

**Ingeniería en Sistemas**

**Computacionales**

**Raúl Tostado Blázquez I.D. 145901**

**Materia: Sistemas Distribuidos**

**Nombre del Trabajo: Práctica Java RMI I y II**

**Profesora: Ofelia D. Cervantes Villagómez**

**San Andrés Cholula, Puebla a**

**18 de marzo de 2015**

Reporte práctica Java RMI I

**Introducción a la práctica**

Java RMI (Remote Method Invocation), es una tecnología desarrollada por Java para invocar métodos remotos. Proporciona un mecanismo simple para la comunicación de Servers y Clients en aplicaciones distribuidas basadas en Java.

La construcción de una aplicación distribuida con RMI incluye:

1. Serialización (**Marshalling**) de parámetros y objetos en Java en el cliente para enviar al servidor
2. Invocación desde el cliente, del **Método Remoto** (RMI) a ejecutar en el servidor
3. Serialización (**Marshalling**) en el servidor para enviar respuesta al **Cliente**
4. Recepción de la respuesta por parte del **Cliente** y ejecución local

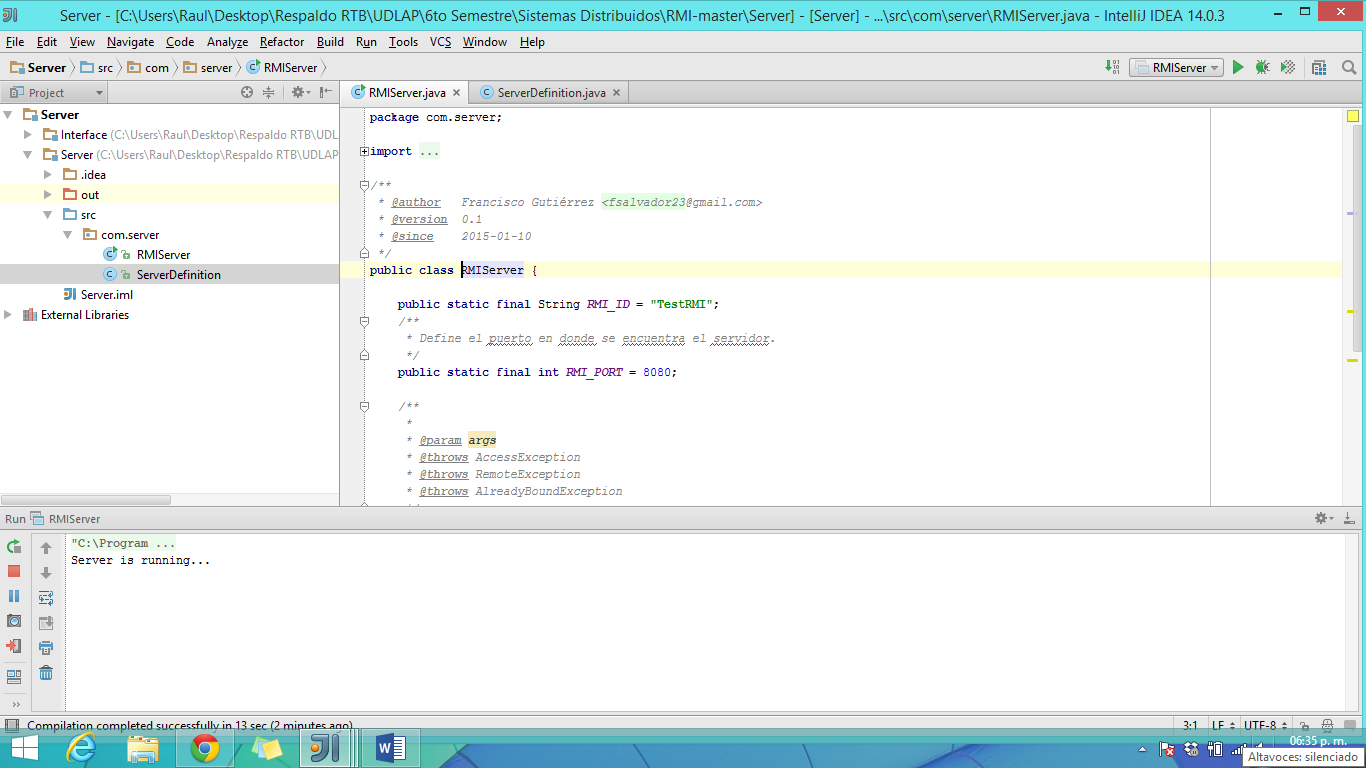
**Antes de comenzar:**

EL primer paso fue hacer una lectura detallada a la práctica, al igual que realizar la descarga de los archivos correspondientes de GitHub y otros complementos necesarios para trabajar. Posteriormente se procedió a conectar los tres programas mediante Intellij.

**Actividad 1: Análisis**

En **RMI** se definen **Client**, **Server** e **Interface**. La Interfaz define los métodos que deseamos implementar en Server. El **Cliente** solicita a **Server** los métodos que desea sean ejecutados a través de un registro que sólo debe realizarse una vez.

* Ejecutar **Server**: Click derecho sobre el proyecto, "**Run As** > **Java Application**".
* Ejecutar **Client**: Click derecho sobre el proyecto, "**Run As** > **Java Application**".
* Analizar el diagrama que muestra la arquitectura de la aplicación distribuida:
* Observar el comportamiento de la aplicación cuando el método remoto de prueba test se invoca con diferentes cadenas.

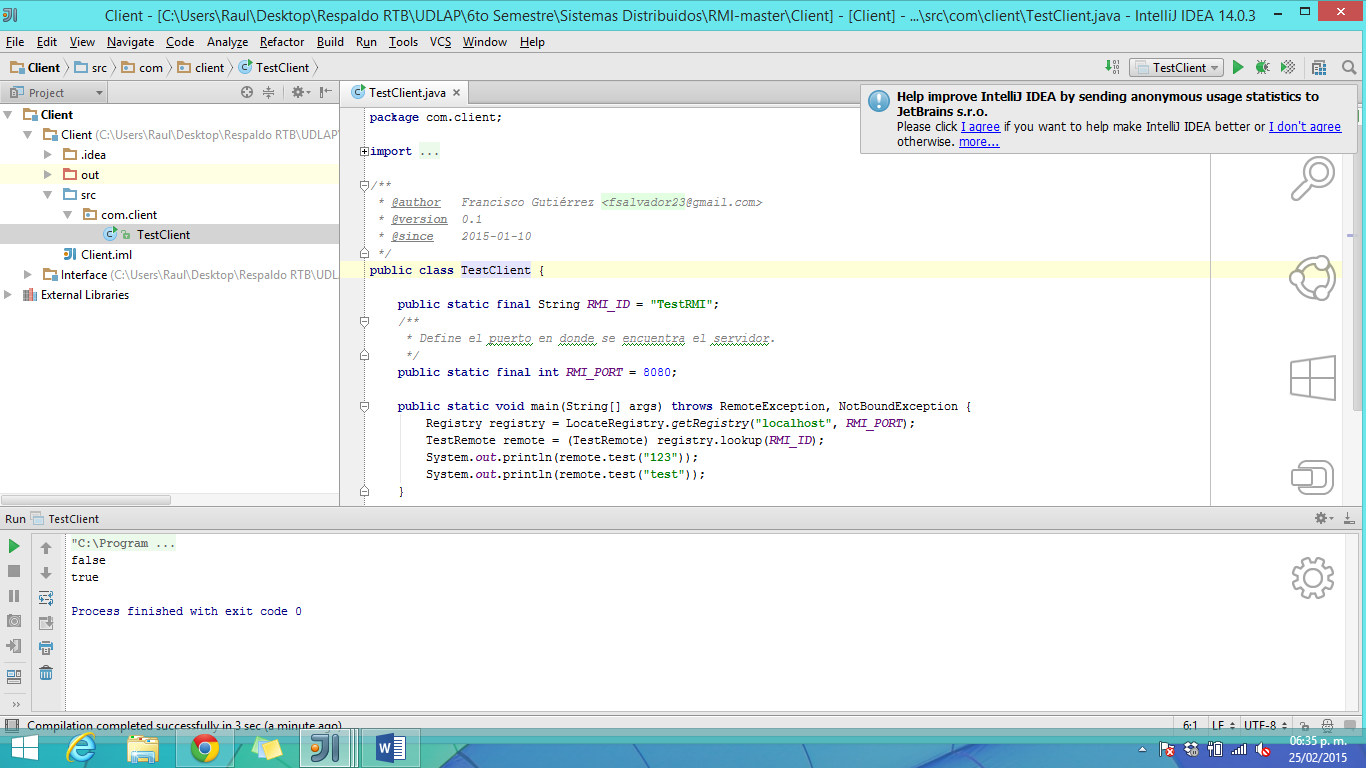


* Analizar la clase **ServerDefinition.java** y **TestClient.java** para comprender los resultados.

**R=** Dentro del servidor es donde se declaran los métodos con los que se va a trabajar y el cliente es el que va a llamar los métodos del servidor, se proporciona algún valor o cadena dependiendo del ejercicio y el servidor se comunicará con el cliente para que posteriormente le devuelva el resultado esperado

* Explicar en el **reporte** por qué aparecen como resultado **false** y **true**

**R=** Para poder explicar cómo funciona el método es necesario analizarlo primero. En un principio manda false porque estamos comparando la cadena “123” con “test” y al ser cadenas diferentes el programa nos devuelve un **false**, mientras que en el segundo ejemplo comparamos “test” con “test” y al ser iguales nos devuelve **true.**

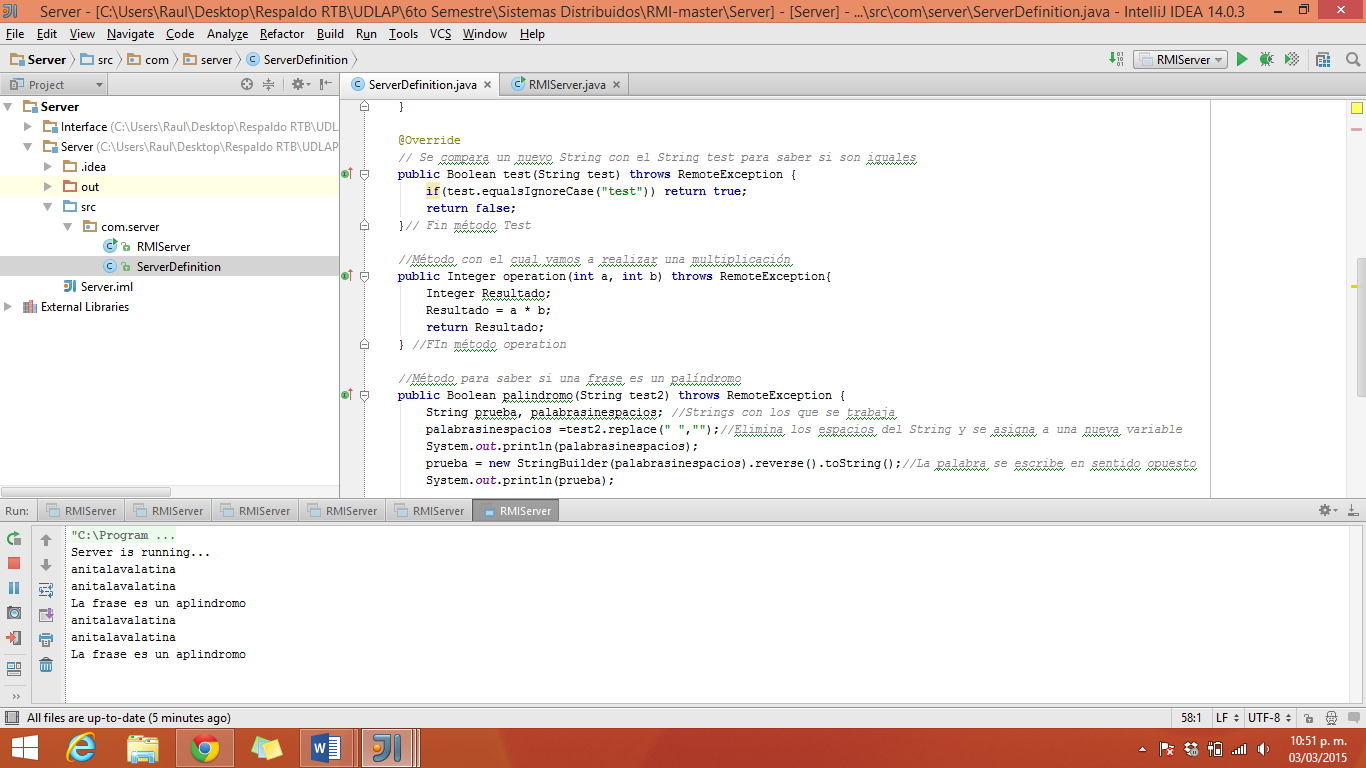


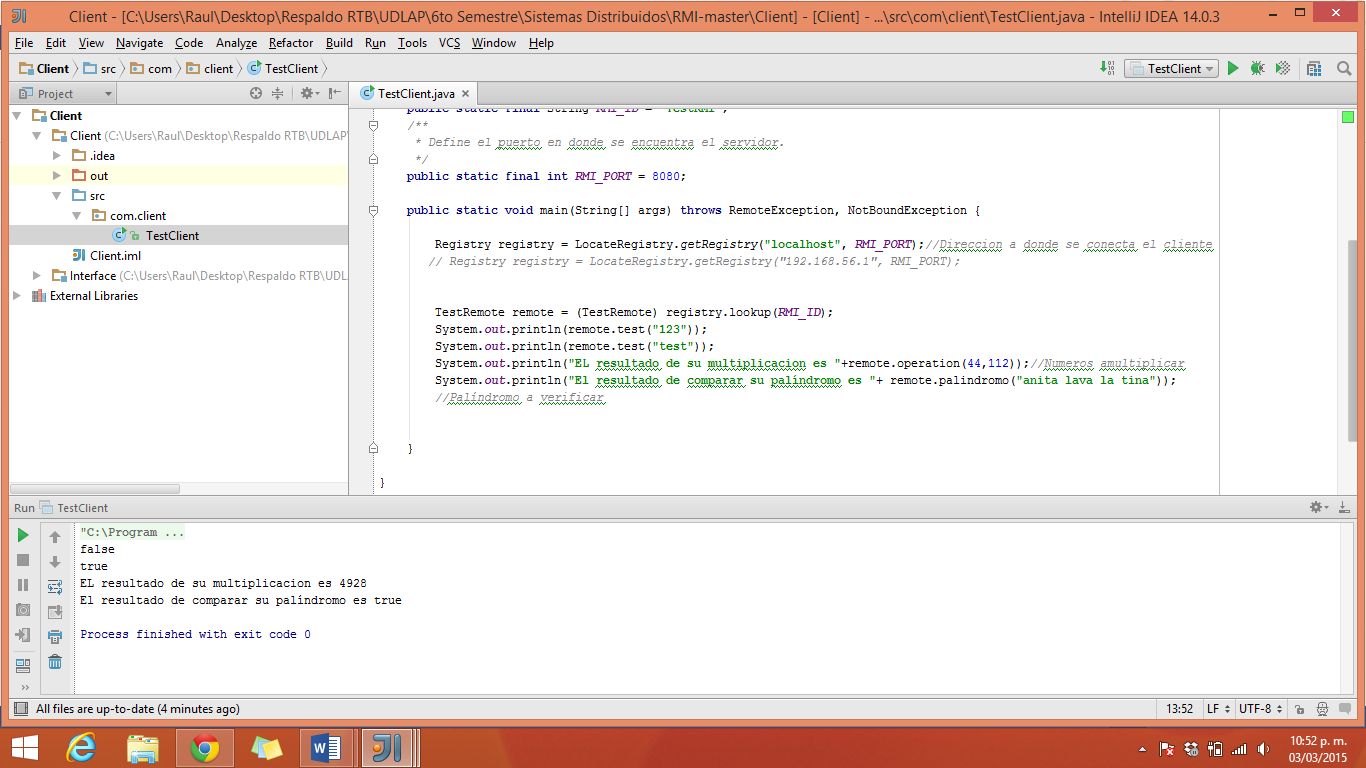
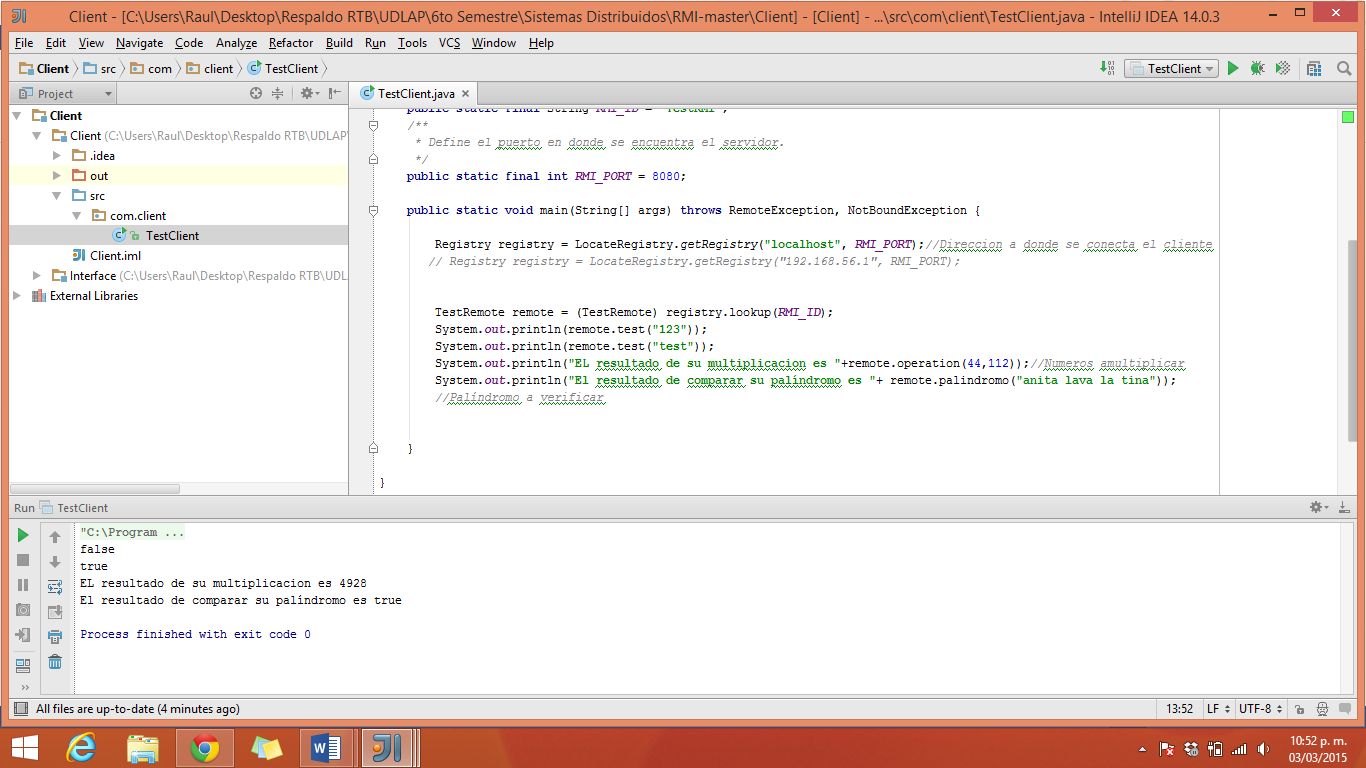
**Actividad 2 - Desarrollo**

En ésta actividad deberás modificar el **Works pace** en Eclipse que se te ha entregado. Como habrás observado el proyecto ejecuta un método Test en el **Server**. El objetivo de ésta actividad es **definir** e **implementar** dos nuevos métodos en el Server que llamaremos desde el Client.

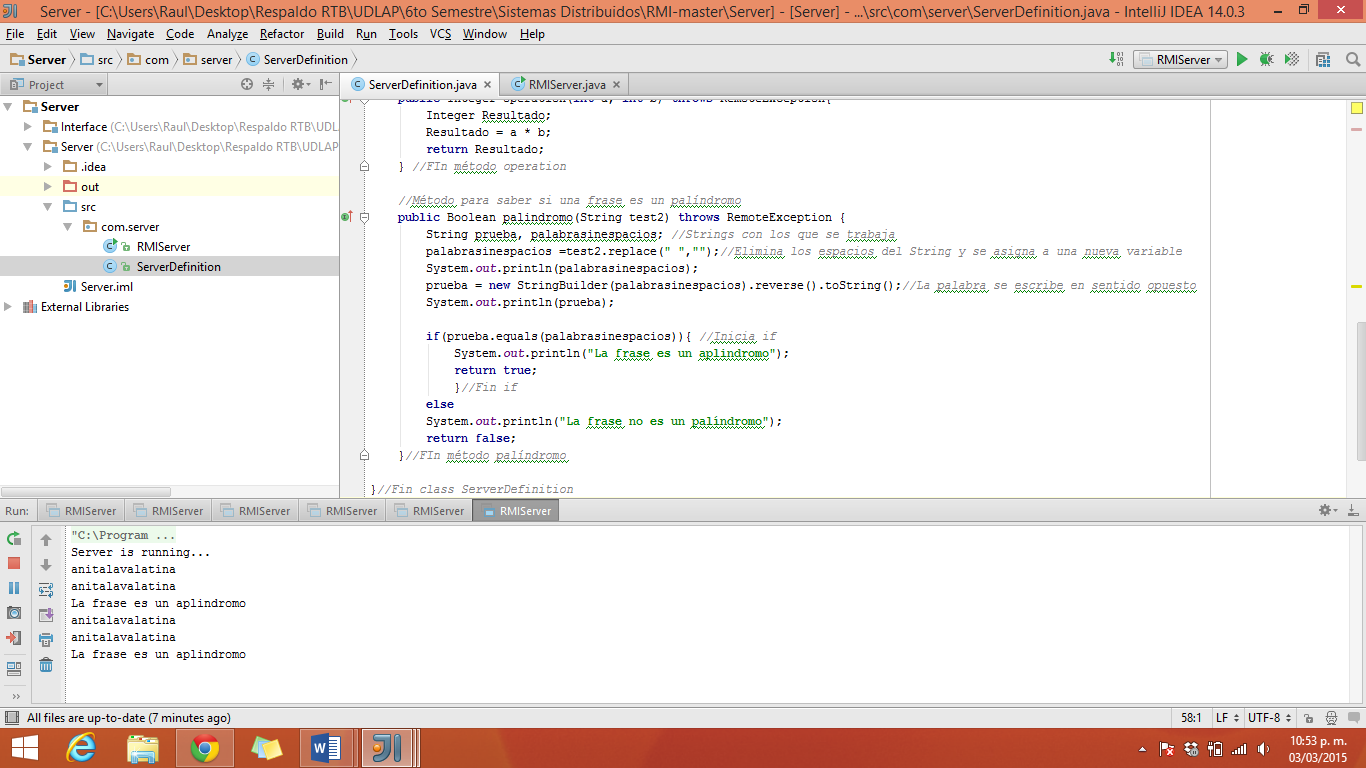
**Pasos a realizar:** Crear los siguientes métodos en la Interfaz **"TestRemote"**:

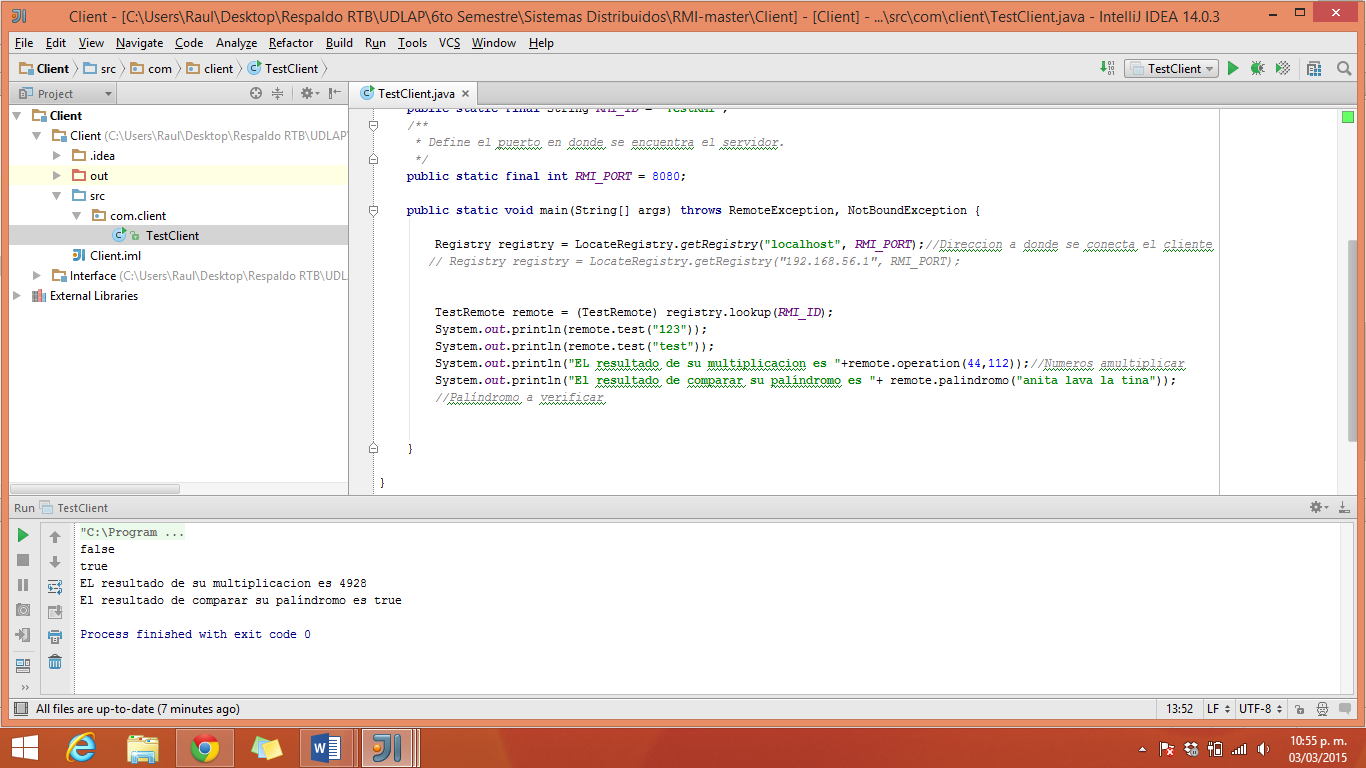
* El **primer** método debe recibir dos enteros, realizar una multiplicación y retornar el resultado.

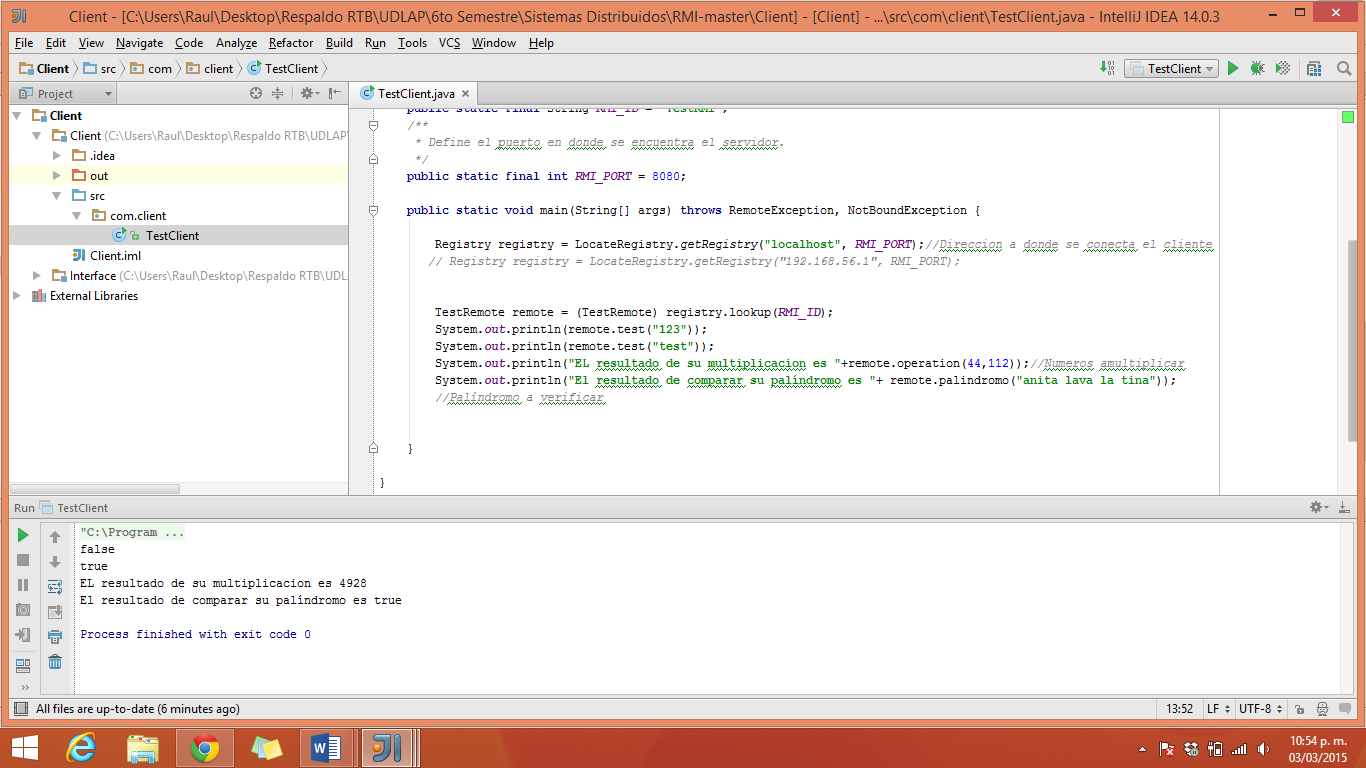




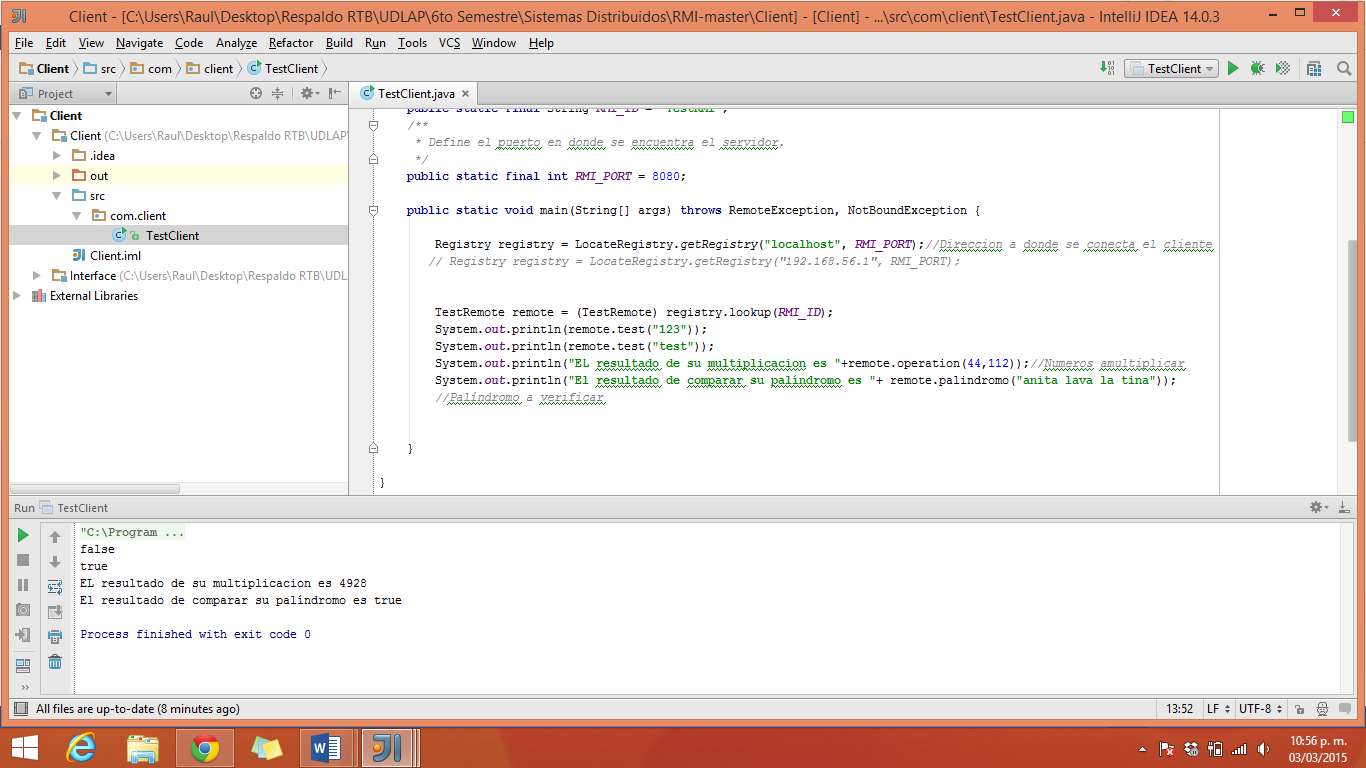
* El **segundo** método debe recibir un String y verificar si es un palíndromo, retornar true o false.







* Implementar los métodos en **Server**, en la clase **"ServerDefinition"**.
* Desarrollar el código necesario en **Client** para invocar remotamente estos métodos.
* Probar los métodos desde **Client** e imprimir el resultado en la **consola**.



Se crearon los métodos correspondientes, se muestra a continuación captura de pantalla, el cual nos muestra el código utilizado, al igual que el resultado esperado.

**Actividad 3 - Limpieza**

Ahora deberás limpiar el código, permitirá comprender de mejor manera la estructura del RMI.

* Identifica las clases **Client** y **Server**.
* Edita los comentarios con tu **nombre**, **correo** y **ID**.
* Agrega tus comentarios a las variables y métodos de cada **Interfaz/Clase**.

**R=** Esta parte de la actividad es mostrada con el código adjunto.

**Actividad 4 - Preguntas**

En el reporte ésta práctica, además de mostrar y explicar el desarrollo de las actividades deberás responder a las siguientes preguntas:

* ¿Cómo se definen nuevos métodos en **Server**?

**R=** Tienes que declararlo en la clase Interface para que pueda comunicarse con el cliente y ya dentro de Server se trabajará en el con el mismo nombre el método indicando cuales son las tareas que debe realizar.

* ¿Cuáles son los pasos necesarios para registrar un nuevo método?

**R=** Como se menciona anteriormente se declaran dentro del Servidor y la Interface para posteriormente ser llamado mediante el cliente.

* ¿Cuál es la función de **Registry**?

**R=** Elegir con que Servidor se va a conectar el cliente, podemos tratarlo como un local host o si se está trabajando dentro de la misma red, puedes conectarte con la computadora de algún otro compañero siempre y cuando los métodos se llamen de la misma manera y estén declarados en la Interface de ambos.

* ¿Qué se hace en **Interface**?

**R=** Nos sirve como base para trabajar ya que el cliente y el servidor extienden de esta clase, y si los métodos no se encuentran declarados en la misma, no hay forma en que el cliente logre saber cuáles son los métodos que serán llamados.

* ¿Por qué es necesaria **Interface**?

**R=** Para que se pueda conectar el cliente con el servidor y que ambos reconozcan los mismos métodos.

* ¿Cómo es la comunicación entre **Client** y **Server**?

**R=** El cliente llama al método del servidor, dándole un parámetro de entrada dependiendo de lo que se pida, es mandado al servidor, el cual lo procesa y posteriormente manda una respuesta al cliente.

* ¿Cómo podrías ejecutar más **Clientes**?

R= Primero tiene que haber un servidor en línea, posteriormente en Registry se debe indicar cuál será el servidor con el que se conectará dando la i.p del mismo y con esto será posible que nuestro servidor trabaja con varios clientes simultáneamente.

**Actividad 5 - Conclusiones**

Esta práctica resulto ser muy útil ya que entendimos de una mejor manera el funcionamiento de RMI, al igual que el manejo del lenguaje Java, resulto más fácil poder realizar las operaciones; también se buscó conectar con un cliente externo, pero en el intento no fue posible ya que la interface fue diferente.

Reporte práctica Java RMI II

**Introducción a la práctica**

Java RMI (Remote Method Invocation), es una tecnología desarrollada por Java para invocar métodos remotos. Proporciona un mecanismo simple para la comunicación de servidores y clientes en aplicaciones distribuidas basadas en Java. La construcción de una aplicación distribuida con RMI incluye:

1. Serialización (Marshalling) de parámetros y objetos en Java en el cliente para enviar al servidor.

2. Invocación desde el cliente, del Método Remoto (RMI) a ejecutar en el servidor.

3. Serialización (Marshalling) en el servidor para enviar respuesta al Cliente.

4. Recepción de la respuesta por parte del Cliente y ejecución local.

La segunda sesión de RMI consiste en comprender mejor el Marshalling (Serialización de objetos) con RMI, es muy importante haber concluido la primera actividad antes de continuar.

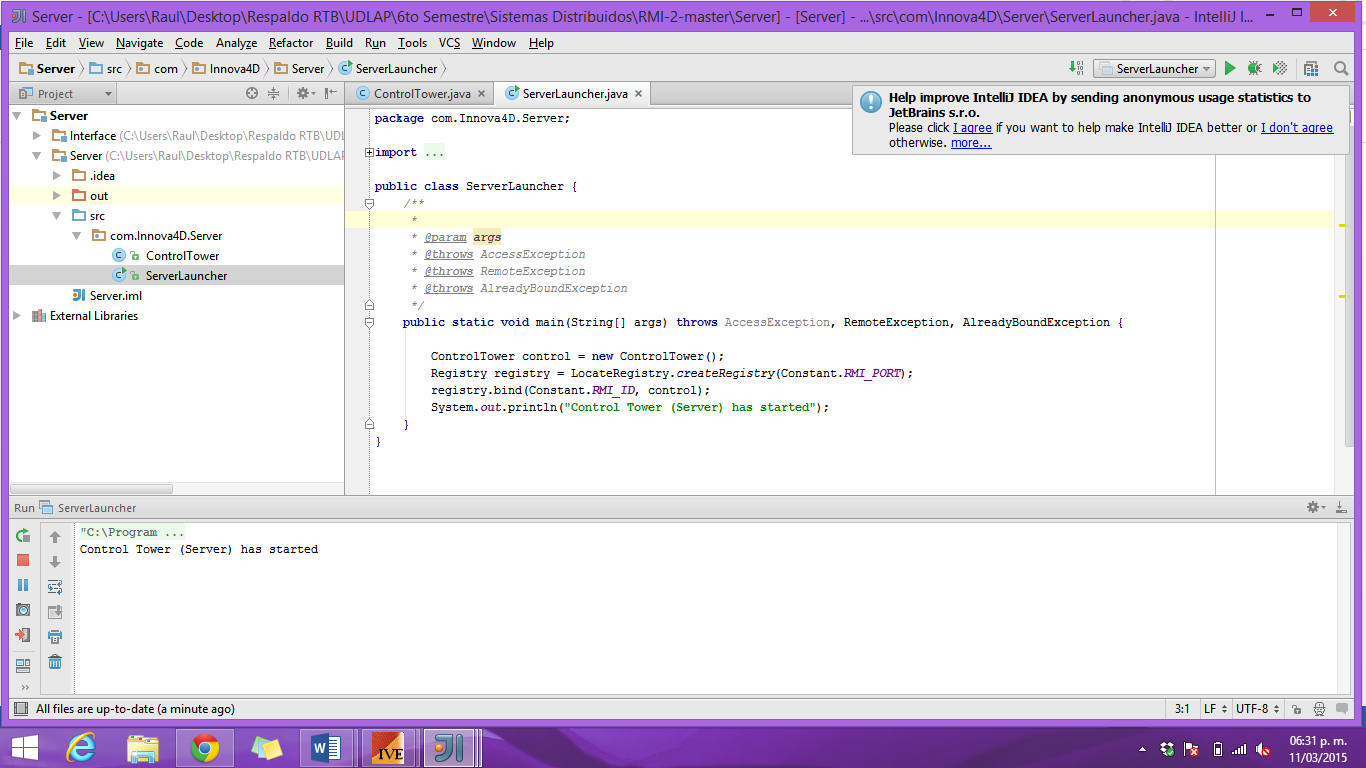
**Antes de comenzar:**

El primer paso en esta segunda práctica fue hacer lectura cuidadosa de la misma para poder entender cómo se va trabajar y posteriormente conectar los programas con los cuales se ejecuta la práctica.

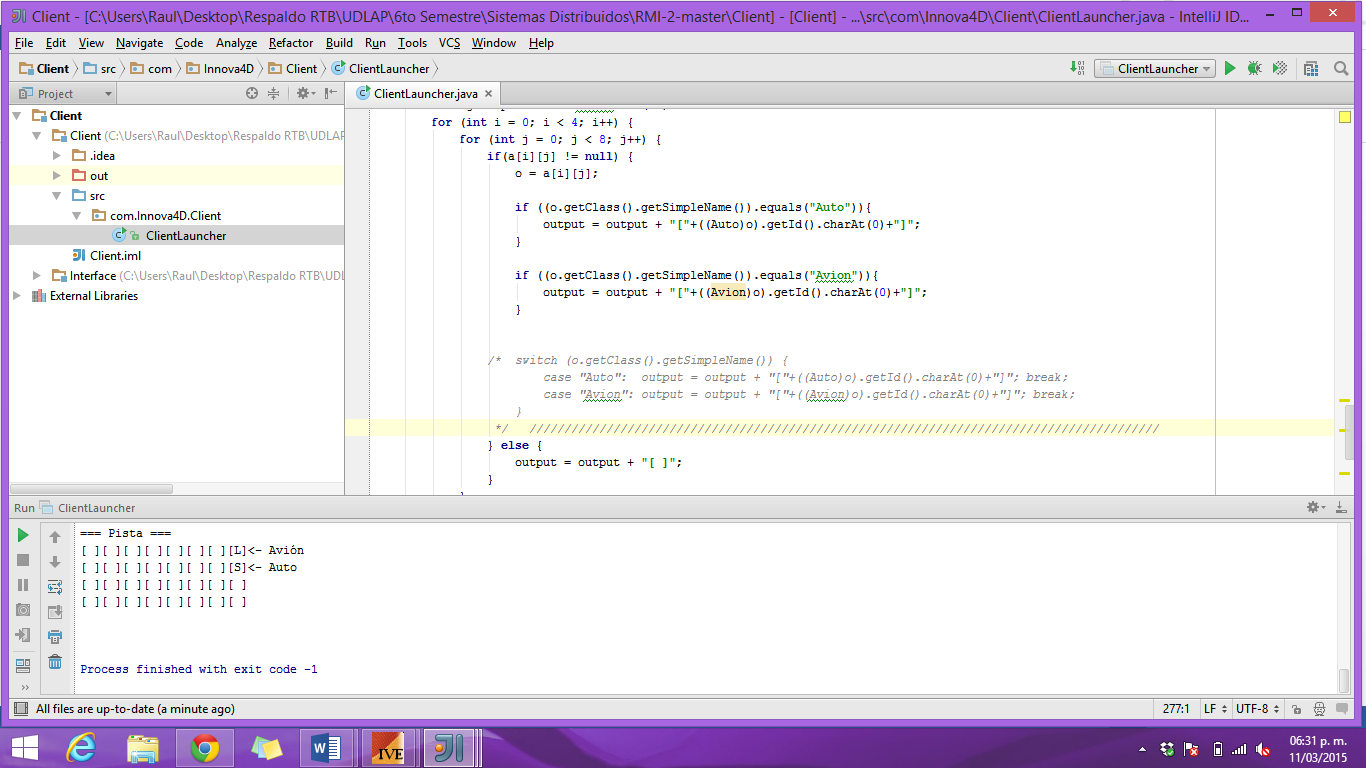
**Actividad 1: Análisis**

En **RMI** se definen **Client**, **Server** e **Interface**. La Interfaz define los métodos que deseamos implementar en Server. El **Cliente** solicita a **Server** los métodos que desea sean ejecutados a través de un registro que sólo debe realizarse una vez.

* Ejecutar **Server**: Click derecho sobre el proyecto, "**Run As** > **Java Application**".



* Ejecutar **Client**: Click derecho sobre el proyecto, "**Run As** > **Java Application**".



* Observar el comportamiento de la aplicación.

**R=** En un principio se debe poner en marcha el servidor, el cual viene siendo la Torre de control con la que se va a trabajar, posteriormente se inicia el cliente, el cual imprime una matriz de objetos 4 x 8. En un principio aparecen en los primeros puestos de la matriz una letra L, que representa al avión Lufthansa y una S que representa a nuestro auto, conforme pasan los segundos, las letras se mueven de izquierda a derecha indicando que hay un desplazamiento en nuestra “pista de aterrizaje”. En una etapa posterior de la práctica agregaremos otros dos elementos que trabajarán dentro de la misma matriz.

* Analizar la clase **RemoteInterface.java** y **ControlTower.java**.

**R=** RemoteInterface es nuestra Interface, en la cual se declaran todos los métodos con los que va a estar trabajando la aplicación, para que puedan ser ejecutados por el Cliente y el Servidor; mientras que ClontroTower es la clase donde se declaran los métodos mencionados en RemoteInterface, indicando que hace cada uno de ellos y cuidando que funcionen de manera correcta, aquí encontramos los métodos para mover los objetos dentro de la matriz.

* Analizar las clases **Avion.java** y **Auto.java**

**R=** Ambas clases sus casi idénticas entre sí, solo cambian algunos nombres de los métodos y valores con los que trabajas, son los objetos que posteriormente serán creados para poder trabajar con ellos y desplazarlos dentro de la matriz.

* Explicar en el **reporte** la funcionalidad de la interfaz y cómo se implementa en el servidor.

**R=** La interface y el servidor deben de tener los mismos métodos, ya que a través de ellos el cliente podrá comunicarse con el servidor, es decir es un intermediario entre ambos para indicar que se está trabajando con los mismos métodos y mantendrá la coherencia cuando tenga que regresar algún resultado.

* Explicar en el **reporte** el método **moverAvion(Avion a, int c)** que se encuentra en **ControlTower.java**

**R=** Es el método que va a mover el avión de izquierda a derecha dentro de la matriz, se basa en la posición del objeto avión; si el avión no se encuentra en la última posición de la matriz, nos da la posibilidad de moverlo una o más posiciones de izquierda a derecha hasta alcanzar su posición final.

* Analizar el Cliente **ClientLauncher.java**.

**R=** Es nuestra clase cliente, se va a conectar con el servidor, va a llamar a los métodos del servidor con los cuales trabajará y también es la clase que ejecuta los Threads de movimiento para cada uno de los objetos, dando un periodo de tiempo para mover los objetos.

* Explicar la funcionalidad del método **main**.

**R=** El método main se ubica en el cliente, va a llamar a la pista con la cual trabajamos y ejecuta cada uno de los métodos que crean los objetos que se van a mover dentro de la pista.

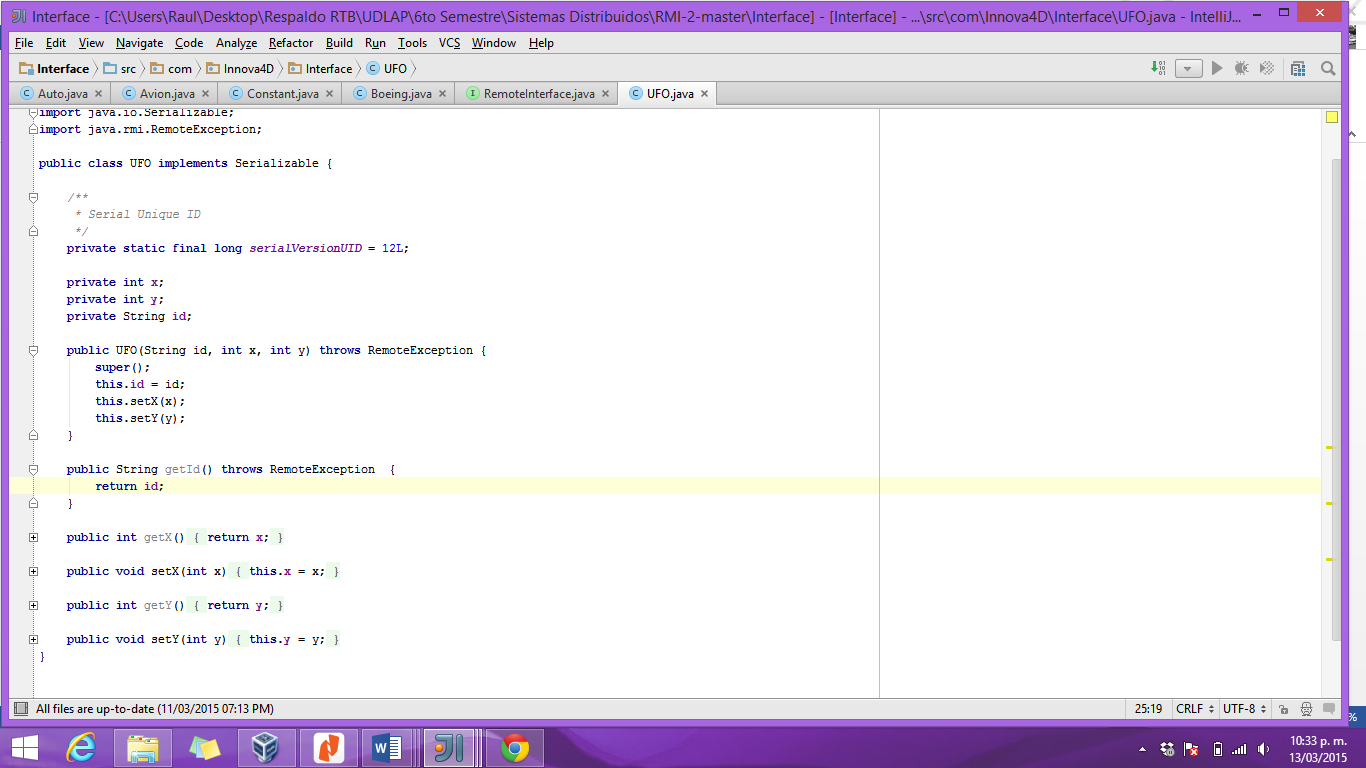
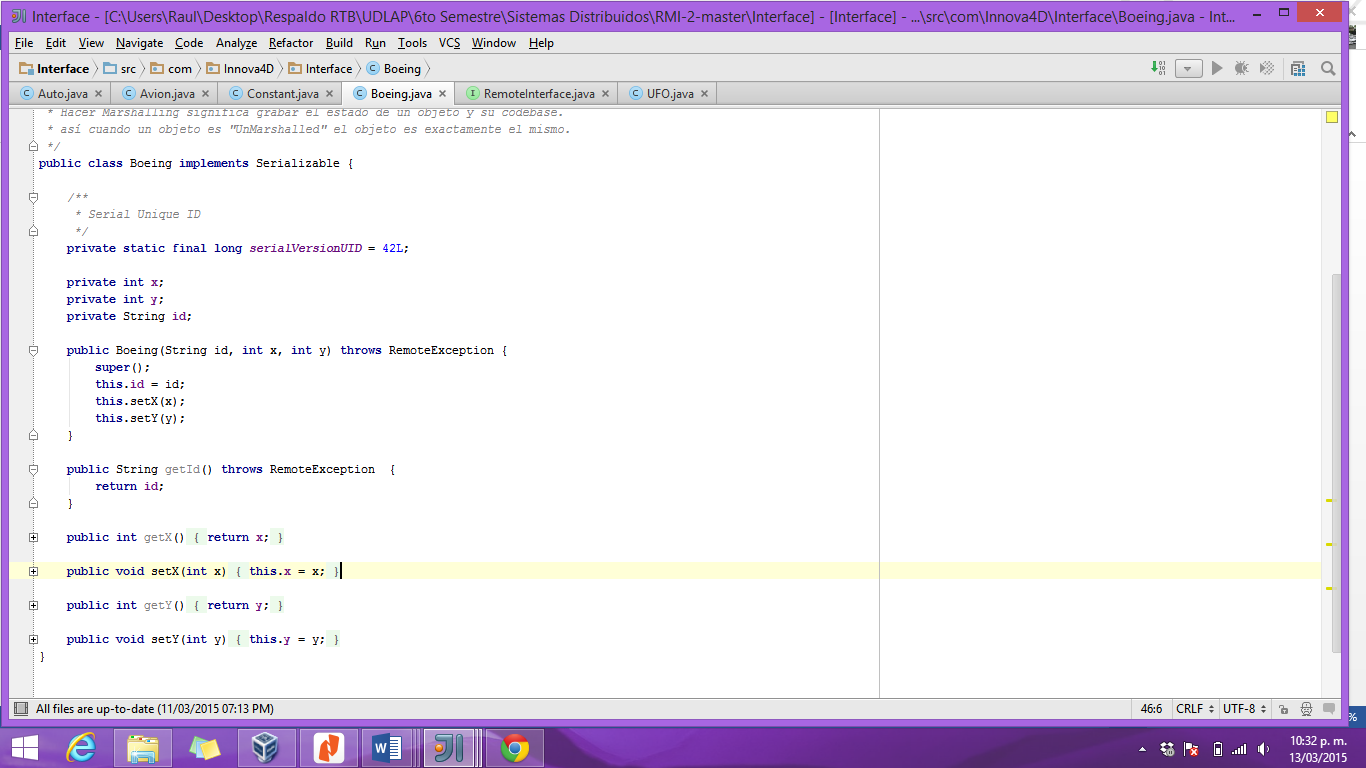
* Explicar los métodos **guiClient()** y **avionClient()**.

**R=** El método guiCliente lo que hace en un principio es conectarse con el servidor, posteriormente crea la pista con la cual se va a trabajar y el método aviónClient también se conecta con el servidor, crea el objeto que se va a desplazar en la matriz, le asigna una posición inicial y posteriormente corre el método Runnable el cual dará inicio al Thread que moverá nuestro objeto en un periodo de tiempo dado.

**Actividad 2 - Desarrollo**

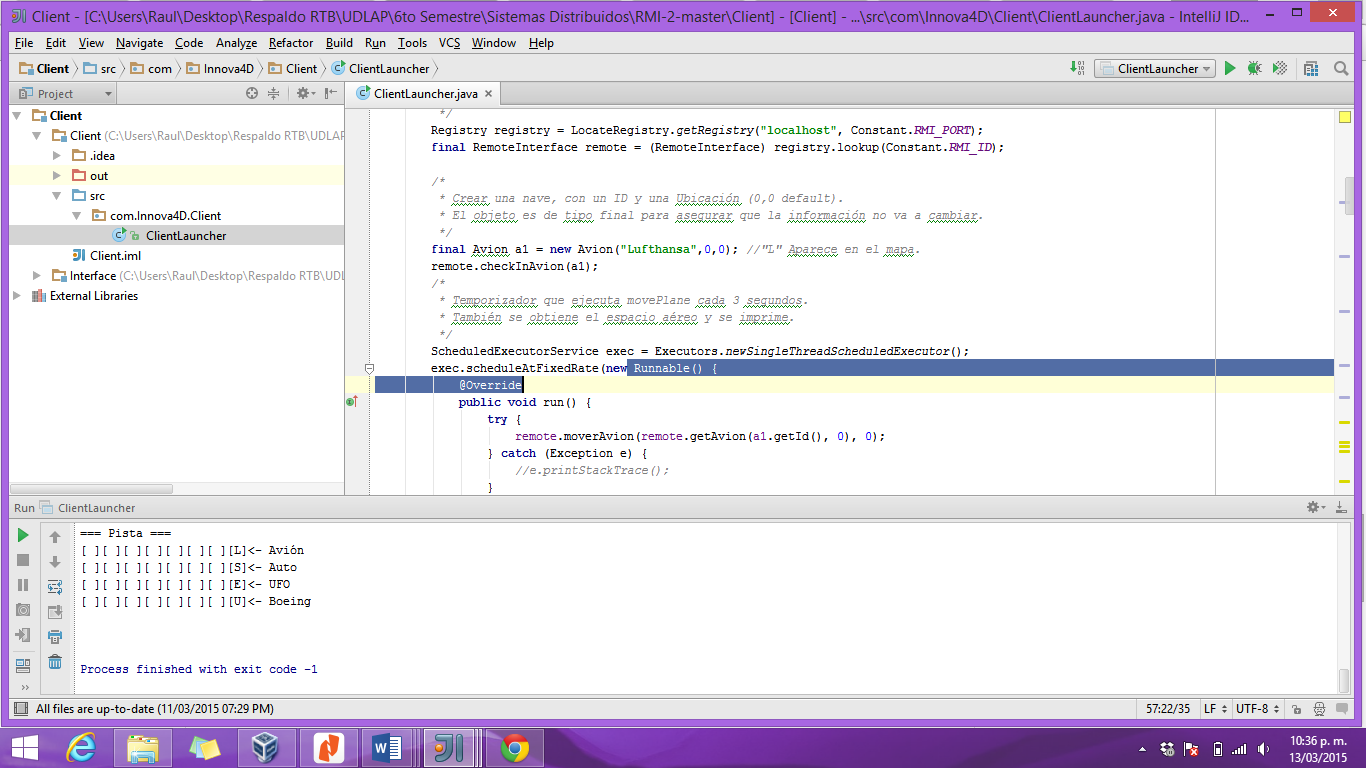
En ésta actividad deberás modificar el **Workspace** en Eclipse que se te ha entregado. Como habrás analizado las clases **Avion** y **Auto** son objetos que se agregan a una **matriz 4x8** que representa una pista aérea, los métodos en el **servidor** permiten informar el desplazamiento de estos vehículos en la **matriz**. Los objetos son creados en el **cliente** y enviados al **servidor** en donde se actualiza su ubicación. El objetivo de ésta actividad es **definir** e **implementar** la lógica de dos nuevos objetos en el **servidor**, que se crearán, enviarán y controlarán desde el **Cliente**.

* Implementar 2 nuevas clases de vehículos (e.g, UFO, Boeing), para agregarlos a la pista (Matriz 4x8).
* Implementar los **2 nuevos objetos** en los **2 últimos carriles** disponibles de la matriz.
* Implementar en el servidor la lógica necesaria para las nuevas clases.
* Implementar en el cliente los vehículos, deben ejecutarse en su propio hilo cada "x" segundos.
* Documentar las nuevas clases agregadas y los nuevos métodos implementados.
* Utilizar las guías de JavaDocs.
* Es importante analizar el código, revisar atentamente el diagrama de clases podría ser útil.



En la parte superior se muestran las nuevas clases creadas Boeing y UFO, los cuales son los nuevos objetos que se van a estar desplazando dentro de la matriz.

A continuación se muestra el resultado obtenido al agregar los nuevos métodos a la matriz, comienzan en los lugares asignados y se van desplazando de izquierda a derecha de acuerdo a la velocidad con la cual es ejecutado cada uno de los Threads correspondientes, se va ejecutando cada segundo la pista para mostrar uno a uno los cambios



# Actividad 3 - Limpieza

Ahora deberás limpiar el código, permitirá comprender de mejor manera la estructura del RMI.

* Identifica las clases **Client** y **Server**.
* Agrega la documentación a las clases con tu **nombre**, **correo** y **ID**.
* Agrega la documentación a las **variables** y **métodos** de cada **Interfaz/Clase**. Nota: La documentación debe ser siguiendo las guías de [Javadocs](http://en.wikipedia.org/wiki/Javadoc)

**R=** La documentación se encuentra adjunta con los programas trabajados durante las sesiones de laboratorio

# Actividad 4 - Preguntas

En el reporte de ésta práctica, además de mostrar y explicar el desarrollo de las actividades deberás responder a las siguientes preguntas:

* ¿Cómo se definen nuevas clases para usarlas con **Server**?

**R=** Tienen que crearse en la Interface en un principio para poder conectar con cliente y servidor y posteriormente definir en el Server los métodos que vamos a usar.

* ¿Cuáles son los pasos necesarios para crear una nueva clase y utilizar el objeto en el Servidor?

**R=** La clase es creada en la Interface, se declaran los métodos y atributos dentro de la misma clase, posteriormente se indica en el Servidor cuál es el método que lo usará y finalmente llamar a este objeto mediante el cliente.

* ¿Cuál es la utilidad de **Interface**?

**R=** Servir como intermediario entre el Cliente y el Servidor y que pueda entender que estamos trabajando “en el mismo idioma”, es decir, hacer Marshalling y Unmarshalling.

* ¿Cuál es la utilidad de **Constant.java**?

**R=** Es una clase que hace más eficiente nuestro proyecto, se declaran variables cuyos valores siempre serán los mismos, en este caso la conexión que hará con el cliente y el puerto que utilizará para esta conexión.

* ¿Cómo se manejan los errores en el Servidor?

**R=** Puede haber errores por no haber declarado el objeto correcto entre los métodos con los que trabaja el servidor, algún valor nulo o correr simultáneamente el servidor, en este caso nos manda una excepción porque el puerto elegido ya está siendo utilizado por este proceso.

* ¿Cuál es la utilidad de la clase **RemoteException**?

**R=** Previene las excepciones al momento de realizar una conexión con el servidor.

* ¿Por qué es necesario utilizar **Serializable** en las nuevas clases?

**R=** La Serialización de un objeto consiste en obtener una secuencia de bytes que represente el estado de dicho objeto. Esta secuencia puede utilizarse de varias maneras (puede enviarse a través de la red, guardarse en un fichero para su uso posterior, utilizarse para recomponer el objeto original, etc.). En este caso nos sirve para Marshalling y Unmarshalling, con lo cual nos aseguramos que el objeto con el que trabajamos sea siempre el mismo.

* Dada la comunicación entre **Client** y **Server**
  + ¿Por qué es necesario que el método avionClient() y los objetos **RemoteInterface** y **Avion** sean de tipo **final**?

**R=** Porque queremos que estos objetos no sean modificados por algún otro elemento de la aplicación, porque si no se puede llegar a presentar algún error durante la ejecución del mismo.

* ¿Por qué es necesario un método **getAvion(String id, int c)**?

**R=** Es necesario para saber en qué posición dentro de la matriz se encuentra nuestro objeto y posteriormente pueda ser movido por el método para mover el avión

* ¿Cómo se manejan los errores en el Cliente?

**R=** Puede haber excepciones por algún problema de conexión, no encuentre en el servidor, encimar objetos en la matriz, lo cual evita su correcto movimiento y finalmente correr dos veces seguidas el programa sin reiniciar el servidor, ya que si pasa esto duplica los objetos y se estorban entre sí.

* ¿Qué significa y cuál es la utilidad de **NotBoundException**?

**R=** Se genera en caso de que nuestro cliente tenga problemas para conectarse con el servidor o no encuentre el mismo.

**Actividad 5 – Conclusiones**

Al terminar la parte dos de la práctica de RMI hubo una mayor comprensión de los métodos utilizados y como trabajan; los ejercicios se realizaron de una mejor manera debido a que usamos Java, un lenguaje con el que estamos más familiarizados.

En general hubo un mayor entendimiento de los elementos con los que estamos trabajando, fue posible identificar Cliente, Servidor y la Interface y cada uno de los componentes propios de cada clase y finalmente se logró un mejor manejo de los Thread para tener varios hilos de ejecución en el programa y así poder variar la velocidad con la que se desplazaban los objetos dentro de la matriz.