### [3. Sistemas basados en objetos distribuidos](https://m4gm.com/moodle/course/view.php?id=30" \l "section-3)

La clase de hoy vamos a iniciar con el tema Sistemas basados en objetos distribuidos.

**Paradigma de paso de mensajes**

Hasta ahora hemos desarrollado programas distribuidos utilizando paso de mensajes.

El paradigma de **paso de mensajes** es el modelo natural para el desarrollo de sistemas distribuidos, ya que reproduce la comunicación entre las personas.

En el paradigma de paso de mensajes, las computadoras comparten los datos utilizando mensajes. El programador debe serializar los datos antes de enviarlos, y des-serializar los datos después de recibirlos.

El desarrollo de sistemas basados en paso de mensajes es complejo debido a que el programador debe controlar el intercambio de los mensajes, además de desarrollar la funcionalidad propia del sistema.

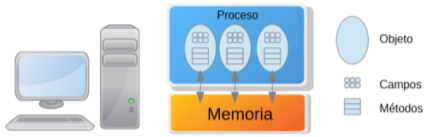
El paradigma de paso de mensajes es **orientado a datos**.

**Objetos locales**

Un objeto encapsula variables (campos) y funciones (métodos). Las variables guardan el estado del objeto y los métodos permiten modificar y acceder el estado del objeto.

Un objeto local es aquel cuyos métodos son invocados por un proceso local, es decir, un proceso que ejecuta en la misma computadora dónde reside el objeto.

Los objetos locales comparten el espacio de direcciones, en otras palabras, los objetos locales son objetos que residen en la misma memoria.



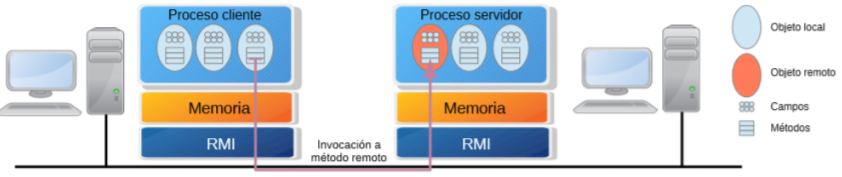
**Objetos remotos**

Un objeto remoto es aquel cuyos métodos son invocados por procesos remotos, es decir, procesos que ejecutan en una computadora remota conectada mediante una red.

Los objetos que se encuentran en diferentes computadoras no comparten el espacio de direcciones, por tanto, solo comparten valores pero no referencias.

La siguiente figura muestra un proceso cliente y un proceso servidor que ejecutan en diferentes computadoras. En el proceso cliente un método local invoca un método remoto, el cual forma parte de un objeto contenido en el proceso servidor.

En este caso, la invocación de los métodos remotos se realiza mediante una capa llamada RMI (*Remote Method Invocation*).



**Paradigma de objetos distribuidos**

El paradigma de objetos distribuido combina objetos locales y objetos remotos. La ventaja que tiene, comparado con el paradigma de paso de mensajes, es que el paradigma de objetos distribuidos representa una abstracción sobre el paso de mensajes, por tanto el programador no debe preocuparse por controlar el paso de mensajes entre los nodos.

El paradigma de objetos distribuidos es **orientado a la acción**, ya que se basa en la acción que realiza el método remoto invocado.

**Remote Method Invocation**

En un sistema que utiliza RMI existe un proceso llamado *registry* el cual hace las funciones de servidor de nombres.

En cada nodo, hay un proceso servidor el cual registra en el servidor de nombreslos objetos que exportará. Cada objeto exportado por el servidor será identificado mediante una URL.

Para acceder a un objeto remoto, el proceso cliente consulta el servidor de nombresutilizando la URL, si el objeto es encontrado, entonces el servidor de nombresregresa al cliente una referencia que apunta al objeto remoto. Entonces el proceso cliente utiliza la referencia para invocar los métodos del objeto remoto, los cuales se ejecutan en el servidor.

El paso de parámetros y regreso de resultado es manejado automáticamente por la capa RMI.

**Java RMI**

Java RMI es un API que implementa la invocación de métodos remotos. JDK incluye un servidor de nombres llamado **rmiregistry**, esta aplicación se encuentra en el directorio bin del JDK

**¿Cómo usar Java RMI?**

Para utilizar Java RMI se debe seguir los siguientes pasos:

1. Para cada objeto remoto se debe crear una interface **I** que defina el prototipo de cada método a exportar. Es necesario declarar que los métodos remotos pueden producir la excepción java.rmi.RemoteException. La interface **I** debe heredar de java.rmi.Remote.

2. El código de los métodos remotos se debe escribir en una clase **C** que implemente la interface **I**. La clase **C** debe ser una subclase de java.rmi.server.UnicastRemoteObject. El constructor default de la clase **C** debe invocar el constructor de la superclase. Es necesario declarar que los métodos remotos pueden producir la excepción java.rmi.RemoteException.

3. El proceso servidor deberá registrar la clase **C** invocando el método bind() o el método rebind() de la clase java.rmi.Naming. A los métodos bind() y rebind() se les pasa como parámetros la URL correspondiente al objeto remoto y una instancia de la clase **C**. La URL tiene la siguiente forma: **rmi://***ip***:***puerto***/***nombre,* donde *ip* es la dirección IP de la computadora dónde ejecuta el programa **rmiregistry**, *puerto* es el número de puerto utilizado por rmiregistry (se puede omitir si rmiregistry utiliza el puerto default 1099) y *nombre* es el nombre con el que identificaremos el objeto.

4. El proceso cliente deber invocar el método lookup() de la clase java.rmi.Naming para obtener una referencia al objeto remoto. El método lookup() regresa una instancia de la clase Remote, la cual se debe convertir al tipo de la interface **I** mediante casting. Utilizando la referencia, el proceso cliente invocará los métodos remotos de la clase **C**.

Por razones de seguridad, la aplicación rmiregistry se debe ejecutar en la misma computadora dónde ejecuta el servidor.

Por default la aplicación rmiregistry utiliza el puerto 1099, si se utiliza otro puerto, se deberá pasar el número de puerto como argumento al ejecutar rmiregistry.

Se puede notar que el proceso servidor permanece en ejecución debido a que los métodos bind() y rebind() crean threads que no terminan.

**Ejemplo de Java RMI**

Como vimos anteriormente, para crear una aplicación que utilice Java RMI es necesario crear una interface, una clase, y dos programas (un cliente y un servidor).

En este caso, vamos a crear un objeto remoto que exportará los siguiente métodos:

* mayusculas(), recibe como parámetro una cadena de caracteres y regresa la misma cadena convertida a mayúsculas.
* suma(), recibe como parámetros dos enteros y regresa la suma.
* checksum(), recibe como parámetro una matriz de enteros y regresa la suma de todos los elementos de la matriz.

Primeramente creamos una interface que incluya los prototipos de los métodos a exportar:

public interface **InterfaceRMI** extends Remote

{

public String mayusculas(String name) throws RemoteException;

public int suma(int a,int b) throws RemoteException;

public long checksum(int[][] m) throws RemoteException;

}

Ahora escribimos la clase **ClaseRMI** la cual va a contener el código de los métodos definidos en la interface **InterfaceRMI**. Notar que la clase **ClaseRMI** es subclase de UnicastRemoteObject e implementa la interface **InterfaceRMI**.

public class **ClaseRMI** extends UnicastRemoteObject implements **InterfaceRMI**

{

// es necesario que el contructor **ClaseRMI**() invoque el constructor de la superclase

public ClaseRMI() throws RemoteException

{

super( );

}

public String mayusculas(String s) throws RemoteException

{

return s.toUpperCase();

}

public int suma(int a,int b)

{

return a + b;

}

public long checksum(int[][] m) throws RemoteException

{

long s = 0;

for (int i = 0; i < m.length; i++)

for (int j = 0; j < m[0].length; j++)

s += m[i][j];

return s;

}

}

La clase **ServidorRMI** registra en el rmiregistry una instancia de la clase **ClaseRMI** utilizando el método rebind().

public class **ServidorRMI**

{

public static void main(String[] args) throws Exception

{

String url = "rmi://localhost/prueba";

ClaseRMI obj = new ClaseRMI();

// registra la instancia en el rmiregistry

Naming.rebind(url,obj);

}

}

El cliente **ClienteRMI** obtiene una referencia al objeto remoto utilizando el método lookup(), esta referencia es utilizada para invocar los métodos remotos.

public class **ClienteRMI**

{

public static void main(String args[]) throws Exception

{  
 // en este caso el objeto remoto se llama "prueba", notar que se utiliza el puerto default 1099

String url = "rmi://localhost/prueba";

// obtiene una referencia que "apunta" al objeto remoto asociado a la URL

InterfaceRMI r = (InterfaceRMI)Naming.lookup(url);

System.out.println(r.mayusculas("hola"));

System.out.println("suma=" + r.suma(10,20));

int[][] m = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};

System.out.println("checksum=" + r.checksum(m));

}

}

**Actividades individuales a realizar**

1. Compilar la interface [InterfaceRMI.java](https://m4gm.com/moodle/mod/resource/view.php?id=1780), la clase [ClaseRMI.java](https://m4gm.com/moodle/mod/resource/view.php?id=1781) y los programas [ClienteRMI.java](https://m4gm.com/moodle/mod/resource/view.php?id=1782) y [ServidorRMI.java](https://m4gm.com/moodle/mod/resource/view.php?id=1783).
2. En una ventana de comandos de Windows (o una terminal de Linux) ejecutar el programa **rmiregistry**.
3. En una ventana de comandos de Windows (o una terminal de Linux) ejecutar el programa ServidorRMI. Notar que el servidor queda en ejecución.
4. En una ventana de comandos de Windows (o una terminal de Linux) ejecutar el programa ClienteRMI. El cliente invoca el método lookup() para obtener del rmiregistry una referencia al objeto remoto, entonces invoca los métodos del objeto remoto los cuales se ejecutan en el servidor.