# Java.RMI

Java ofrece por defecto la librería RMI para invocar métodos remotamente. Es una forma sencilla y útil de comunicar servidor y cliente en aplicaciones desarrolladas exclusivamente bajo este lenguaje. Su gran ventaja es el envío de objetos en la comunicación. Lamentablemente, en Android este mecanismo está capado y su implementación no es posible. Aunque existen alternativas para hacer uso de esta tecnología.

Para ello contamos con Lipe RMI.

# Lipe RMI

Lipe RMI es una librería (*externa*) que reemplaza el engranaje RMI nativo en Java. La intención del proyecto (*libre y de código abierto*) es mejorar la implementación actual prosperando la arquitectura de las comunicaciones en el uso de internet y optimizando el ancho de banda.

Gracias a esta propuesta podemos desarrollar nuestra conexión RMI para los dispositivos Android.

# Implementar Lipe RMI

A continuación, veremos cómo empezar a trabajar con esta API.

## Importar la librería

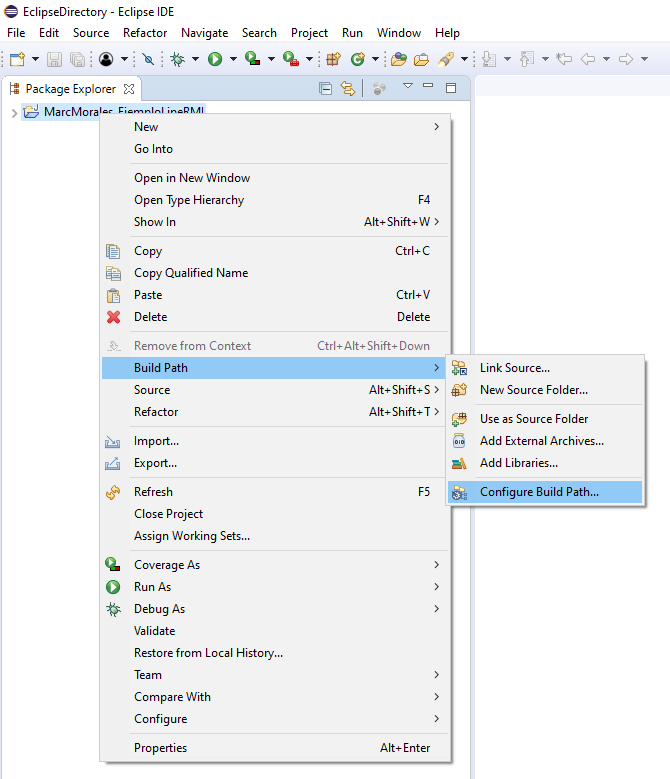
Lo primero es lo primero, descargar e importar la librería.

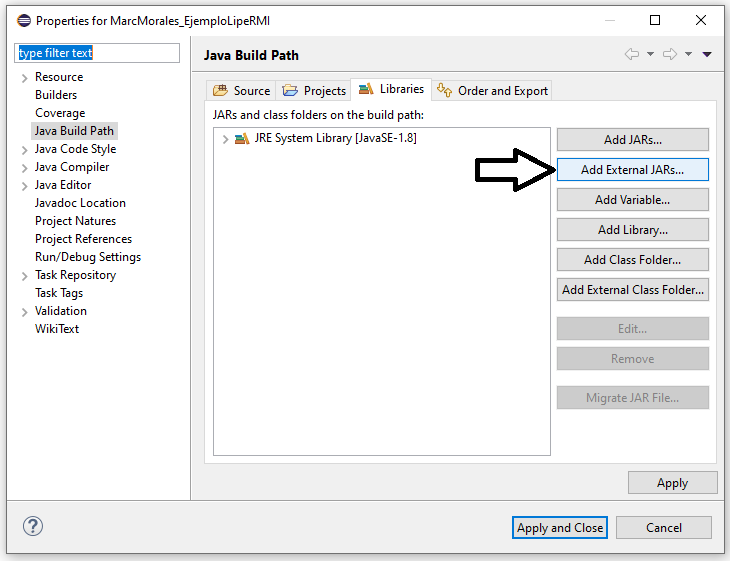
**Descárgala aquí:** <http://lipermi.sourceforge.net/download.php>

### Importar en Eclipse

Para importar la librería en Eclipse debemos seguir estas instrucciones:

*Click derecho sobre nuestro proyecto > Build Path > Configure Build Path > Add External Jars > Seleccionar el .jar de Lipe RMI*

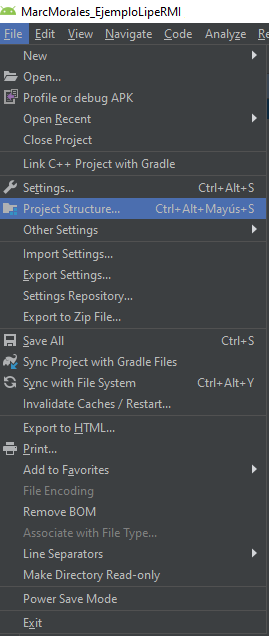


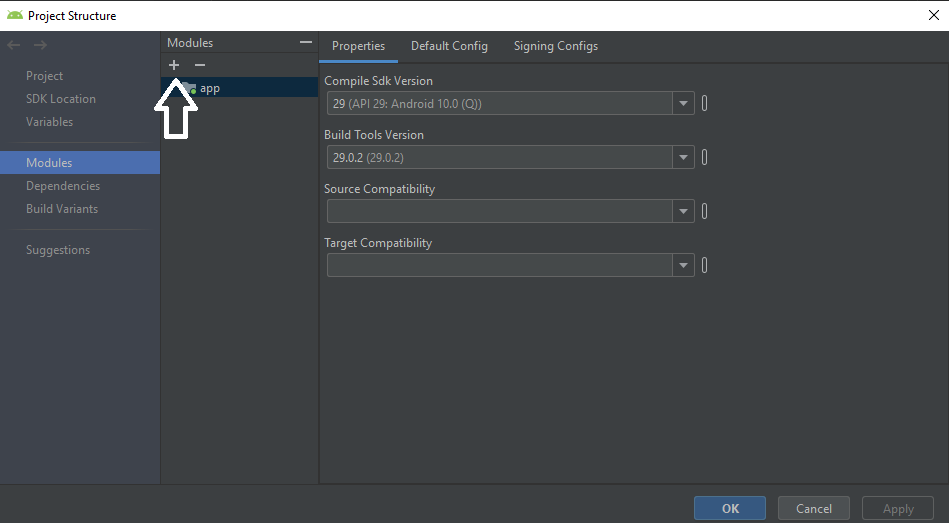


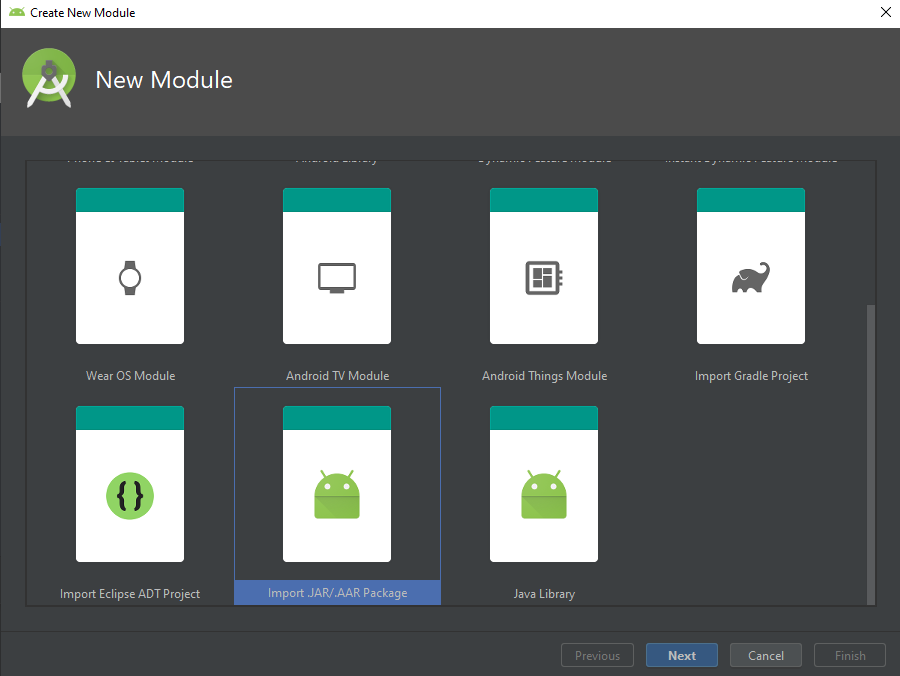
### Importar en Android Studio

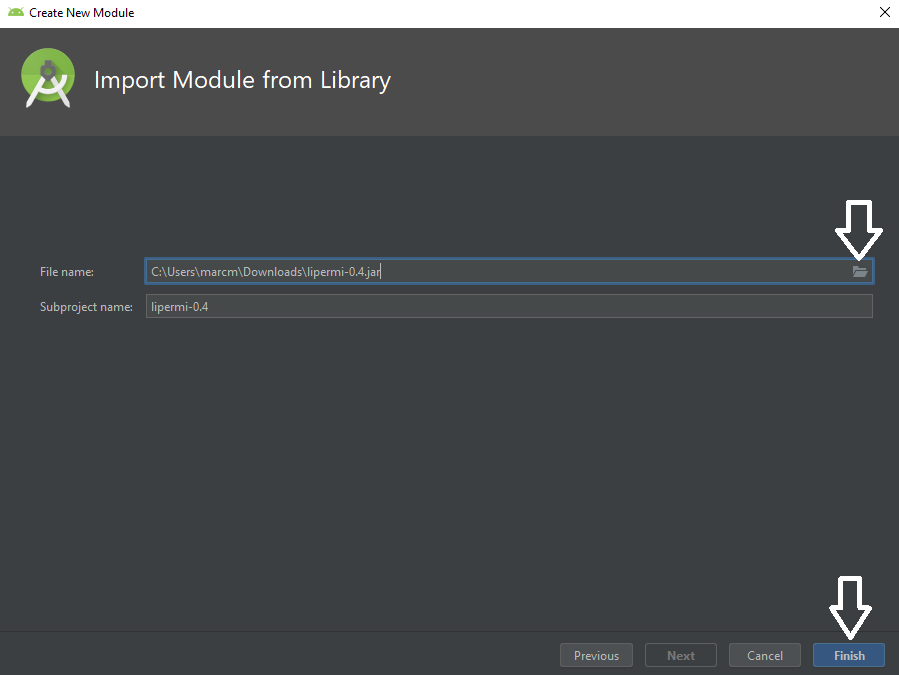
Para importar la librería en Android Studio debemos seguir estas instrucciones:

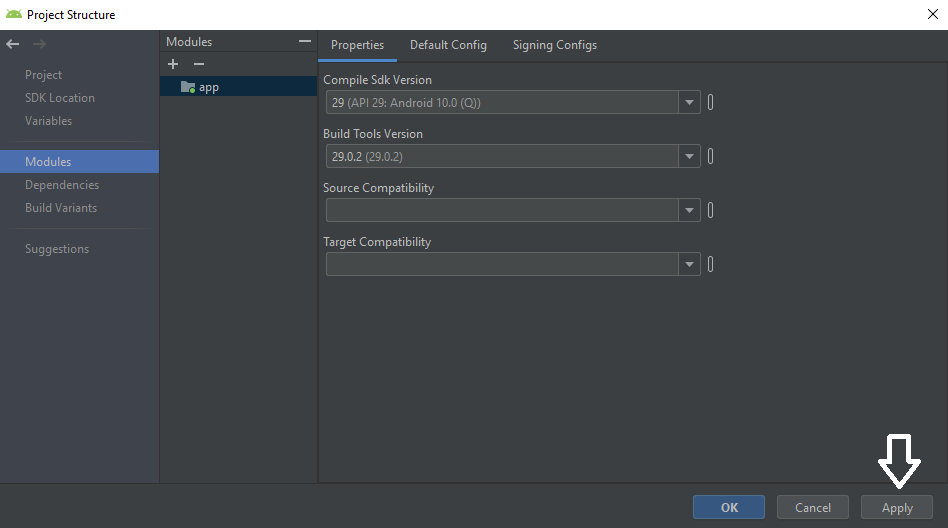
*File > Project Structure > Modules > + > Import .JAR / .AAR Package > Seleccionar el .jar de Lipe RMI > Finish > Apply > Dependencies > Seleccionar app > + > Module Dependency > Marcar la casilla de Lipe RMI (teniendo seleccionado implementation en el Step 2 > Ok > Apply*

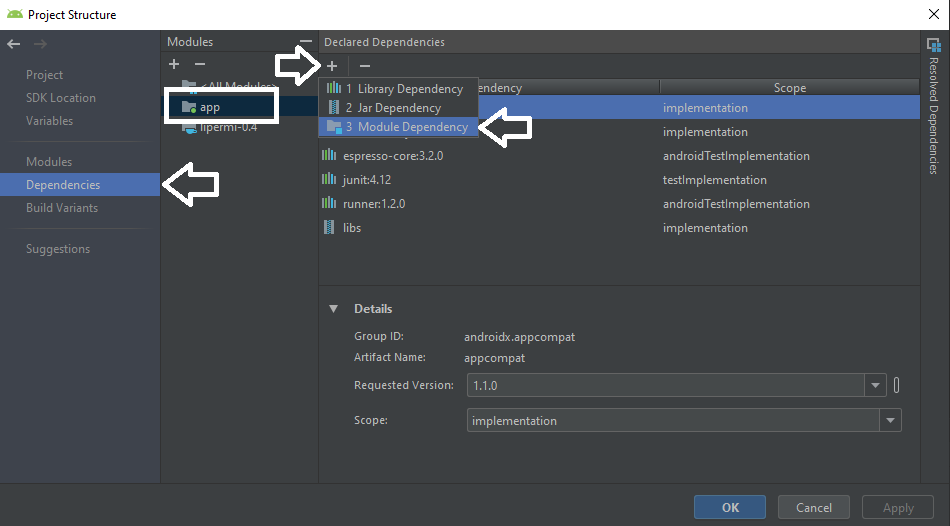


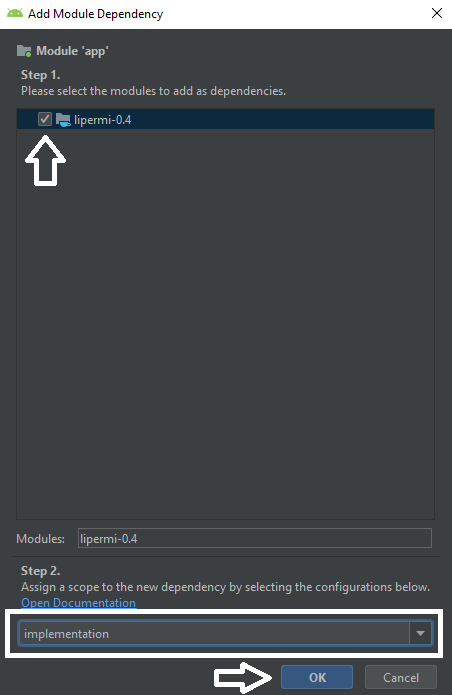


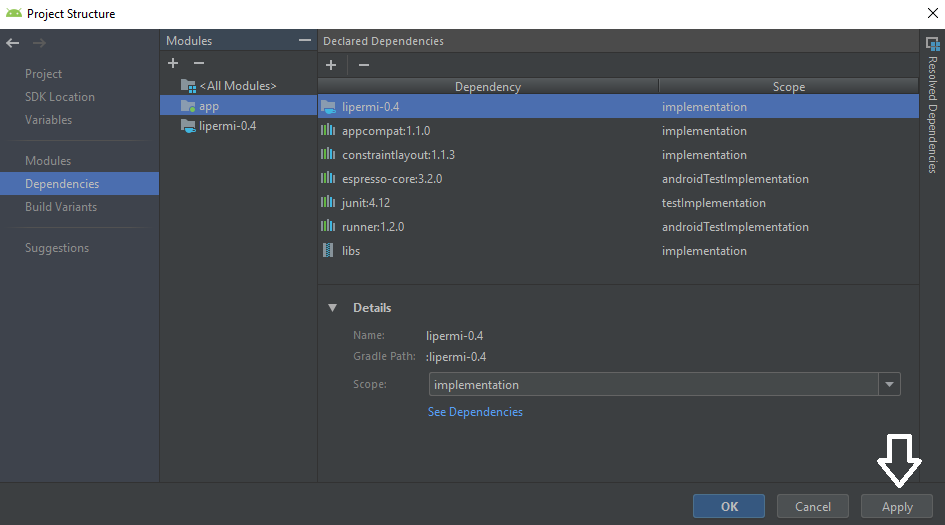












## Establecer la interface

Con la librería importada ya podemos empezar a programar la comunicación.

Igual que con el RMI nativo de Java necesitamos construir una interface con los métodos que pueden ejecutarse.

Aquí tienes un ejemplo:

1. **public** **interface** interfazRMI {
2. **public** ArrayList<String> ejemploMetodo();
3. }

**Comentarios:**

En este código (*con el que trabajaremos para la ejemplificación*) se contempla un método llamado “*ejemploMetodo*” en el que el cliente solicita un ArrayList de String al servidor.

Recuerda que esta Interface debe estar tanto en el cliente como en el servidor.

## Programar el servidor

Ya podemos entrar en la materia del asunto y empezar a desarrollar nuestro servidor. Para ello vamos a instanciar diferentes variables de la librería Lipe RMI.

Aquí tienes un ejemplo:

1. **public** **class** ejemploServer **implements** interfazRMI {
2. **public** ejemploServer() {
3. **try** {
4. CallHandler callHandler = **new** CallHandler();
5. callHandler.registerGlobal(interfazRMI.**class**, **this**);
6. Server server = **new** Server();
7. server.bind(7777, callHandler);
8. server.addServerListener(**new** IServerListener() {
10. @Override
11. **public** **void** clientDisconnected([Socket](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+socket) socket) {
12. [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Cliente desconectado: " + socket.getInetAddress());
13. }
15. @Override
16. **public** **void** clientConnected([Socket](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+socket) socket) {
17. [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Cliente conectado: " + socket.getInetAddress());
18. }
19. });
20. [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Servidor iniciado.");
21. } **catch** ([Exception](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+exception) e) {
22. e.printStackTrace();
23. }
24. }
26. @Override
27. **public** ArrayList<String> ejemploMetodo() {
28. [System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Se ha solicitado un ArrayList de String.");
29. ArrayList<String> array = **new** ArrayList<String>();
30. array.add("Texto de ejemplo.");
31. **return** array;
32. }
33. }

**Comentarios:**

En este código instanciamos un objeto Call Handler (*responsable de conocer los objetos locales exportados y las instancias remotas*). Con él llamamos a la función “*registerGlobal*” donde incluimos la interfaz y la clase actual como parámetros.

Posteriormente, declaramos un objeto Server. Con este podemos llamar a la función “*bind*” donde incluimos el puerto a través del que se realizará la conexión y la variable Call Handler como parámetros.

Sobre este objeto incluimos un “*IServerListener*”, este “*listener*” implementa dos métodos “*clientDisconnected*” y “*clientConnected*”. Este último es llamado cuando el cliente se conecta mientras que se llama al primero en el caso contrario.

Y, finalmente, también incluimos el método contemplado en la Interface. Recuerda tenerla implementada en la clase.

## Programar el cliente

Con el servidor en marcha ya sólo falta desarrollar el cliente. Para ello vamos a instanciar de nuevo diferentes variables de la librería Lipe RMI, algunas ya las hemos visto y otras son nuevas.

Aquí tienes un ejemplo:

1. **public** **class** MainActivity **extends** AppCompatActivity {
3. @Override
4. **protected** **void** onCreate(Bundle savedInstanceState) {
5. **super**.onCreate(savedInstanceState);
6. setContentView(R.layout.activity\_main);
7. **new** ejemploCliente().execute();
8. }
10. **class** ejemploCliente **extends** AsyncTask<[Void](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+void), [Void](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+void), MainActivity> {
11. @Override
12. **protected** MainActivity doInBackground([Void](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+void)... params) {
13. **try** {
14. CallHandler callHandler = **new** CallHandler();
15. Client client = **new** Client("ipServidor", 7777, callHandler);
16. interfazRMI inter = (interfazRMI) client.getGlobal(interfazRMI.**class**);
17. ArrayList<String> array = inter.ejemploMetodo();
18. client.close();
19. } **catch** ([IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception) e) {
21. }
22. **return** **null**;
23. }
24. }
25. }

**Comentarios:**

En este código instanciamos un objeto Call Handler (*cómo ya lo habíamos hecho anteriormente*).

Posteriormente, declaramos un objeto Client. Al instanciarlo incluimos como parámetros la IP del servidor, el puerto a través del que se realiza la conexión (*asegúrate de utilizar el mismo que empleaste en el servidor*) y el objeto Call Handler.

Acto seguido, vamos a instanciar la Interface parseando una llamada con el objeto Client a la función “*getGlobal*” donde incluimos como parámetro la propia interface. Con esta instanciada ya podemos llamar a los métodos que están contemplados en la misma.

Recuerda cerrar el objeto Client una vez hayas terminado de utilizarlo.

## TIPS

Para terminar aquí tienes algunos consejos que pueden serte útiles:

* Asegúrate de declarar el permiso de Internet en el AndroidManifest.xml de tu proyecto Android. Puedes hacerlo con esta línea:

<uses-permission android:name="android.permission.INTERNET" />

* Tener el Firewall activado puede ocasionar problemas.
* Utiliza el mismo nombre de package tanto en cliente como en servidor. Recuerda que el nombre de un package en Android es algo parecido a: *com.lipermi.nombreapp.conexion*.
* A veces las sugerencias “*Import*” de los IDE no funcionan correctamente, esto es lo que deberías importar:

En el servidor:

1. import java.net.Socket;
2. import lipermi.handler.CallHandler;
3. import lipermi.net.Server;
4. import lipermi.net.IServerListener;

En el cliente:

1. import android.os.AsyncTask;
2. import java.io.IOException;
3. import lipermi.handler.CallHandler;
4. import lipermi.net.Client;

* Una vez termines la implementación desinstala tu aplicación del emulador de Android (o de tu dispositivo) antes de lanzarla de nuevo. Si no haces esto puede que tu conexión no funcione.

Finalmente, si quieres aprender más sobre Lipe RMI recuerda que tienes más información al respecto en su página web.

**Lipe RMI:** <http://lipermi.sourceforge.net/>