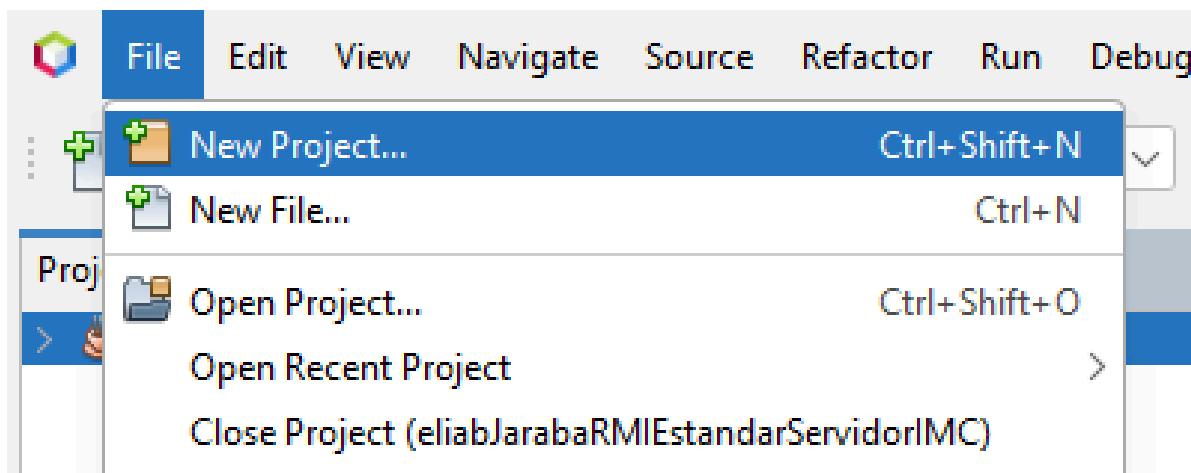


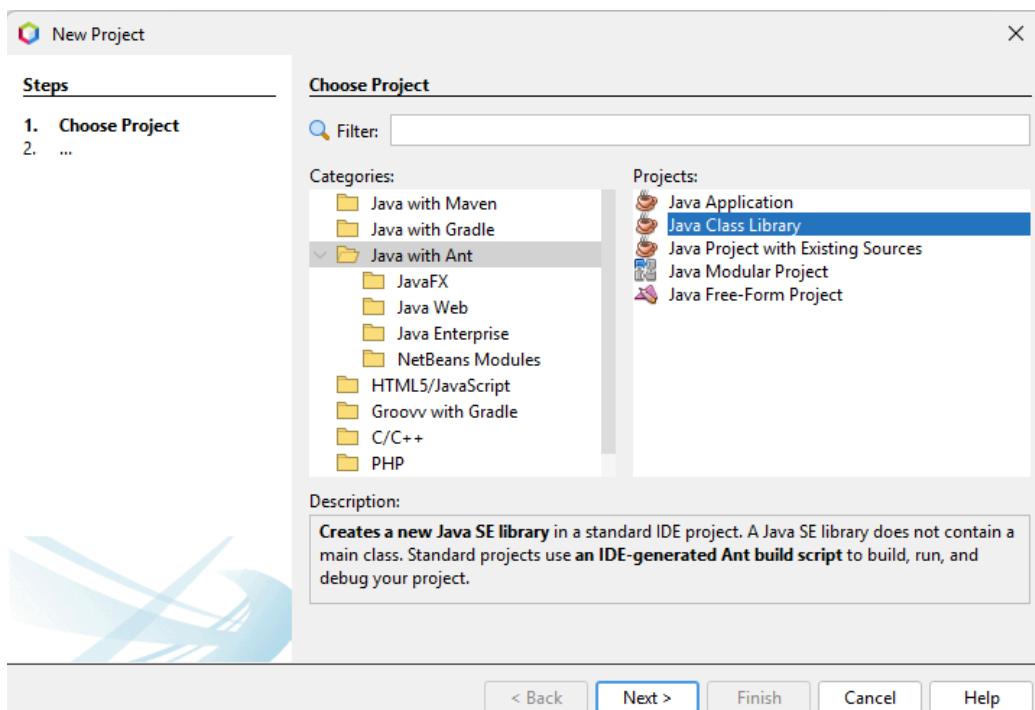
proyecto RMI Estándar - Calculadora IMC

Eliab Ricardo Jaraba Rios

Crear una librería

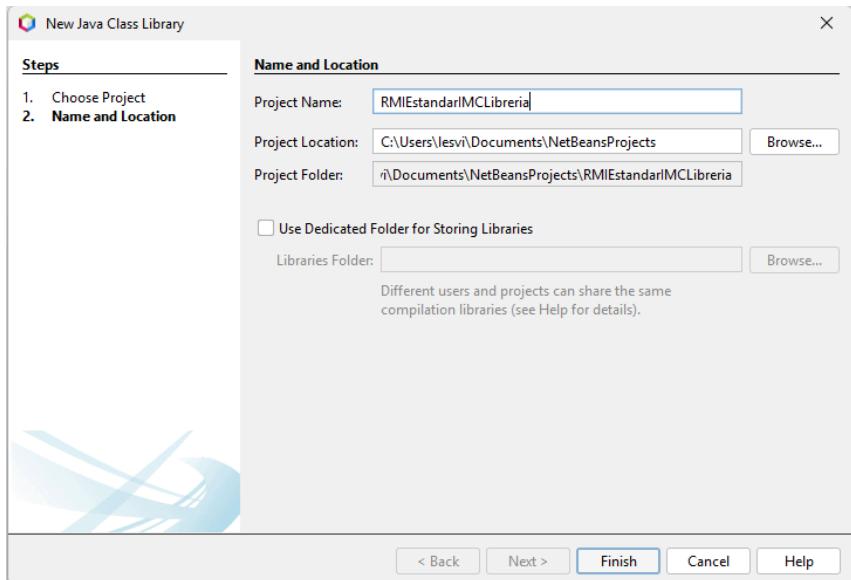


Lo primero que creamos es una librería; damos clic a menú y después clic en new project

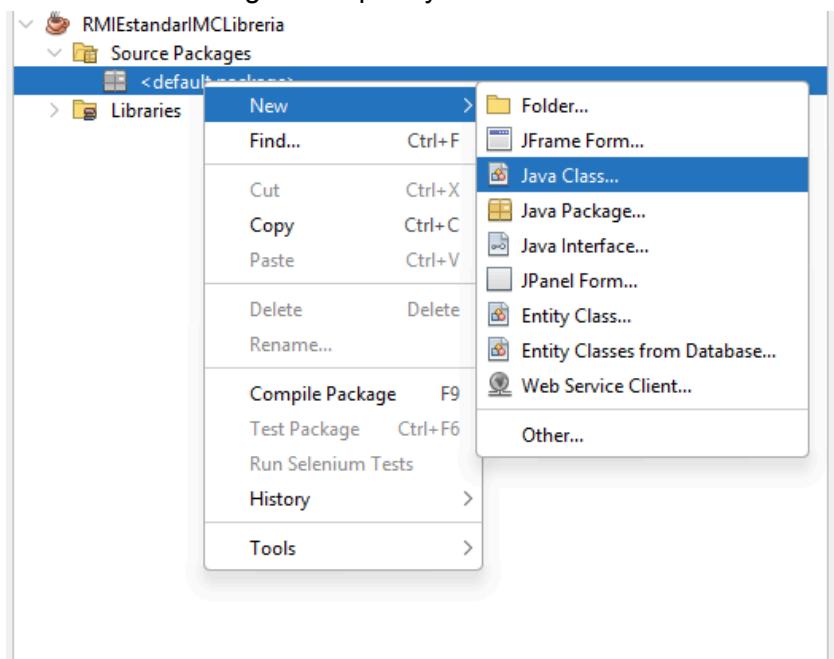


Va a salir una ventana, lo primero es:

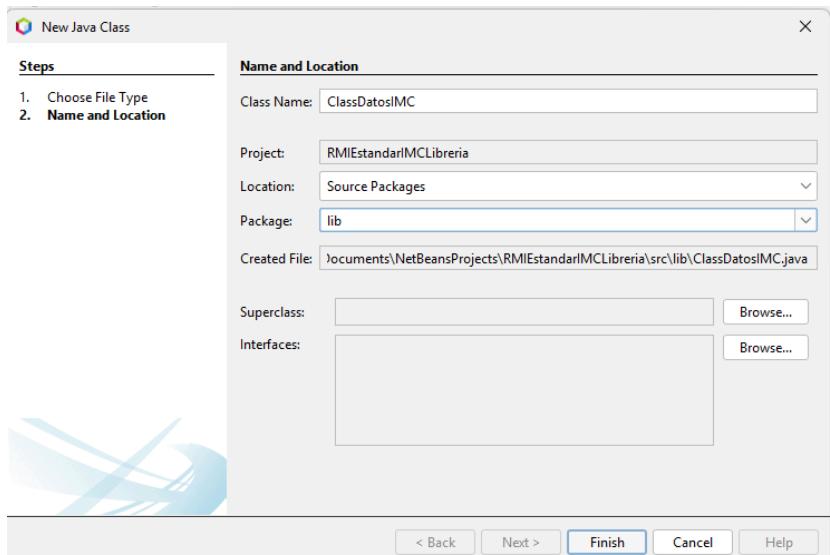
1. Dar click en el botón nuevo proyecto.
2. Escoger la categoría java with ant.
3. y después elegir Java Class library.
4. por último dar click en el botón next.



En esta otra ventana se puede ver que le ponemos nombre al proyecto, le colocamos en alguna carpeta y terminamos con el botón Finish.

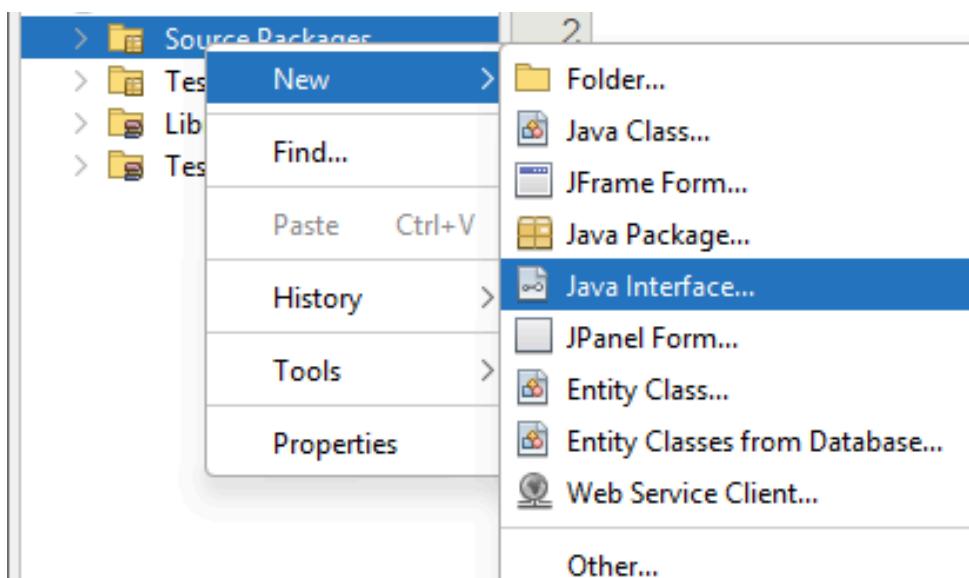


en el paquete de default le damos click derecho, saldrán diferentes opciones de un menú y damos click en new y después click en java class

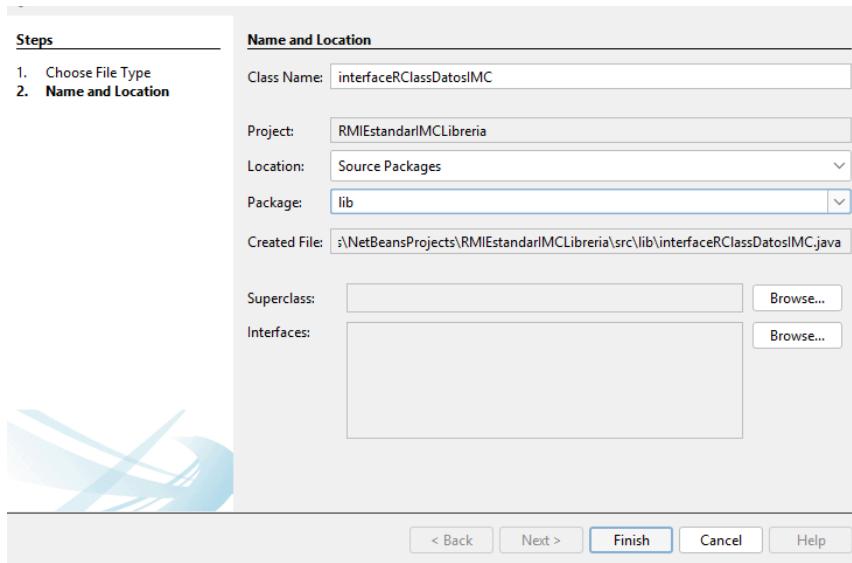


En esta ventana le ponemos nombre al proyecto esta es la parte de la clase ClassDatosIMC

1. En la casilla de project Names; colocamos el nombre de la clase ClassDatosIMC
2. En la casilla de package colocamos el nombre del paquete nuevo que se va crear.
3. finalizar todo dando click en el botón finish.

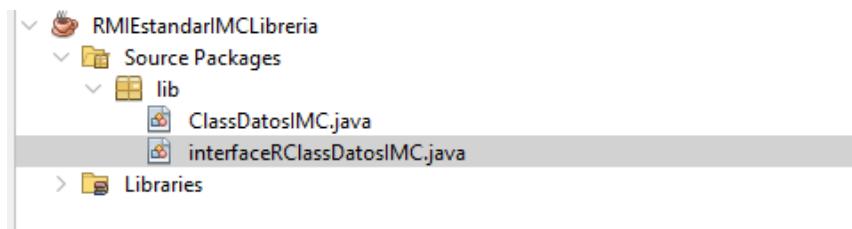


en la carpeta de Source le damos click derecho, saldrán diferentes opciones de un menú y damos click en new y después click en java interface



En esta ventana le ponemos nombre al proyecto esta es la parte de la clase interfaceRClassDatosIMC

1. En la casilla de project Names; colocamos el nombre de la clase interfaceRClassDatosIMC
2. En la casilla de package colocamos el nombre del paquete ya antes creado.
3. finalizar todo dando click en el botón finish.



Aquí verificamos que la clase y la interfaz se crearon bien.

Archivo ClassDatosIMC

```
public class ClassDatosIMC {  
    private float peso;  
    private float altura;  
    private float resultado;  
    private String interpretacion;  
  
    public ClassDatosIMC() {  
    }  
  
    public ClassDatosIMC(float peso, float altura) {  
        this.peso = peso;  
        this.altura = altura;  
    }  
  
    public float getPeso() {  
        return peso;  
    }  
  
    public void setPeso(float peso) {  
        this.peso = peso;  
    }  
  
    public float getAltura() {  
        return altura;  
    }  
  
    public void setAltura(float altura) {  
        this.altura = altura;  
    }  
  
    public float getResultado() {  
        return resultado;  
    }  
  
    public void setResultado(float resultado) {  
        this.resultado = resultado;  
    }  
}
```

En primera instancia se importa la clase serializable y se agrega al lado del nombre de la clase como implements serializable.

La clase tiene 4 atributos: peso, altura, resultado e interpretación, además, la clase tiene 2 constructores; uno vacío y, el otro solo acepta peso y altura.

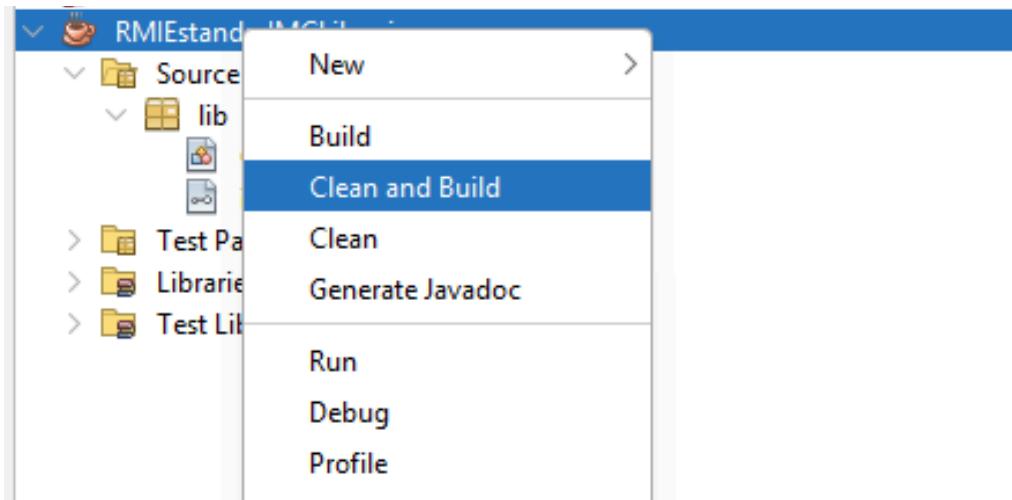
La clase tiene varios métodos accesores y mutadores de los atributos de la clase.

archivo interfaceRClassDatosIMC

```
package lib;
import java.rmi.Remote;
import java.rmi.RemoteException;

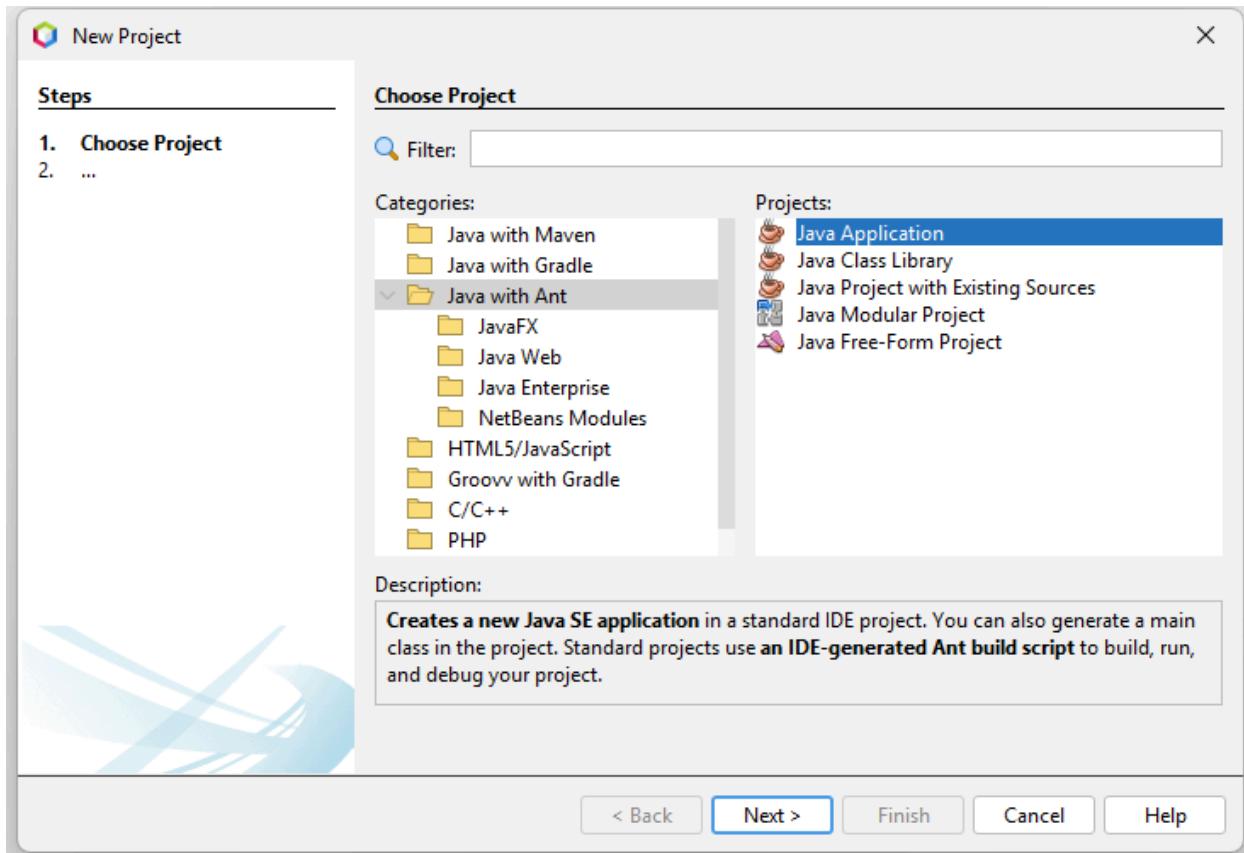
public interface interfaceRClassDatosIMC extends Remote{
    public ClassDatosIMC CalcularIMC (ClassDatosIMC datosimc) throws RemoteException ;
}
```

En la interfaz se crea un método clacularImc, además se tiene que ingresar un parámetro, que es tipo de la clase antes creada(DatosImc).



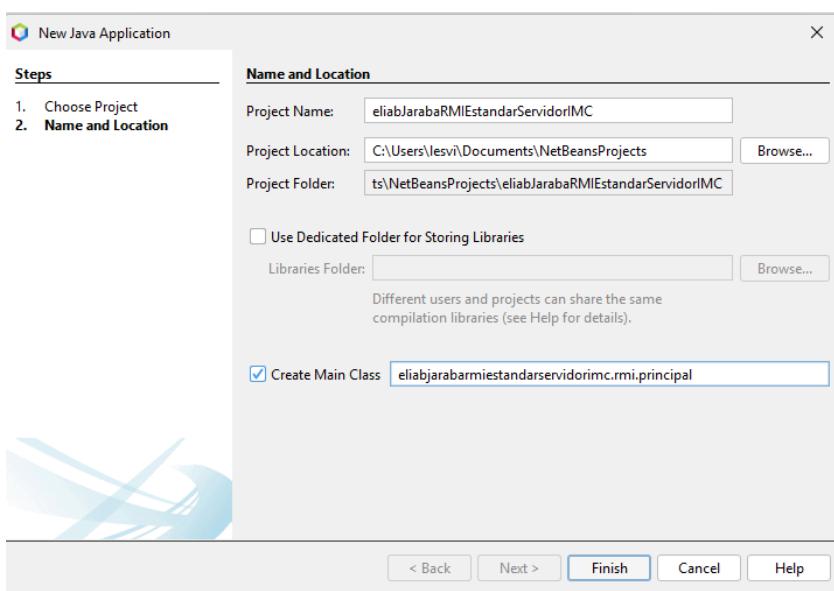
Ahora creamos un archivo.jar. De la librería ya creada, clic derecho en el nombre del proyecto y clic en clean and build.

SERVIDOR RMI ESTÁNDAR



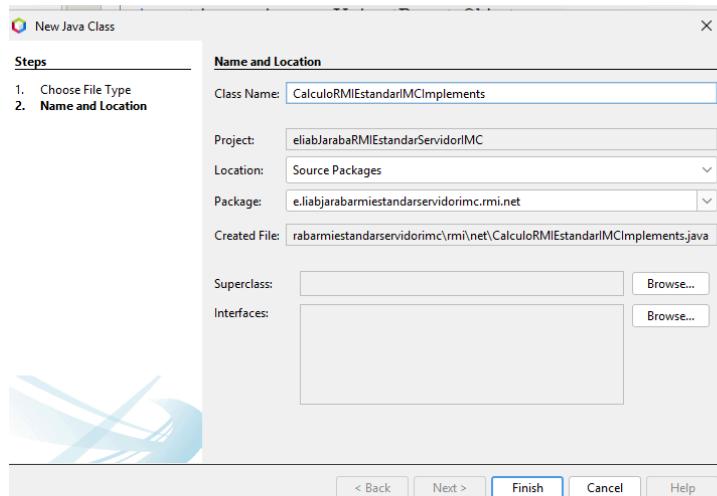
Lo primero es crear el archivo:

1. Dar click en el botón nuevo proyecto.
2. Escoger la categoría java with ant.
3. y después elegir Java Application.
4. por último dar click en el botón next.



En esta ventana le ponemos nombre al proyecto esta es la parte del servidor:

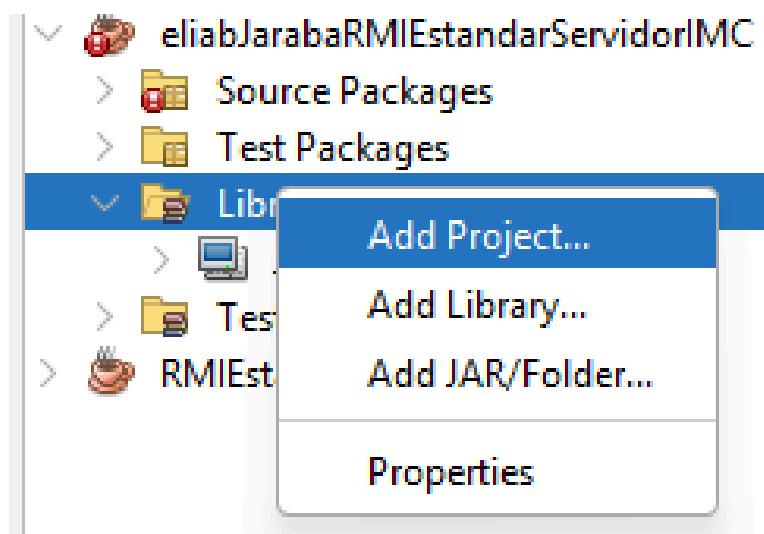
1. En la casilla de project Names; colocamos el nombre del proyecto
2. y 3. damos click en el botón Browse y encontramos la carpeta donde queremos guardar el proyecto.
4. dar click en la casilla cuadrada y
5. al lado se coloca el nombre de la main class.
6. finalizamos todo dando click en el botón finish



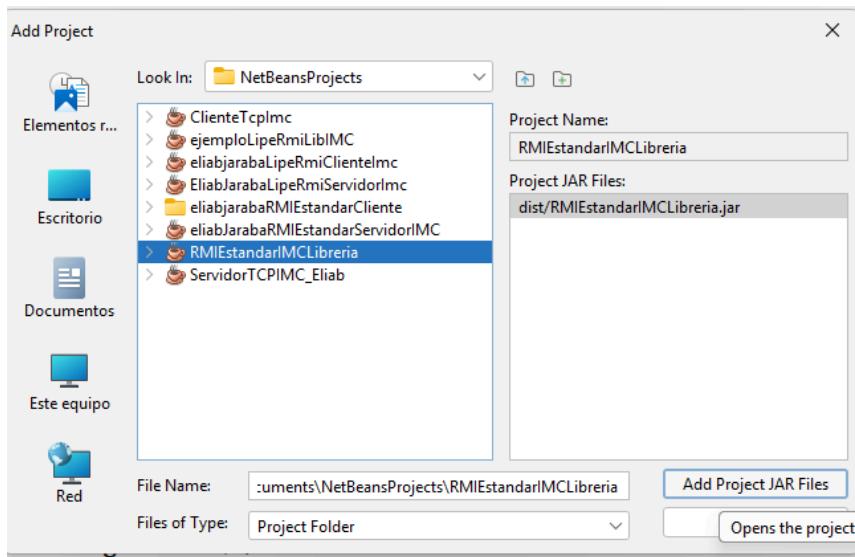
damos click derecho en un paquete, saldrá un menú con diferente opciones y damos click en new y después click en java class.

En esta ventana le ponemos nombre al proyecto esta es la parte del subProcesoCliente

1. En la casilla de project Names; colocamos el nombre del proyecto subProcesoCliente
2. En la casilla de package colocamos el nombre del paquete ya creado.
3. finalizar todo dando click en el botón finish.



hay que agregar el .jar ya creado con anterioridad, y en el proyecto servidor en la carpeta de librerías; clic derecho y add project.



En esta ventana que aparece; se busca la librería y la agregamos al proyecto servidor.

ARCHIVO CalculoRMIEstandarIMCImplements

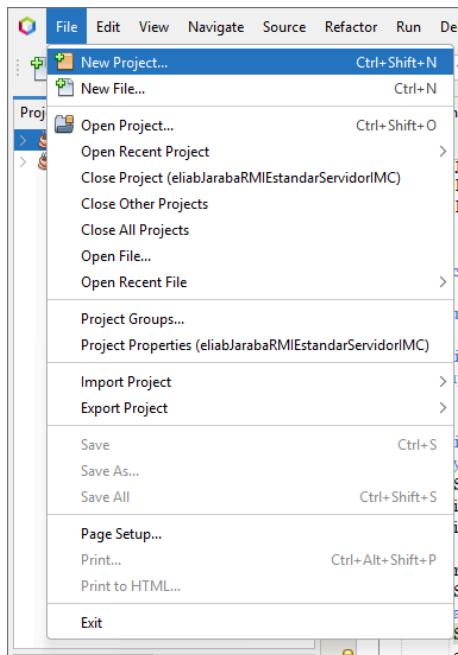
```
1 package eliababariablestandarservidorimc.rmi.net;
2 import java.rmi.AlreadyBoundException;
3 import java.rmi.registry.Registry;
4 import java.rmi.registry.LocateRegistry;
5 import java.rmi.server.UnicastRemoteObject;
6 import java.rmi.RemoteException;
7 import lib.ClassDatosIMC;
8 import lib.interfaceRClassDatosIMC;
9
10
11 public class CalculoRMIEstandarIMCImplements extends UnicastRemoteObject implements interfaceRClassDatosIMC{
12
13     public ClassDatosIMC datosimc;
14
15     public CalculoRMIEstandarIMCImplements() throws RemoteException{
16         super();
17     }
18
19     public static void main( String []args) throws AlreadyBoundException{
20         try {
21             String name = "interfaceRClassDatosIMC";
22             interfaceRClassDatosIMC OServicio =new CalculoRMIEstandarIMCImplements();
23             interfaceRClassDatosIMC stub = (interfaceRClassDatosIMC) UnicastRemoteObject.exportObject(OServicio,0);
24             Registry registry = LocateRegistry.getRegistry(1099);
25             registry.rebind(name, stub);
26             System.out.println("El servidor esta funcionando bien");
27         } catch (Exception ex) {
28             System.out.println("Error de url mal formada: " + ex.getMessage());
29             ex.printStackTrace();
30             System.out.println("El servidor no funciona");
31         }
32     }
33
34     @Override
35     public ClassDatosIMC CalcularIMC (ClassDatosIMC datosimc) throws RemoteException{
36         float resultado = 0;
37
38         if(datosimc.getPeso() <= 0 || datosimc.getAltura() <= 0){
39             datosimc.setInterpretacion("ERROR: El peso y la altura deben ser mayor que 0");
40             return datosimc;
41         }else {
42             resultado = datosimc.getPeso() / (datosimc.getAltura() * datosimc.getAltura());
43             datosimc.setResultado(resultado);
44             if (resultado < 18.5) {
45                 datosimc.setInterpretacion(" Debes consultar de un medico, tu peso es muy bajo ");
46             } else if (resultado >= 18.5 && resultado <= 24.9) {
47                 datosimc.setInterpretacion(" Estan bien de peso ");
48             } else if (resultado >= 24.9 && resultado <= 29.9) {
49                 datosimc.setInterpretacion(" Debes bajar un poco de peso ");
50             } else {
51                 datosimc.setInterpretacion(" Debes consultar de un medico, tu peso es muy alto ");
52             }
53             return datosimc;
54         }
55     }
56
57     }
58
59     }
60
61     }
62 }
```

Se importan las clases necesarias para trabajar como es el caso de

eliabjarabarmiestandarservidorIMC.rmi.net,
 java.rmi.registry.Registry,
 java.rmi.server.UnicastRemoteObject, java.rmi.RemoteException, lib.ClassDatosIMC,
 lib.interfaceRClassDatosIMC. Estos definen los datos y la interfaz remota para el cálculo. Clase Principal: La clase CalculoRMIEstandarIMCImplements extiende UnicastRemoteObject (lo que la hace compatible con RMI) e implementa la interfaz interfaceRClassDatosIMC (que define los métodos remotos, como el cálculo del IMC). El constructor básico CalculoRMIEstandarIMCImplements() hace la invocación al constructor de la clase padre. Es necesario para inicializar la parte remota. Método Principal (main): Se crea una instancia del servicio remoto CalculoRMIEstandarIMCImplements. Este servicio se registra en el puerto 1099 para que otros programas puedan conectarse. Como se muestra un mensaje indicando que el servidor está funcionando. Si hay un error, se notifica y se imprime la excepción. Método Remoto (CalcularIMC): Este método recibe un objeto ClassDatosIMC que contiene los datos necesarios para el cálculo (peso y altura). Primero, creamos un atributo llamado datos de tipo DatosIMC. Le agregamos el constructor a la clase, debajo agregamos un método llamado calcularIMC, el cual recibe un parámetro de tipo DatosIMC.

Dentro del método se crea una variable float llamada resultado, primero hay un if con un else; el if comprueba si los datos enviados son iguales a cero si es así manda un error, el else tiene la fórmula IMC y da un resultado, ese resultado es comparado para ver en qué categoría cabe si flaco, normal y gordo, al final devuelve un resultado de tipo DatosIMC. El servidor envía mensajes para informar al cliente sobre el estado del cálculo (por ejemplo, errores o resultados).

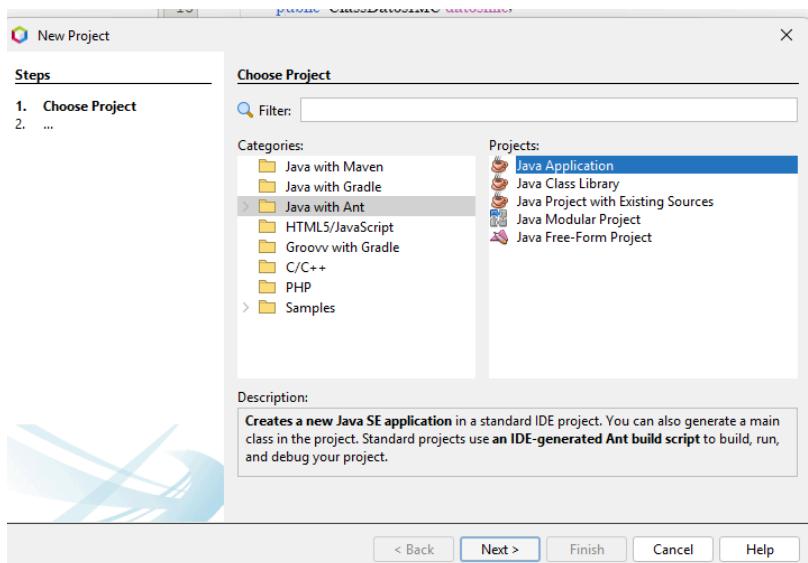
CLIENTE RMI ESTÁNDAR



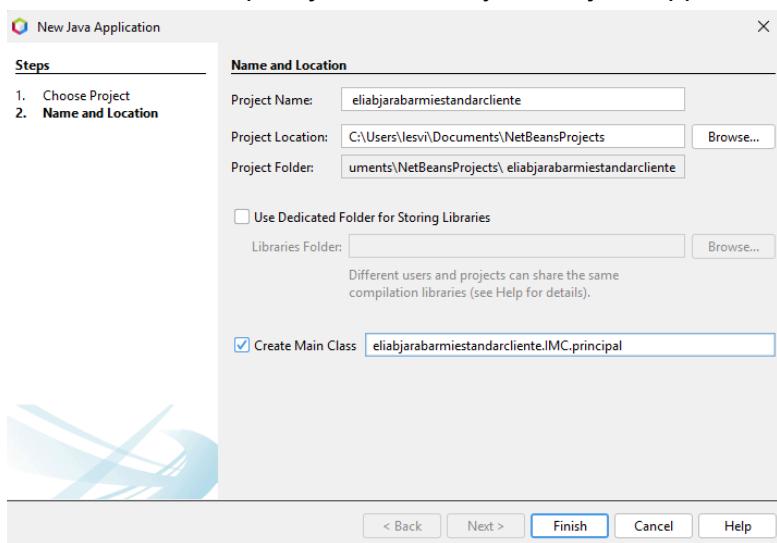
Lo primero es crear el nuevo archivo para el cliente :

1. Dar click en el botón nuevo proyecto.
2. Escoger la categoría java with ant.

3. y después elegir Java Application.
4. por último dar click en el botón next.

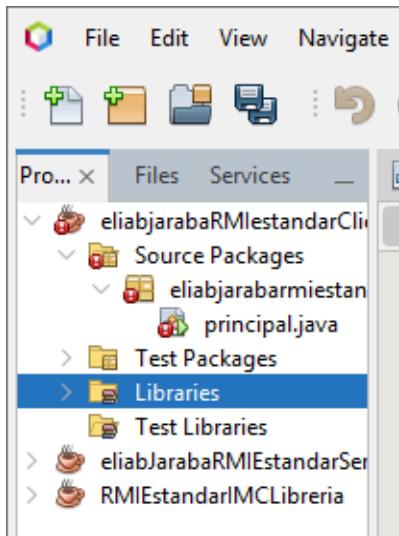


damos clic a la carpeta java with ant y clc en java application, seguimos con el botón next.

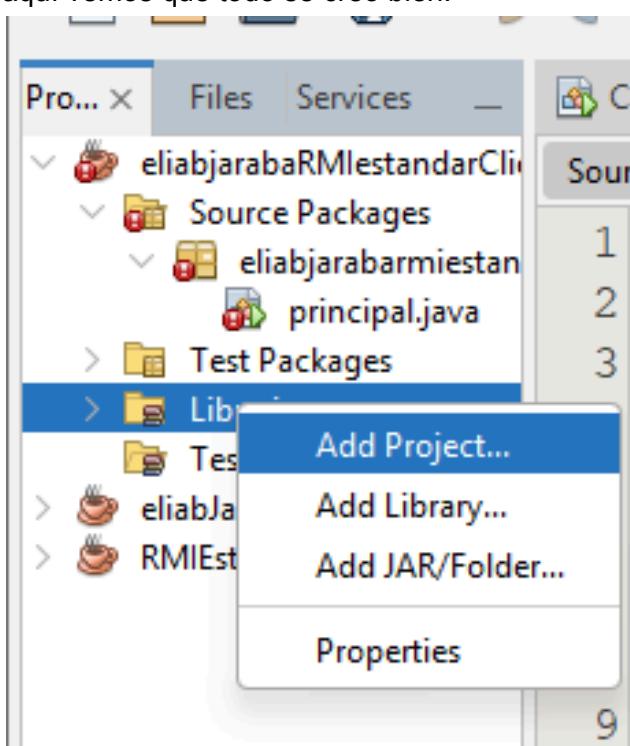


En esta ventana le ponemos nombre al proyecto esta es la parte del cliente:

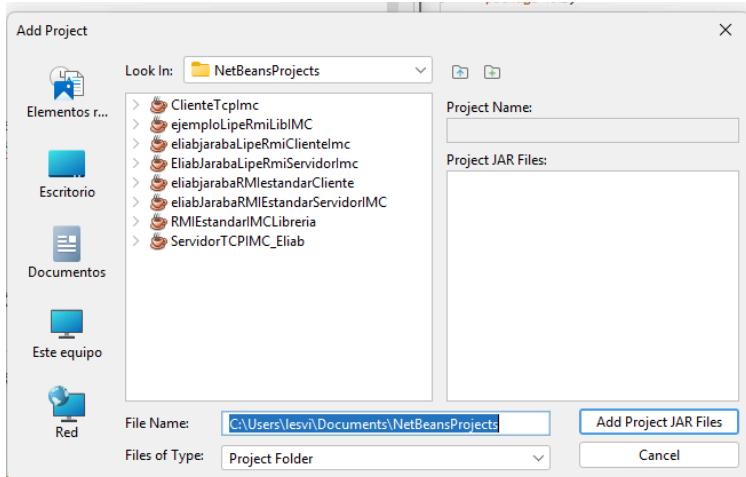
1. En la casilla de project Names; colocamos el nombre del proyecto
2. y 3. damos click en el botón Browse y encontramos la carpeta donde queremos guardar el proyecto.
4. dar click en la casilla cuadrada y
5. al lado se coloca el nombre de la main class cliente.
6. finalizamos todo dando click en el botón finish



aquí vemos que todo se creó bien.



Aquí agregamos el archivo .jar. Aquí en el proyecto.



buscamos la librería y damos clic en ella y después clic en el botón add project jar files.

Archivo principal.

se colocan en las librerías `java.awt.BorderLayout`, `java.rmi.registry.LocateRegistry`, `java.rmi.registry.Registry`, `java.util.Scanner`, `lib.interfaceRClassDatosIMC`, `lib.ClassDatosIMC`. De forma sencilla la librería `java.rmi.registry.LocateRegistry` y `java.rmi.registry.Registry`: Usadas para conectar con un registro RMI que permite localizar objetos remotos. `java.util.Scanner`: Permite la entrada de datos por parte del usuario. `lib.interfaceRClassDatosIMC` y

lib.ClassDatosIMC: Representan clases definidas en otro lugar que probablemente sean el contrato (interfaz) y la implementación para manejar datos del IMC. La clase principal contiene el método main, que ejecuta la aplicación. Se crean dos instancias de ClassDatosIMC: Una (datos) para almacenar los datos que ingresa el usuario. Otra (res) para almacenar el resultado returnedo desde el servicio remoto. Después se crea una variable de tipo scanner que solicita al usuario ingresar su peso y altura. Estos valores se guardan en la instancia datos. Usa el registro RMI para buscar el objeto remoto registrado como "interfaceRClassDatosIMC". Esto devuelve un "stub" (representación del objeto remoto) que permite al cliente invocar métodos en el servidor. Más abajo el método remoto CalcularIMC pasándole la instancia datos. Este método probablemente calcula el IMC en el servidor y retorna un objeto de tipo ClassDatosIMC con los resultados. Después se muestra el resultado numérico del IMC y un mensaje interpretativo (por ejemplo, "Bajo peso", "Peso normal", "Sobrepeso"). En caso de que algo falle (como problemas de red o conexión al servidor), captura la excepción y muestra detalles del error.

Comparación de RMI LipeRMI y RMI estándar de java.

Similitudes

- Propósito principal: Tanto el RMI estándar como Lipermi permiten la comunicación entre aplicaciones distribuidas mediante la invocación de métodos remotos, conectando un cliente y un servidor. En ambos casos, el cliente envía una solicitud al servidor, este procesa la información y devuelve una respuesta, lo que facilita la implementación de arquitectura cliente-servidor.
- Modelo de trabajo: Ambas implementaciones siguen el paradigma RPC (Remote Procedure Call), en el que se invocan métodos en el servidor de forma casi transparente para el cliente. Necesitan que tanto el cliente como el servidor comparten una interfaz remota, que define los métodos que se podrán invocar de forma remota. Interacción
- Cliente-Servidor: En ambos casos, se trabaja con stub (representación en el cliente del objeto remoto) y, de alguna manera, un "skeleton" (en el lado del servidor), aunque estos se manejan de manera diferente en cada tecnología.
- Soporte para objetos remotos: Tanto en RMI estándar como en Lipermi, los métodos pueden recibir y retornar objetos complejos siempre que sean serializables, lo que permite el intercambio de datos más robusto.

Diferencias

- Configuración. RMI Estándar: Requiere un proceso más complejo de configuración, incluyendo el registro de objetos en el registro RMI (como LocateRegistry) y la exportación manual de objetos remotos. Además, puede ser necesario usar comandos adicionales como rmic para generar clases stub en versiones más antiguas de Java. Lipermi: Es mucho más simple y elimina la necesidad de un registro RMI. Los objetos remotos se exportan automáticamente sin necesidad de generar stubs manualmente ni trabajar con el registro.
- Librerías externas. RMI Estándar: Es una tecnología nativa de Java, lo que significa que no necesitas incluir bibliotecas externas. Lipermi: Es una biblioteca externa que debes agregar al proyecto como dependencia, aunque simplifica algunas tareas.

- Desempeño. RMI Estándar: Tiene más sobrecarga debido a la necesidad de interactuar con el registro RMI y otros procesos internos. Lipermi: Es más ligera y está optimizada, por lo que puede ser más rápida en proyectos pequeños o medianos.
- Flexibilidad. RMI Estándar: tiende a ser más robusta y adecuada para aplicaciones más complejas que requieran una arquitectura distribuida con mayores niveles de personalización. Lipermi: Es más flexible y sencilla, lo que la hace ideal para prototipos rápidos o sistemas más ligeros.
- Arquitectura del registro. RMI Estándar: Utiliza un registro centralizado (Registry) que permite localizar los servicios remotos mediante un nombre específico. Lipermi: No depende de un registro central. En su lugar, el cliente se conecta directamente al servidor especificando una dirección IP y un puerto.
- Compatibilidad con Java avanzado. RMI Estándar: Está profundamente integrado con el ecosistema de Java y puede ser usado con características avanzadas como seguridad basada en políticas (security.policy). Lipermi: Aunque funcional, es menos formal y no incluye características de seguridad tan avanzadas como las de RMI estándar.