

# EXPOSICION 3

## Equipo 4

ESCOM

### Implementacion de Skeleton

Equipo:

- Ramirez Contreras Angel Humberto
- Reyes Vivar Fernando
- Morales Torres Alejandro



# CONTENIDO

**Introduccion**

**¿Qué es un SKELETON?**

**Evolución de  
SKELETONS en JAVA**

**Implementación de  
SKELETON**

**Implementación de  
servidor**

**Implementación del  
cliente**

**Conclusiones**



# INTRODUCCION

---

HOY VAMOS A EXPLORAR UN TEMA CRUCIAL EN EL ÁMBITO DEL DESARROLLO DE APLICACIONES DISTRIBUIDAS: **LA IMPLEMENTACIÓN DE SKELETON EN JAVA RMI** (REMOTE METHOD INVOCATION). JAVA RMI ES UNA PODEROSA API QUE FACILITA **LA COMUNICACIÓN Y LA INTEROPERABILIDAD EN ENTORNOS DISTRIBUIDOS.**



# RECORDAMOS...

**JAVA RMI** ES UN MODELO DE OBJETOS DISTRIBUIDO QUE NOS PERMITE EJECUTAR OBJETOS DE UNA JVM Y QUE INVOQUEN A SU VEZ OBJETOS QUE SE EJECUTAN EN OTRAS JVM.




En este modelo no se hace referencia directamente a un servidor, sino a una **interfaz remota** que esta implementado mediante sockets con su homólogo servidor denominado **skeleton**.



# QUE ES UN SKELETON?

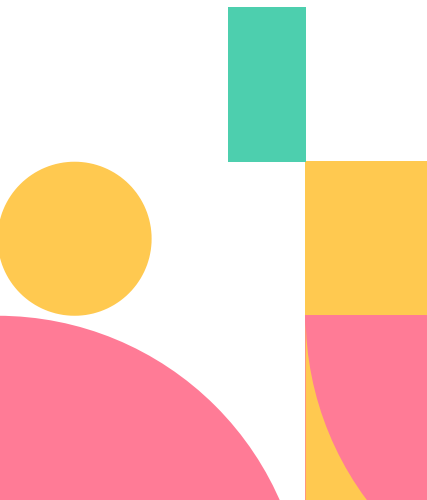
Es un esqueleto del **objeto remoto** que se encuentra del lado del servidor, es generado automáticamente durante la compilación y es el responsable de la **comunicación en red**.





# EVOLUCION DE STUBS y SKELETONS EN JAVA

Esta clase era necesaria en la versión 1.2 de Java, sin embargo, en versiones posteriores a esta ya no es necesaria la generación de skeletons gracias a la API de Java Reflection



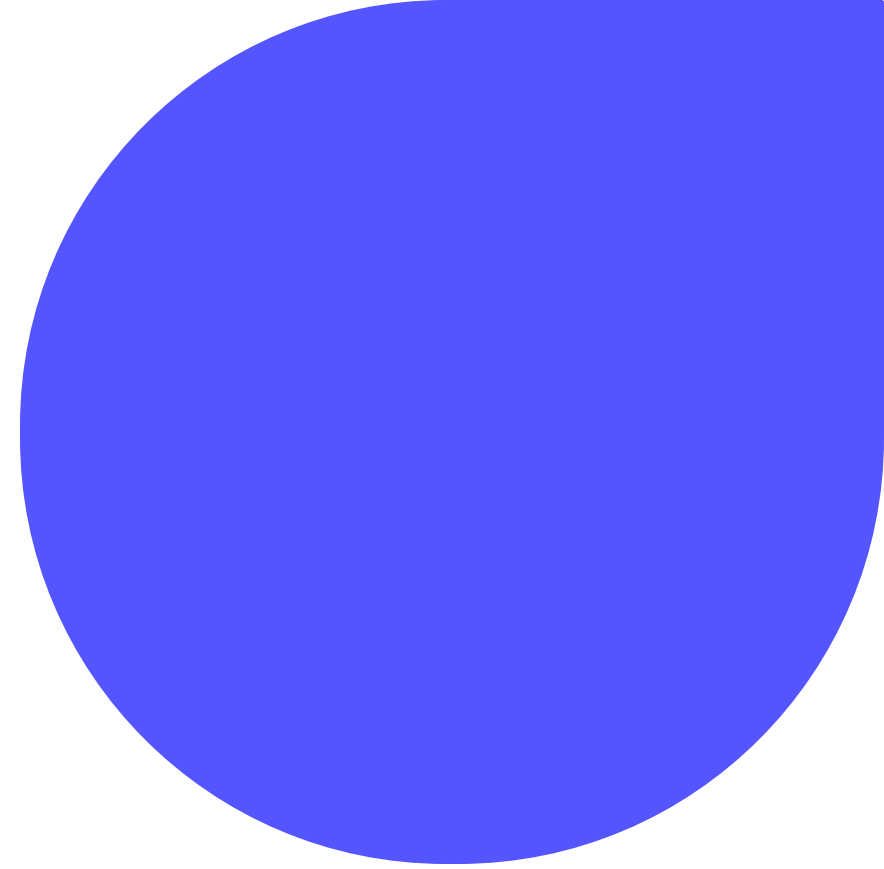
En la version 1.5 de Java podemos prescindir la generación de stubs, es por esto por lo que solo se usan si se requieren para la compatibilidad hacia atrás con otras aplicaciones de Java escritas en versiones más antiguas.





# IMPLEMENTACIÓN SKELETON

El modelo de objetos distribuido que propone Java permite que los objetos que se ejecutan en una JVM invoquen los métodos de objetos que se ejecutan en otras JVM.

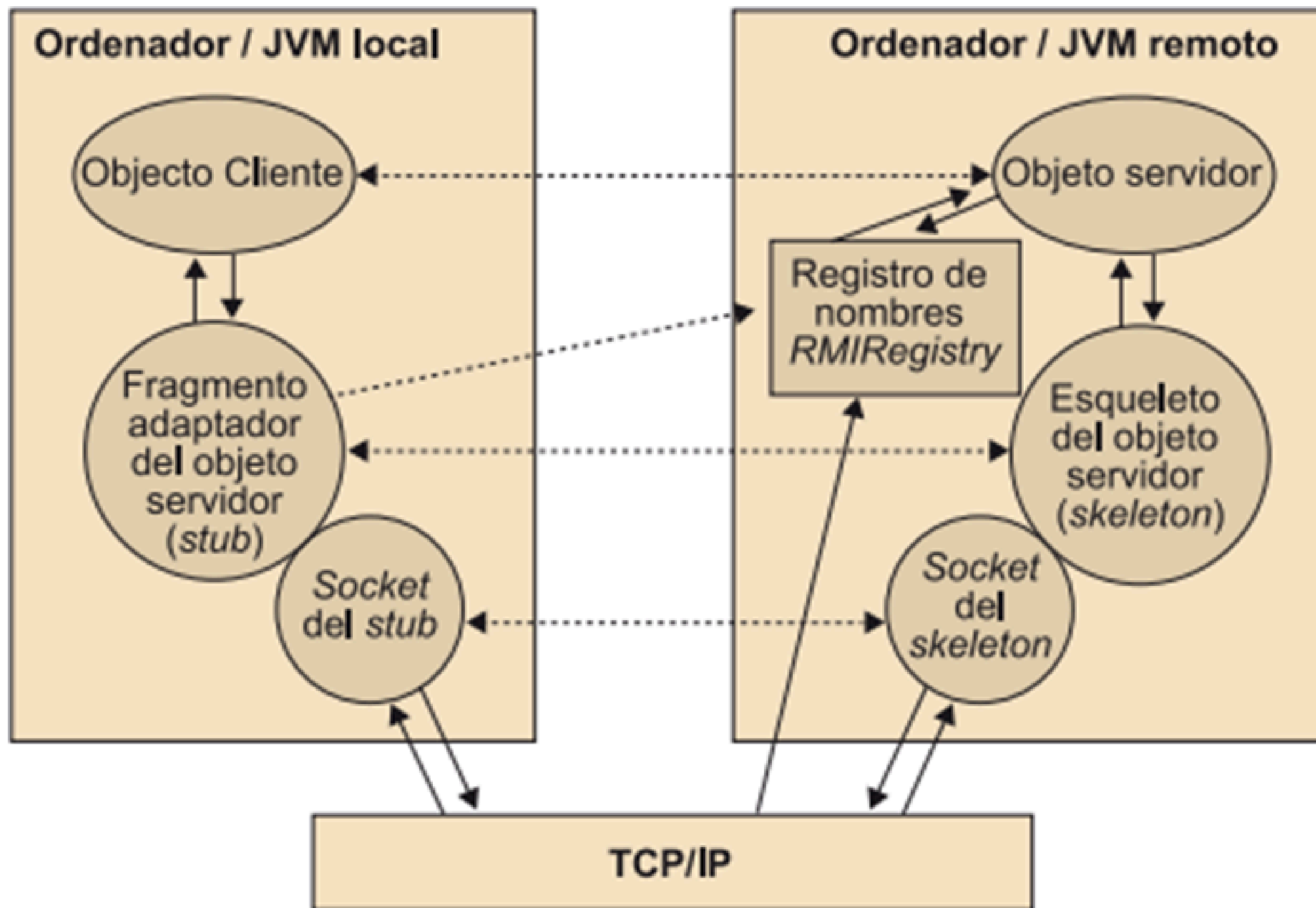


El objeto cliente nunca hace referencia directamente a un objeto servidor, sino a una interfaz remota que es implementada por este. Dicha interfaz actúa como servidor intermediario, también llamado fragmento adaptador stub, que implementa los métodos remotos.



Tanto los stubs como los skeletons son generados automáticamente como clases durante la compilación, y son los auténticos responsables de la comunicación





# IMPLEMENTACIÓN SERVIDOR

La parte servidora de una aplicación RMI contiene fundamentalmente los objetos remotos que los clientes buscan para invocar sus métodos y toda la infraestructura necesaria para recibir las invocaciones y devolver los valores de los métodos ejecutados.



# SERVIDOR

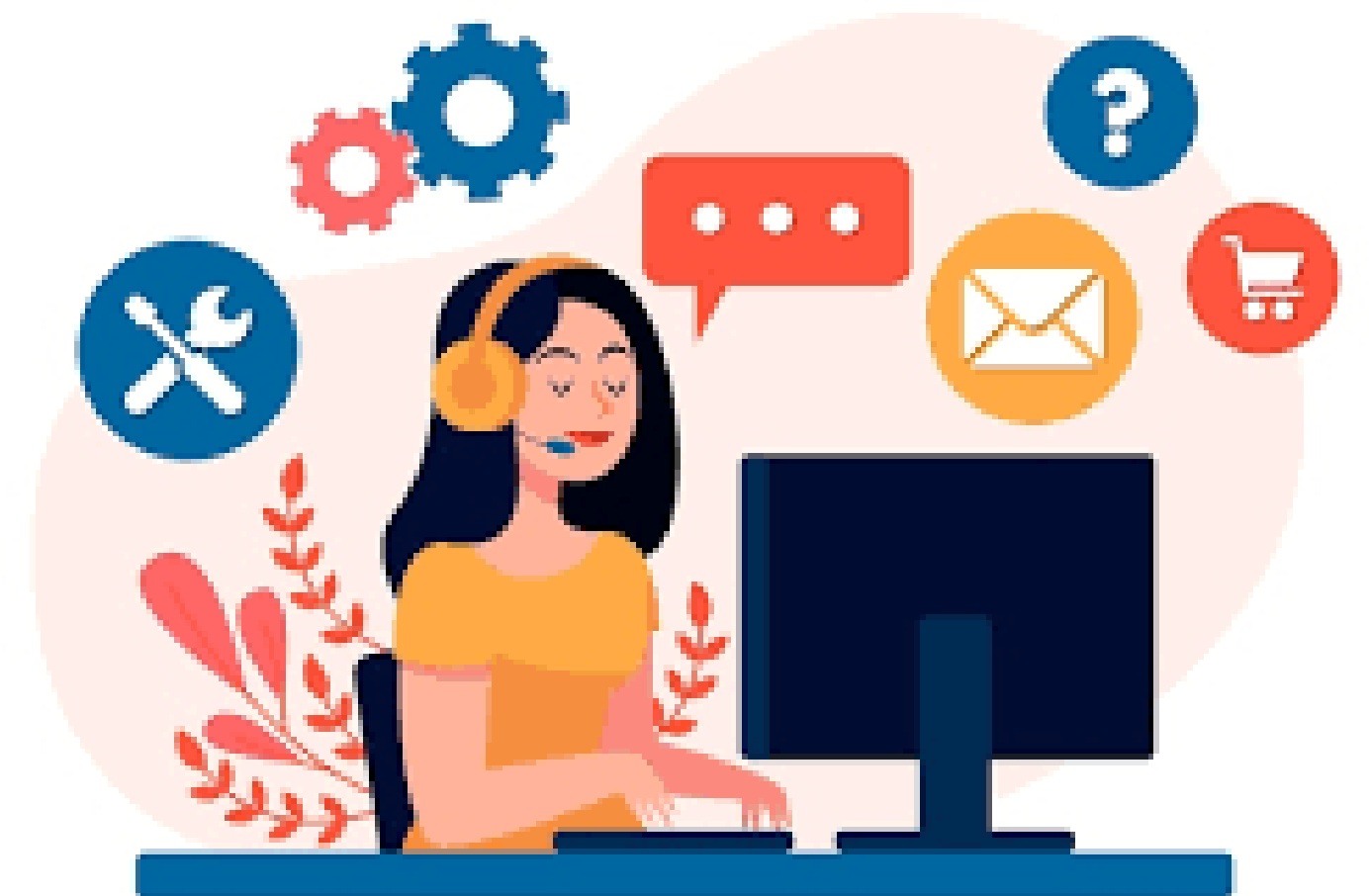
```
1 package skeleton;
2
3 import java.rmi.Naming;
4
5
6 public class server {
7     public static void main(String[] args) {
8         try {
9             LocateRegistry.createRegistry(5800);
10             helloImpl obj = new helloImpl();
11             Naming.rebind("//localhost:5800/Hello", obj);
12             System.out.println("Server ready");
13         } catch (Exception e) {
14             System.err.println("Server exception: " + e.toString());
15             e.printStackTrace();
16         }
17     }
18 }
```

# IMPLEMENTACIÓN

## CLIENTE

El cliente es el encargado de llamar a los métodos del objeto servidor. Para invocar un método remoto, el cliente debe realizar las siguientes tareas :

- Localizar el objeto remoto
- Invocar métodos remotos



En el lado del cliente es esencial contar con un stub del objeto servidor. Este stub se encarga de procesar y enviar las llamadas del cliente al servidor, así como de recibir los resultados de estas invocaciones.





# CLIENTE

```
1 package skeleton;
2
3 import java.rmi.Naming;
4
5 public class client {
6     public static void main(String[] args) {
7         try {
8             hello stub = (hello) Naming.Lookup("//localhost:5800/Hello");
9             String response = stub.sayHello();
10            System.out.println("Response: " + response);
11        } catch (Exception e) {
12            System.err.println("Client exception: " + e.toString());
13            e.printStackTrace();
14        }
15    }
16 }
17
```

# Conclusiones

LA EXPOSICIÓN SOBRE JAVA RMI (REMOTE METHOD INVOCATION) REVELA LA POTENCIA DE ESTA TECNOLOGÍA PARA LA COMUNICACIÓN ENTRE PROCESOS DISTRIBUIDOS. SU CAPACIDAD PARA TRANSMITIR OBJETOS JAVA ENTRE SISTEMAS PERMITE UNA INTEGRACIÓN FLUIDA Y EFICIENTE EN APLICACIONES DISTRIBUIDAS, OFRECIENDO UNA PERSPECTIVA FASCINANTE DEL POTENCIAL DE LA PROGRAMACIÓN DISTRIBUIDA EN EL DESARROLLO DE SOFTWARE MODERNO.

# Referencias

[1] CABALLÉ LLOBET, S. (2022). JAVA RMI (3.ª ED., PP. 29, 32-). ESPAÑA : FUNDACIÓ PER A LA UNIVERSITAT OBERTA DE CATALUNYA. ESPAÑA : FUNDACIÓ PER A LA UNIVERSITAT OBERTA DE CATALUNYA.