

Java RMI.

Use lo de allí, como si lo tuviera aquí.

*-¿Cual es al diferencia entre programar un juego con modo individual y uno con modo multijugador?*

-Ahora, con Java RMI (Java Remote Method Invocation), ninguna.

*-¿Y qué es eso Java RMI*?

*–*Es un mecanismo de los programas en Java, para comunicar diferentes máquinas, utilizado en las aplicaciones distribuidas.

*-¿Como empiezo a utilizar Java RMI en mis aplicaciones distribuidas?*

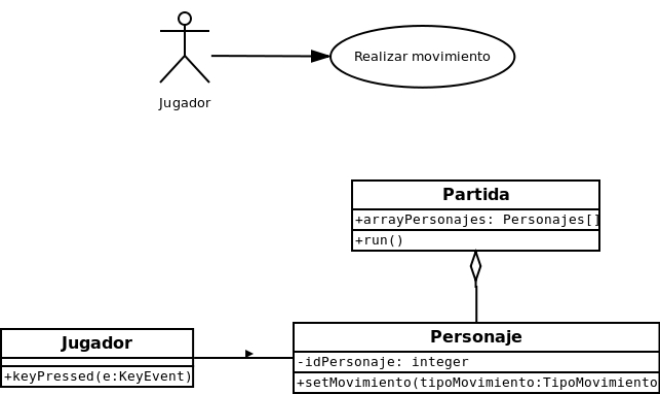
-Muy fácil, el guión a seguir, sin entrar detalles técnicos, sería:

1. Codifique su clase, instánciela y márquela como remota para que esté accesible a través de la red.
2. Invoque un método de dicha clase desde cualquier otra máquina.
3. Espere el resultado del método (si es que devuelve algún valor)

Después de haberte vendido la moto, vamos a lo interesante, *¿qué tiene que ver Java RMI con los videojuegos on-line multijugador?* ¿Cómo puedo usar/aplicar esta tecnología?

Tengo la necesidad de comunicar los distintos jugadores, para que cada uno conozca la situación actual del juego y, por ejemplo, pueda informar cada uno de los movimientos que realizan a los demás. Pues yo, me planteo crear un diagrama de clases olvidándome de donde estará cada clase instanciada (vamos, todo como si estuviera en local) y prestando especial interés en las relaciones de uso/asociación que hay entre las clases (porque alguna de ellas, se harán vía invocación remota)

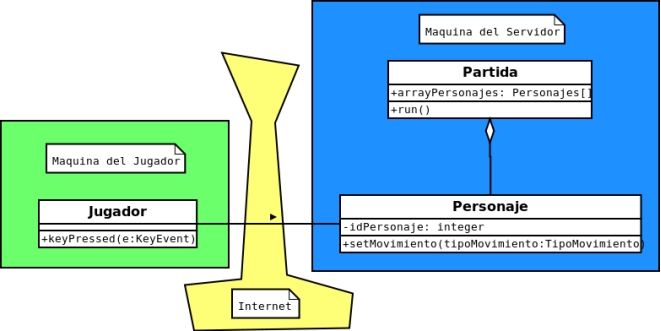
Me fijaré, para ilustrar con un ejemplo, en la funcionalidad de comunicar el movimiento que quiere realizar un jugador con su personaje/ficha/vehículo



En la situación anterior, si nos encontramos en una partida on-line, sabemos:

1. El servidor, el escenario/tablero/circuito y los personajes/fichas/vehículos estarán en el servidor.
2. En cada máquina del jugador tendremos un aplicación cliente, que controla los eventos de teclado/ratón/gamepad para enviarlos al servidor.

Para realizar esta comunicación, el caso normal es generar un sistema de comunicación UDP o TCP, con sus socket y métodos, para cuando el jugador quiera comunicar a su personaje (alojado en el servidor)



Pues, Java RMI, ya nos da esa implementación sobre TCP, y gracias a ello nos olvidamos de socket y demás. Simplemente debemos de definir mediante una interfaz, que métodos (en el ejemplo setMovimiento(tipoMovimiento)) podrán ser utilizados remotamente. Creamos el objeto que implemente esa interfaz, en caso del ejemplo: Personaje, creamos una instancia y la registramos en rmiregistry, obtendremos una referencia de objeto remoto, la cual usará el Jugador,para llamar a los métodos remotos que definimos anteriormente. Además, los métodos remotos definidos, pueden lanzar RemoteException, para dar soporte a cualquier tipo de error que pudiera surgir en la llamada, o dentro de la ejecución del método.

Lo importante de todo esto, no es intentar explicar qué y cómo es Java RMI (hay miles de tutoriales por internet), si no saber lo fácil y cómodo que resulta utilizarlo. Me parece muy simple el diseño de las aplicaciones, y por extensión, rápido de entender. En contra veo, puede ser mas lento que la implementación tradicional mediante socket UDP/TCP, ya que cada llamada a

un método remoto, implica establecimiento de comunicación TCP, envio y confirmación.

En mi proyecto en cuestión, lo estoy utilizando en muchas funcionalidades a implementar, por ejemplo:

1. Un Jugador comprueba si existe un servidor de partidas en una IP determinada, se traduce en un metodo remoto, devolviendo información a cerca del servidor, o RemoteException, si no existe o está disponible dicho servidor
2. El jugador anterior, pide unirse al servidor anterior (después de comprobar que está disponible) el método remoto del servidor, le devuelve la referencia del Personaje asignado que entra en la partida.
3. En el desarrollo de una partida, un jugador pide constantemente la situación de la partida, devolviendo el servidor un objeto con toda la información de situación de los demás contrincantes.