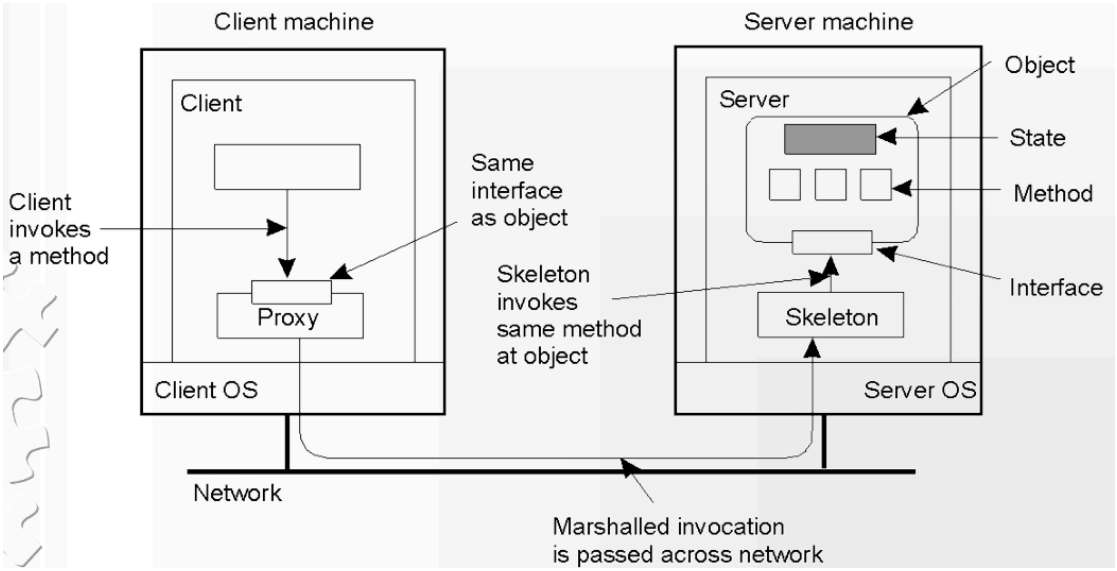
INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Desarrollo de Sistemas Distribuidos

Práctica 5 ‘Objetos Distribuidos - RMI’



Docente:

Carreto Arellano Chadwick

Grupo:

4CM11

Alumnos:

Rodríguez Hernández Marlene Guadalupe

Fecha: martes 1, abril 2025

Antecedentes

Los objetos distribuidos son aquellos que están gestionados por un servidor y sus clientes invocan sus métodos utilizando un "método de invocación remota". El cliente invoca el método mediante un mensaje al servidor que gestiona el objeto, se ejecuta el método del objeto en el servidor y el resultado se devuelve al cliente en otro mensaje.

Hay distintos métodos, como los siguientes:

- **RMI,** Remote Invocation Method (Sistema de Invocación Remota de Métodos) : Fue el primer framework para crear sistemas distribuidos de Java. Este sistema permite, a un objeto que se está ejecutando en una Máquina Virtual Java (VM), llamar a métodos de otro objeto que está en otra VM diferente. Esta tecnología está asociada al lenguaje de programación Java, es decir, que permite la comunicación entre objetos creados en este lenguaje.
- **DCOM,** Distributed Component Object Model: El Modelo de Objeto Componente Distribuido, está incluido en los sistemas operativos de Microsoft. Es un juego de conceptos e interfaces de programa, en el cual los objetos de programa del cliente, pueden solicitar servicios objetos del programa servidores, en otros ordenadores dentro de una red. Esta tecnología está asociada a la plataforma de productos Microsoft.
- **CORBA,** Common Object Request Broker Architecture: Tecnología introducida por el Grupo de Administración de Objetos OMG, creada para establecer una plataforma para la gestión de objetos remotos independiente del lenguaje de programación.

El método que utilizaremos en esta práctica es el RMI, las metas que se pretenden alcanzar al soportar objetos distribuidos en java son:

- Proporcionar invocación remota de objetos que se encuentran en MVs diferentes.
- Soportar llamadas a los servidores desde los applets.
- Integrar el modelo de objetos distribuidos en el lenguaje Java de una manera natural, conservando en medida de lo posible la semántica de los objetos Java.
- Hacer tan simple como sea posible la escritura de aplicaciones distribuidas.
- Preservar la seguridad proporcionada por el ambiente Java.
- Proporcionar varias semánticas para las referencias de los objetos remotos (persistentes, no persistentes y de "activación retardada").

Planteamiento del problema

Siguiendo los objetos distribuidos, implementaremos el servidor RMI y a su vez crearemos un cliente RMI que pueda interactuar con el servidor.

Propuesta de solución

Vamos a definir la interfaz remota (Carrera de Tortugas) y declararemos los métodos remotos que los clientes podrán llamar que serán avanzar y finalizar, para que podamos implementar el servidor RMI y crear al cliente que pueda interactuar con el servidor, de esta forma cada cliente (tortuga) se registra en el servidor y este lleva el control del avance de cada cliente (tortuga), así el cliente (tortuga) muestra en una GUI los avances de todas las tortugas (simulando una carrera en tiempo real), por último se usa RMI para que el cliente (tortugas) pueda comunicarse con el servidor que tiene la lógica de la carrera.

Materiales y métodos empleados

- uso VSCode con la extensión de Java
- librería java.rmi.server.UnicastRemoteObject
- librería java.rmi.registry.Registry
- librería java.rmi.registry.LocalRegistry
- librería java.rmi.RemoteException
- librería java.util.HashMap
- librería java.util.Map
- librería java.util.UUID
- compilo con javac Servidor.java

Desarrollo de la solución

A continuación se explica el funcionamiento de cada uno de los archivos del código:

- CarreraTortugas.java

CarreraTortugas.java > ...

```java
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Map;

public interface CarreraTortugas extends Remote {
    String registrarTortuga() throws RemoteException;
    int avanzar(String idTortuga) throws RemoteException;
    boolean haFinalizado(String idTortuga) throws RemoteException;
    String obtenerGanador() throws RemoteException;
    Map<String, Integer> obtenerEstadoCarrera() throws RemoteException;
}
```

Este funciona como la interfaz remota, en la cual se definen los métodos que el servidor ofrece a los clientes, como:

- registrarTortuga(), para registrar una nueva tortuga
- avanzar(id), para hacer que una tortuga avance
- obtenerEstadoCarrera(), para saber como van todas las tortugas
- obtenerGanador(), para saber quien gano

Esta interfaz extiende Remote y todos los métodos lanzan RemoteException.

- ServidorCarrera.java, aqui se hace la implementación en el servidor

```
public class ServidorCarrera extends UnicastRemoteObject implements CarreraTortugas {
    private static final int META = 100;
    private Map<String, Integer> posiciones;
    private String ganador = null;
```

se implementa la interfaz CarreraTortuga y se guarda el estado de cada tortuga con un Map<String, Integer>

```
public Map<String, Integer> obtenerEstadoCarrera() throws RemoteException {
    return new HashMap<>(posiciones); // Devolvemos una copia para evitar cambios no controlados
}
```

también, manteniendo lo de cada práctica hacemos que cada tortuga avance aleatoriamente para simular que cada una corre a distinta velocidad y a su vez se declara al ganador cuando una tortuga llega a la meta

```
public int avanzar(String idTortuga) throws RemoteException {
    if (!posiciones.containsKey(idTortuga) || ganador != null) {
        return posiciones.getOrDefault(idTortuga, defaultValue:0);
    }

    int avance = (int) (Math.random() * 10);
    posiciones.put(idTortuga, posiciones.get(idTortuga) + avance);

    System.out.println("Tortuga " + idTortuga + " avanzó a: " + posiciones.get(idTortuga));

    if (posiciones.get(idTortuga) >= META && ganador == null) {
        ganador = idTortuga;
        System.out.println("¡Tortuga " + idTortuga + " ha ganado la carrera!");
    }

    return posiciones.get(idTortuga);
}
```

Se expone el objeto remoto con:

```
public static void main(String[] args) {
    try {
        ServidorCarrera servidor = new ServidorCarrera();
        Registry registro = LocateRegistry.createRegistry(port:1099);
        registro.rebind(name:"CarreraTortugas", servidor);
        System.out.println(x:"Servidor RMI listo.");
    } catch (RemoteException e) {
        e.printStackTrace();
    }
}
```

específicamente en la parte de

Registry Registro = LocalRegistry.createRegistry(1099);

registro.rebind(“CarreraTortugas”, servidor);

- ClienteGUI.java, esta es la interfaz grafica del cliente, la cual se conecta al servidor usando el siguiente código

```
private void conectarServidor() {
    try {
        Registry registro = LocateRegistry.getRegistry(host:"localhost", port:1099);
        carrera = (CarreraTortugas) registro.lookup(name:"CarreraTortugas");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

así, registra a la tortuga automáticamente al iniciar, después abre una GUI con barras de progreso para cada tortuga (JProgressBar) y usa un hilo (Thread para actualizar constantemente el estado de la carrera consultando al servidor.

Finalmente muestra al ganador final

```java
private void actualizarCarrera(Map<String, Integer> estado, String ganador) {
    for (String id : estado.keySet()) {
        if (!barras.containsKey(id)) {
            JProgressBar barra = new JProgressBar(min:0, max:100);
            barra.setStringPainted(b:true);
            barras.put(id, barra);
            add(barra);
            revalidate();
        }
        barras.get(id).setValue(estado.get(id));
    }

    if (ganador != null) {
        labelGanador.setText("¡Tortuga ganadora: " + ganador + "!");
    }
}
```

Resultados

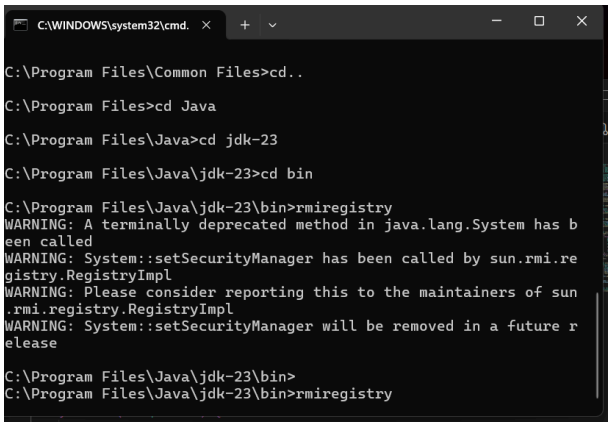
Una vez compilados y ejecutados los archivos, se lanza el servidor el cual se empieza a escuchar en el puerto 1099, entonces iniciamos el cliente, el cual:

- se conecta al servidor
- se registra como una tortuga
- empieza a preguntar al servidor el estado de la carrera

Así cada cliente (tortuga) irá avanzando cada 500ms y el servidor actualiza sus posiciones, de esta forma cuando una yegua a 100, se mostrará quién ganó y todos los demás se detendrán.

Para compilar y ejecutar

- En cmd, corremos el rmiregistry



- Compilamos, y esperamos a que se crean los archivos .class e iniciamos el servidor en otra terminal

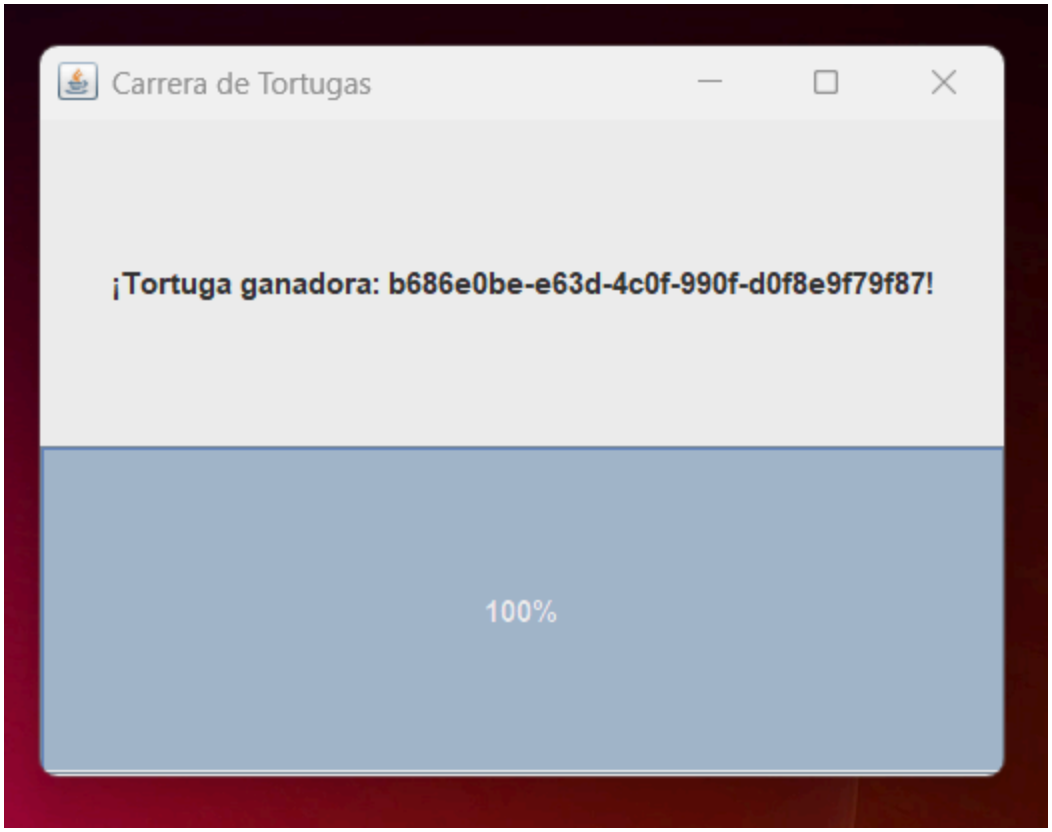
```
C:\Users\kirit\Downloads\ESCOM\DSD-R\Objetos-Distribuidos-RMI>javac *.java

C:\Users\kirit\Downloads\ESCOM\DSD-R\Objetos-Distribuidos-RMI>java ServidorC
arrera
Servidor RMI listo.
Tortuga registrada con ID: b686e0be-e63d-4c0f-990f-d0f8e9f79f87
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 3
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 10
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 18
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 21
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 22
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 22
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 31
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 37
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 46
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 52
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 61
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 61
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 61
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 61
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 67
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 73
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 74
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 76
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 77
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 86
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 86
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 86
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 91
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 94
Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 avanzó a: 100
¡Tortuga b686e0be-e63d-4c0f-990f-d0f8e9f79f87 ha ganado la carrera!
```

- También conectamos un cliente en otra terminal

```
C:\Users\kirit\Downloads\ESCOM\DSD-R\Objetos-Distribuidos-RMI>java ClienteGUI
Tortuga registrada con ID: b686e0be-e63d-4c0f-990f-d0f8e9f79f87
```

- Por último vemos como se abre la ventana con el progreso del cliente (tortuga) en la carrera, solo fue una por lo que solamente ganó una en este caso



Conclusiones

Gracias a esta nueva práctica he comprendido que un servidor RMI puede manejar diferentes clientes (o en este caso varias tortugas), y de esta forma los clientes RMI (las tortugas) se conectan y avanzan en la carrera, también que un sistema distribuido es aquel en donde clientes y servidores pueden ejecutarse en diferentes instancia tanto como en diferentes computadoras, pero por ahora pude observar cómo se conectaban varias tortugas como anteriormente en prácticas anteriores pero ahora de diferente forma, utilizando nuevos métodos y distintas funciones para lograrlo.

Enlace a la práctica en GitHub

https://github.com/Marlene0807/Desarrollo-de-Sistemas-Distribuidos.git

Referencias

OBJETOS DISTRIBUIDOS - SISTEMAS DISTRIBUIDOS

Java RMI

Comunicación entre Procesos

Objetos-Distribuidos.pdf

RMIv2019