Trabajo Práctico Especial 1: Tickets de Vuelo

31 de Agosto de 2022

Objetivo

Implementar en grupos un **sistema remoto *thread-safe*** para la **administración de asientos de vuelo** de una aerolínea, a partir de la existencia de uno o más modelos de avión y los tickets de vuelo, permitiendo notificar a los pasajeros de los eventos y ofreciendo reportes de los mapas de asientos hasta el momento.

Descripción Funcional

Servicios Remotos

El sistema requiere el desarrollo de **los siguientes servicios remotos**:

Servicio de Administración de Vuelos

- Funcionalidad: Administrar modelos de avión, vuelos y cambio de tickets de vuelos cancelados
- Usuario: Personal Administrativo de la Aerolínea
- Debe contar con métodos para:
    - **Agregar un modelo de avión** a partir de un nombre y la cantidad de filas y columnas de asientos para cada categoría de asiento
        i. Un modelo se identifica por su nombre. Si el modelo a agregar ya existe, se debe arrojar un error
        ii. Existen tres categorías de asientos: BUSINESS, PREMIUM_ECONOMY y ECONOMY
        iii. Se puede determinar que cada asiento del avión tendrá un número correspondiente a la fila y una letra correspondiente a la columna (donde la numeración de las filas es ascendente comenzando en 1 y la de las columnas es alfabética comenzando en A)
            1. La numeración de las filas es compartida por todas las categorías del avión donde la fila 1 corresponde a la categoría más alta (BUSINESS)
            2. De esta forma una fila de asientos tiene una sola categoría
            3. Dos filas de asientos de distinta categoría pueden tener una cantidad distinta de columnas
            4. Todos los aviones deben tener al menos una fila y una columna de asientos. Si para una categoría se indica una cantidad de filas y columnas negativa o cero, o la cantidad final de asientos del avión es cero, se debe arrojar un error
            5. No es necesario que un avión tenga al menos una fila de asientos de cada categoría
            6. Se garantiza que no se agregarán modelos de avión con más de 25 columnas de asientos

TPE 1: Tickets de Vuelo

- **Agregar un vuelo** a partir del nombre del modelo de avión, un código de vuelo, un código del aeropuerto destino y el detalle de los tickets de vuelo indicando para cada ticket el nombre del pasajero y la categoría del ticket (categoría de asiento comprada)
    i. Si el modelo no existe, se debe arrojar un error
    ii. Un vuelo se identifica por su código de vuelo. Si el vuelo a agregar ya existe, se debe arrojar un error
    iii. Cada vuelo podrá estar pendiente, cancelado o confirmado. Al agregar un vuelo, éste se inicia pendiente
    iv. Los pasajeros inician sin asientos asignados en el vuelo
    v. Se garantiza que los nombres de los pasajeros no se repiten en la totalidad de los vuelos del sistema
    vi. Un modelo de avión puede ser utilizado por múltiples vuelos
- **Consultar el estado de un vuelo** a partir del código de vuelo, indicando si está pendiente, cancelado o confirmado. Si no existe un vuelo con ese código se debe arrojar un error
- **Confirmar un vuelo pendiente** a partir del código de vuelo. Si no existe un vuelo con ese código o no tiene el estado pendiente arroja un error
- **Cancelar un vuelo pendiente** a partir del código de vuelo. Si no existe un vuelo con ese código o no tiene el estado pendiente arroja un error
- **Forzar el cambio de tickets de vuelos cancelados por tickets de vuelos alternativos**. Se considera vuelo alternativo a un vuelo pendiente que tenga el mismo aeropuerto destino que el del vuelo original
    i. Sólo se deben considerar los vuelos alternativos que tengan al menos un asiento libre de una categoría igual o inferior a la categoría del ticket de vuelo original del pasajero
    ii. En caso de existir más de un vuelo alternativo se prefieren los asientos de mejor categoría. Si hay disponibilidad en varios vuelos se elige asignar al vuelo menos completo. En caso de igualdad elegir el vuelo de menor código alfabético.
    iii. El orden de procesamiento es alfabético por código de vuelo y luego alfabético por nombre del pasajero
    iv. En caso de no poder encontrarle un vuelo alternativo a un pasajero, este mantiene su ticket original con el vuelo cancelado

Servicio de Asignación de Asientos

- Funcionalidad: Asignar asientos y ofrecer vuelos alternativos y cambios de ticket
- Usuario: Pasajeros de un vuelo
- Debe contar con métodos para:
    - **Consultar si un asiento está libre u ocupado** a partir del código de vuelo, el nombre del pasajero, el número de la fila y la letra de la columna del asiento
    - **Asignar un asiento libre a un pasajero** a partir del código de vuelo y el número de la fila y la letra de la columna del asiento
        i. El asiento será asignado al pasajero si el asiento está disponible y la categoría del mismo es igual o inferior a la categoría de su ticket para ese vuelo
        ii. Un asiento solo puede ser asignado si el vuelo está pendiente
        iii. Un pasajero no puede tener dos asientos en el vuelo

- **Mover a un pasajero de un asiento asignado a un asiento libre del mismo vuelo**, a partir del código de vuelo, el nombre del pasajero y el número de la fila y la letra de la columna del asiento libre
    i. En caso de que el pasajero tenga un asiento asignado para moverlo las condiciones para otorgar el asiento son las mismas que para asignar
- **Listar la disponibilidad en los vuelos alternativos de un pasajero**, a partir del código de vuelo original y el nombre del pasajero
    i. Para un pasajero con ticket en un vuelo no confirmado, listar la disponibilidad de asientos asignables en los vuelos alternativos
- **Cambiar el ticket de vuelo**, a partir del nombre del pasajero, el código de vuelo original y el código de vuelo alternativo
    i. En caso de que el pasajero tenga un ticket para un vuelo no confirmado, se lo mueve al vuelo solicitado siempre y cuando este sea un vuelo alternativo

- **Importante**: para todos los métodos cuando corresponda se debe arrojar un error
    - Si no existe un vuelo con ese código
    - Si no existe en el avión un asiento con los valores de la fila y columna provistos
    - En los casos particulares de cada método

Servicio de Notificaciones del Vuelo

- Funcionalidad: Notificar a los pasajeros de los eventos relacionados a sus tickets de vuelo al momento que ocurran
- Usuario: Pasajeros de un vuelo
- Debe contar con un método para:
    - **Registrar a un pasajero para que éste sea notificado de los eventos relacionados a un ticket de vuelo pendiente o cancelado** a partir del código identificador del vuelo y el nombre del pasajero
        i. Se considera un evento:
            1. **Registro exitoso**
            2. **Se confirmó el vuelo**
            3. **Se canceló el vuelo**
            4. **Se asignó un asiento en el vuelo**
            5. **Se movió de un asiento asignado a otro libre del mismo vuelo**
            6. **Se cambió el ticket por el de un vuelo alternativo**
        ii. En todos los eventos se debe indicar
            1. Código de vuelo
            2. Código de aeropuerto destino
            3. Categoría de asiento asignado (si es que tiene)
            4. Número de fila y letra de la columna del asiento asignado (si es que tiene)
        iii. En los dos últimos eventos se debe indicar además la categoría, número de fila y letra de la columna del asiento original (pudiendo no tener asiento en el sexto evento).
        iv. Se debe arrojar un error:
            1. Si no existe un vuelo con ese código
            2. Si el pasajero no tiene un ticket para ese vuelo
            3. Si el vuelo está confirmado
        v. Un pasajero puede registrarse múltiples veces para ser notificado de un mismo vuelo

Servicio de Consulta del Mapa de Asientos

- Funcionalidad: Consultar el mapa de asientos de un vuelo
- Usuario: Tripulación del vuelo
- Debe contar con métodos para:
    - **Consultar el mapa de asientos de un vuelo**, a partir del código del mismo, y uno de los siguientes criterios:
        i. Todos los asientos del vuelo
        ii. Los que tengan una categoría indicada
        iii. Los que pertenezcan a una fila indicada

      Para todos los mapas de asientos se debe indicar:
        i. Categoría de la fila de asientos
        ii. El número de la fila y la letra de la columna del asiento
        iii. Si el asiento está asignado, además indicar la primera letra del nombre del pasajero
    - Se debe arrojar un error si no existen asientos para el criterio indicado

Clientes

Para poder probar el sistema se requiere que se implementen cuatro programas cliente, cada uno coincidente con cada interfaz remota.

Cliente de Administración de Vuelos

La información de cuál es la acción a realizar se recibe a través de argumentos de línea de comando al llamar al *script* del cliente de administración de vuelos y el resultado se debe imprimir en pantalla.

Por ejemplo:

```
$> ./run-admin -DserverAddress=xx.xx.xx.xx:yyyy -Daction=actionName
[ -DinPath=filename | -Dflight=flightCode ]
```

donde

- **`run-admin`** es el *script* que invoca a la clase del cliente de administración de vuelos
- `xx.xx.xx.xx:yyyy` es la dirección IP y el puerto donde está publicado el servicio de administración de vuelos
- `actionName` es el nombre de la acción a realizar
    - **`models:`** Agrega una lote de modelos de aviones (ver detalle más abajo)
    - **`flights`**: Agrega un lote de vuelos (ver detalle más abajo)
    - **`status:`** Consulta el estado del vuelo de código `flightCode`. Deberá imprimir en pantalla el estado del vuelo luego de invocar a la acción o el error correspondiente
    - **`confirm:`** Confirma el vuelo de código `flightCode`. Deberá imprimir en pantalla el estado del vuelo luego de invocar a la acción o el error correspondiente
    - **`cancel:`** Cancela el vuelo de código `flightCode`. Deberá imprimir en pantalla el estado del vuelo luego de invocar a la acción o el error correspondiente
    - **`reticketing:`** Fuerza el cambio de tickets de vuelos cancelados por tickets de vuelos alternativos

TPE 1: Tickets de Vuelo

Para todas las acciones se deberá imprimir en pantalla el resultado de la operación o el error correspondiente.

De esta forma,

```
$> ./run-admin -DserverAddress=10.6.0.1:1099 -Daction=status
-Dflight=AA100
```

consulta el estado del vuelo `AA100` utilizando el servicio remoto publicado en `10.6.0.1:1099` e imprime en pantalla el mensaje correspondiente:

```
Flight AA100 is PENDING.
```

Acción `reticketing`

Se debe imprimir en pantalla la cantidad de tickets que fueron cambiados y los nombres de los pasajeros cuyos tickets no pudieron ser cambiados y los códigos de esos vuelos cancelados. Por ejemplo, para un *reticketing* de seis tickets de vuelo cancelados donde uno no pudo ser cambiado, una salida posible es la siguiente:

```
5 tickets where changed.
Cannot find alternative flight for John with Ticket AA100
```

Acción `models`

El cliente deberá leer de un archivo CSV un **lote de modelos de aviones** con los siguientes campos (delimitados por un ";"):

- **El nombre del modelo**
- **La cantidad de filas y columnas de asientos agrupadas por secciones** donde se indican una o mas secciones (delimitados por una ","):
    - Cada sección corresponde a una categoría de asientos y cuenta con los siguientes campos (delimitados por un "#"):
        - Categoría de asiento de la sección
        - Cantidad de filas de asientos de la sección
        - Cantidad de columnas de asientos de la sección

Cada línea del archivo representa un modelo de avión. El archivo contendrá una primera línea de encabezado.

Ejemplo de las primeras tres líneas de archivo:

```
Model;Seats
Boeing 787;BUSINESS#2#3,PREMIUM_ECONOMY#3#3,ECONOMY#20#10
Airbus A321;ECONOMY#15#9,PREMIUM_ECONOMY#3#6
```

- En la segunda línea se indica que el modelo de avión `Boeing 787` cuenta con tres secciones de asientos:
    - La sección de categoría `BUSINESS` tiene 2 filas y 3 columnas
    - La sección de categoría `PREMIUM_ECONOMY` tiene 3 filas y 3 columnas
    - La sección de categoría `ECONOMY` tiene 20 filas y 10 columnas

- Entonces el modelo tendrá asientos de la fila 1 a la 25, donde las filas 1 y 2 serán `BUSINESS`, la 3, 4 y 5 serán `PREMIUM_ECONOMY` y las demás filas serán `ECONOMY`. El modelo tendrá asientos de la columna A a la C inclusive en las categorías `BUSINESS` y `PREMIUM_ECONOMY` y de la columna A a la J inclusive en la categoría `ECONOMY`

El *path* del archivo necesario para cargar un lote de modelos de aviones, se recibe con `-DinPath=filename` y el resultado se debe imprimir en pantalla.

De esta forma,

```
$> ./run-admin -DserverAddress=10.6.0.1:1099 -Daction=models
-DinPath=../modelos.csv
```

realiza la carga de los modelos presentes en el archivo `modelos.csv` que se encuentra ubicado en el directorio superior a donde está el intérprete de comandos, utilizando el servicio remoto publicado en `10.6.0.1:1099` e imprime en pantalla un mensaje luego de haber realizado todas las solicitudes. Por ejemplo, para un archivo `modelos.csv` de 101 renglones se imprime:

```
100 models added.
```

En el caso de que la carga de lote de modelos de aviones ocasione un error, se deben imprimir el detalle de los modelos que no pudieron agregarse. Por ejemplo, para un archivo que contenga tres modelos donde uno de ellos cuenta con una cantidad de filas negativa en una de las secciones de asientos se imprime:

```
Cannot add model Boeing 777.
2 models added.
```

Acción `flights`

El cliente deberá leer de un archivo CSV un **lote de vuelos** con los siguientes campos (delimitados por un “;”)**:**

- **El nombre del modelo de avión del vuelo**
- **El código del vuelo**
- **El código del aeropuerto destino del vuelo**
- **Uno o más tickets del vuelo** (delimitados por una “,”):
  - Cada ticket del vuelo cuenta con los siguientes campos (delimitados por un “#”):
    - Categoría de asiento del ticket
    - Pasajero del ticket

Cada línea del archivo representa un vuelo. El archivo contendrá una primera línea de encabezado.

Ejemplo de las primeras tres líneas de archivo:

```
Model;FlightCode;DestinyAirport;Tickets
Boeing 787;AA100;JFK;BUSINESS#John,ECONOMY#Juliet,BUSINESS#Elizabeth
Airbus A321;AA101;JFK;ECONOMY#Alfred
```

- En la segunda línea se indica que el vuelo de código `AA100` con destino a `JFK` utilizará un avión `Boeing 787` donde se vendieron tres tickets:
  - John y Elizabeth tienen un ticket con categoría `BUSINESS`
  - Juliet tiene un ticket con categoría `ECONOMY`

El *path* del archivo necesario para cargar un lote de modelos de aviones, se recibe con `-DinPath=filename` y el resultado se debe imprimir en pantalla.

De esta forma,

```
$> ./run-admin -DserverAddress=10.6.0.1:1099 -Daction=flights
-DinPath=../vuelos.csv
```

realiza la carga de los vuelos presentes en el archivo `vuelos.csv` que se encuentra ubicado en el directorio superior a donde está el intérprete de comandos, utilizando el servicio remoto publicado en `10.6.0.1:1099` e imprime en pantalla un mensaje luego de haber realizado todas las solicitudes. Por ejemplo, para un archivo `vuelos.csv` con 101 renglones se imprime:

```
100 flights added.
```

En el caso de que la carga de lote de vuelos ocasione un error, se deben imprimir el detalle de los vuelos que no pudieron agregarse. Por ejemplo, para un archivo que contenga un vuelo con un modelo de avión inexistente se imprime:

```
Cannot add flight AA954.
2 flights added.
```

Cliente de Asignación de Asientos

La información de cuál es la acción a realizar se recibe a través de argumentos de línea de comando al llamar al *script* del cliente de asignación de asientos y el resultado se debe imprimir en pantalla.

Por ejemplo:

```
$> ./run-seatAssign -DserverAddress=xx.xx.xx.xx:yyyy -Daction=actionName
-Dflight=flightCode [ -Dpassenger=name | -Drow=num | -Dcol=L |
-DoriginalFlight=originFlightCode ]
```

donde

- **`run-seatAssign`** es el *script* que invoca al cliente de asignación de asientos
- `xx.xx.xx.xx:yyyy` es la dirección IP y el puerto donde está publicado el servicio de asignación de asientos
- `actionName` es el nombre de la acción a realizar
  - **`status:`** Deberá imprimir en pantalla si el asiento de fila `num` y columna `L` del vuelo de código `flightCode` está libre u ocupado luego de invocar a la acción
  - **`assign:`** Asigna al pasajero `name` al asiento libre de fila `num` y columna `L` del vuelo de código `flightCode`.

- **`move`:** Mueve al pasajero `name` de un asiento asignado en el vuelo de código `flightCode` a un asiento libre del mismo vuelo, ubicado en la fila `num` y columna `L`.
- **`alternatives`:** Listar los vuelos alternativos al vuelo de código `flightCode` para el pasajero `name`. Para cada categoría de asiento en cada vuelo alternativo se debe listar
  - El código del aeropuerto destino
  - El código del vuelo
  - La cantidad de asientos asignables de la categoría
  - La categoría de los asientos asignables
- **`changeTicket`**: Cambia el ticket del pasajero `name` de un vuelo de código `originFlightCode` a otro vuelo alternativo de código `flightCode`

Para todas las acciones se deberá imprimir en pantalla el resultado de la operación o el error correspondiente.

De esta forma,

```
$> ./run-seatAssign -DserverAddress=10.6.0.1:1099 -Daction=status
-Dflight=AA100 -Drow=2 -Dcol=C
```

imprime en pantalla si el asiento `2C` del vuelo `AA100` está libre u ocupado. Las posibles salidas son:

```
Seat 2C is FREE.
```

o

```
Seat 2C is ASSIGNED to John.
```

Acción `alternatives`

El cliente deberá listar la salida de forma ordenada, primero por la categoría (antes las mejores categorías), luego descendente por cantidad de asientos libres y por último alfabético por código de vuelo. Por ejemplo, un listado de vuelos alternativos para un pasajero con un ticket de vuelo cancelado con categoría `BUSINESS`:

```
JFK | AA101 | 7 BUSINESS
JFK | AA119 | 3 BUSINESS
JFK | AA103 | 18 PREMIUM_ECONOMY
JFK | BC103 | 18 PREMIUM_ECONOMY
JFK | AA119 | 15 PREMIUM_ECONOMY
```

Cliente de Notificaciones del Vuelo

El código del vuelo y el nombre del pasajero se recibe a través de argumentos de línea de comando al llamar al *script* del cliente de notificaciones del vuelo y el resultado se debe imprimir en pantalla.

Por ejemplo:

```
$> ./run-notifcations -DserverAddress=xx.xx.xx.xx:yyyy
-Dflight=flightCode -Dpassenger=name
```

donde

- **run-notifcations** es el *script* que invoca a la clase del cliente de notificaciones del vuelo
- `xx.xx.xx.xx:yyyy` es la dirección IP y el puerto donde está publicado el servicio de notificaciones del vuelo
- `flightCode`: el código del vuelo
- `name`: el nombre del pasajero

Se deberá imprimir en pantalla el resultado de la operación o el error correspondiente.
De esta forma,

```
$>  ./run-notifcations -DserverAddress=10.6.0.1:1099 -Dflight=AA100
-Dpassenger=John
```

registra al pasajero `John` para recibir notificaciones del vuelo `AA100` utilizando el servicio remoto publicado en `10.6.0.1:1099` y cada vez que se produce un evento relacionado al vuelo en cuestión se imprime un mensaje.
Ejemplos de los eventos posibles:

You are following Flight AA100 with destination JFK.

Your seat is BUSINESS 1B for Flight AA100 with destination JFK.

Your seat changed to BUSINESS 2C from BUSINESS 1B for Flight AA100 with destination JFK.

Your Flight AA100 with destination JFK was cancelled and your seat is BUSINESS 2C.

Your ticket changed to Flight AA101 with destination JFK from Flight AA100 with destination JFK.

Your Flight AA100 with destination JFK was confirmed and your seat is PREMIUM_ECONOMY 15D.

El cliente de seguimiento del vuelo deberá finalizar su ejecución:

- En caso de que falle la registración de un pasajero
- Luego de que no se puedan producir más eventos para ese vuelo

Cliente de Consulta del Mapa de Asientos

Deberá dejar en archivos CSV los resultados de las consultas realizadas. Las consultas posibles son las siguientes:

Consulta 1: Asientos del vuelo

Cada línea de la salida debe contener los siguientes campos:

- Para cada asiento
  - el número de fila del asiento
  - la letra de la columna del asiento
  - un * si el asiento está libre o la primera letra del nombre del pasajero si el asiento está asignado
- Para cada fila de asientos
  - La categoría de la fila de asientos

Se debe respetar el orden del mapa de asientos, listando primero la fila 1 y la columna A.

❏ Salida de ejemplo:

```
| 01 A * | 01 B * | 01 C * |   BUSINESS
| 02 A * | 02 B J | 02 C * |   BUSINESS
| 03 A * | 03 B * | 03 C * |   PREMIUM_ECONOMY
| 04 A * | 04 B A | 04 C * |   PREMIUM_ECONOMY
| 05 A * | 05 B * | 05 C * |   PREMIUM_ECONOMY
| 06 A * | 06 B * | 06 C * | 06 D * | 06 E * |  ECONOMY
| 07 A * | 07 B * | 07 C * | 07 D * | 07 E * |  ECONOMY
```

Consulta 2: Asientos del vuelo que tengan una categoría indicada

Respetando lo enunciado en la Consulta 1, se deben listar únicamente las filas de asientos de una categoría donde la categoría se recibe por parámetro.

❏ Salida de ejemplo para la categoría `PREMIUM_ECONOMY`:

```
| 03 A * | 03 B * | 03 C * |   PREMIUM_ECONOMY
| 04 A * | 04 B A | 04 C * |   PREMIUM_ECONOMY
| 05 A * | 05 B * | 05 C * |   PREMIUM_ECONOMY
```

Consulta 3: Asientos del vuelo que tengan a una fila indicada

Respetando lo enunciado en la Consulta 1, se debe listar únicamente una fila de asientos donde el número de la fila se recibe por parámetro.

❏ Salida de ejemplo para la fila 2:

```
| 02 A * | 02 B J | 02 C * |   BUSINESS
```

La información de cuál es la consulta a realizar se recibe a través de argumentos de línea de comando al llamar al *script* del cliente de consulta del mapa de asientos y el resultado se debe escribir en un archivo.

Por ejemplo:

```
$> ./run-seatMap -DserverAddress=xx.xx.xx.xx:yyyy -Dflight=flightCode [
-Dcategory=catName | -Drow=rowNumber ] -DoutPath=output.csv
```

donde

- **`run-seatMap`** es el *script* que invoca a la clase del cliente de consulta del mapa de asientos
- `xx.xx.xx.xx:yyyy` es la dirección IP y el puerto donde está publicado el servicio de consulta del mapa de asientos
- Si no se indica `-Dcategory` ni `-Drow` se resuelve la **Consulta 1**
- Si se indica `-Dcategory`, `catName` es el nombre de la categoría de asiento elegida para resolver la **Consulta 2**
- Si se indica `-Drow`, `rowNumber` es el número de la fila de asientos elegida para resolver la **Consulta 3**
- Si no se indica `-Dflight` la consulta falla
- Si se indican ambos `-Dcategory` y `-Drow` la consulta falla
- `output.csv` es el *path* del archivo de salida con los resultados de la consulta elegida

De esta forma,

```
$> ./run-seatMap -DserverAddress=10.6.0.1:1099 -Dflight=AA123
-DoutPath=query1.csv
```

realiza la **Consulta 1** de los asientos del vuelo de código `AA123` utilizando el servicio remoto publicado en `10.6.0.1:1099` y deja en `query1.csv` los resultados obtenidos según el formato arriba mencionado.

La siguiente invocación

```
$> ./run-seatMap -DserverAddress=10.6.0.1:1099 -Dflight=AA123
-DoutPath=../query2.csv -Dcategory=BUSINESS
```

realiza la **Consulta 2** de los asientos de categoría `BUSINESS` del vuelo de código `AA123` utilizando el servicio remoto publicado en `10.6.0.1:1099` y deja en `query2.csv` (ubicado en el directorio superior a donde está el intérprete de comandos) los resultados obtenidos según el formato arriba mencionado.

En caso de que ocurra un error o que no exista ninguna fila de asientos que cumpla con el criterio de la consulta, el cliente de consulta deberá imprimir en pantalla un mensaje y no generar el archivo de salida.

Hechos y Consideraciones

Para simplificar el desarrollo se pueden tomar los siguientes considerandos como ciertos:

- Se asume que el formato y contenido de los archivos de entrada de lote de modelos y lote de vuelos es correcto y no es necesario validarlo
- No es necesario que tenga persistencia. Al reiniciar el sistema se comienza de cero la operación del mismo

Requisitos

TPE 1: Tickets de Vuelo

Se requiere implementar:

- Todos los servicios deben ser implementados utilizando RMI y `UnicastRemoteObject`, teniendo en cuenta que los servicios deben poder atender pedidos de clientes de manera concurrente y responder a lo indicado en este enunciado
- Los clientes indicados cada uno como una aplicación diferente

Muy Importante:

➔ **Respetar exactamente los nombres de los *scripts*, los nombres de los archivos de salida y el orden y formato de los parámetros del *scripts***

➔ En todos los `pom.xml` que entreguen deberán definir el `artifactId` de acuerdo a la siguiente convención: "`tpe1-gX-Z`" donde X es el número de grupo y Z es `parent`, `api`, `server` o `client`. Por ejemplo: **`<artifactId>`**`tpe1-g7-api`**`</artifactId>`**

➔ En todos los `pom.xml` que entreguen deberán incluir el tag `name` con la siguiente convención: "`tpe1-gX-Z`" donde X es el número de grupo y Z es `parent`, `api`, `server` o `client`. Por ejemplo: **`<name>`**`tpe1-g7-api`**`</name>`**

➔ La implementación debe **respetar exactamente el formato de salida enunciado**

Material a entregar

Cada grupo deberá subir al **Campus ITBA** un archivo compactado conteniendo:

- El **código fuente** de la aplicación:
  - Utilizando el arquetipo de Maven utilizado en las clases
  - Con una correcta separación de las clases en los módulos *api*, *client* y *server*
  - Un README indicando cómo preparar el entorno a partir del código fuente para correr el servidor y los cuatro clientes
  - El directorio oculto `.git/` donde se encuentra la historia de commits y modificaciones
  - **No se deben entregar los binarios**
- Un **documento breve** (no más de dos carillas) explicando:
  - Decisiones de diseño e implementación de los servicios
  - Criterios aplicados para el trabajo concurrente
  - Potenciales puntos de mejora y/o expansión

Corrección

El trabajo no se considerará aprobado si:

- No se entregó el trabajo práctico en tiempo y forma
- No se entrega alguno de los materiales solicitados en la sección anterior
- El código no compila utilizando Maven en consola (de acuerdo a lo especificado en el README a entregar)

- El servicio no inicia cuando se siguen los pasos del README
- Los clientes no corren al seguir los pasos del README

Si nada de esto se cumple, se procederá a la corrección donde se tomará en cuenta:

- Que los servicios y clientes funcionen correctamente según las especificaciones dadas
- El resultado de las pruebas y lo discutido en el coloquio
- La aplicación de los temas vistos en clase: Java 8, Concurrencia y RMI
- La modularización, diseño testeo y reutilización de código
- El contenido y desarrollo del informe

Uso de Git

Es obligatorio el uso de un repositorio Git para la resolución de este TPE. Deberán crear un repositorio donde todos los integrantes del grupo colaboren con la implementación. No se aceptarán entregas que utilicen un repositorio git con un único *commit* que consista en la totalidad del código a entregar.

Los distintos *commits* deben permitir ver la evolución del trabajo, tanto grupal como individual.

Muy importante: **los repositorios creados deben ser privados, solo visibles para los integrantes del grupo y la cátedra en caso de que se lo solicite específicamente.**

Cronograma

- **Presentación del Enunciado: miércoles 31/08**
- **Entrega del trabajo: Estará disponible hasta el jueves 15/09 a las 23:59** la actividad "Entrega TPE 1" localizada en la sección Contenido / Evaluación / TPE 1. En la misma deberán cargar el paquete con el **código fuente** y el **documento**
- **El día miércoles 28/09 a las 18:00** cada grupo tendrá un espacio de 10 minutos para un coloquio. Durante el mismo se les hará una devolución del trabajo, indicando la nota y los principales errores cometidos. A criterio de la cátedra también se podrán realizar preguntas sobre la implementación. Opcionalmente se les podrá solicitar la ejecución de la aplicación a la cátedra para revisar el funcionamiento de alguna funcionalidad. Para ello es necesario que un integrante del equipo tenga el sistema remoto “levantado”.
- **El día del recuperatorio será el miércoles 16/11**
- **No se aceptarán entregas pasado el día y horario establecido como límite**

Dudas sobre el TPE

Las mismas deben volcarse en los **Debates** del Campus ITBA.

Recomendaciones

- **Clases y métodos útiles para consultar**

- `java.nio.file.Files.readAllLines`
- `java.nio.file.Files.write`