Trabajo Práctico Especial 1
Sala de Emergencias

11 de Septiembre de 2024

Objetivo

Implementar en grupos un **sistema remoto *thread-safe*** para la **atención de emergencias de un hospital** permitiendo notificar al personal médico y ofreciendo reportes de las emergencias atendidas hasta el momento.

Introducción

En una **sala de emergencias** de un hospital, a medida que llegan los pacientes, se los categoriza en un **nivel de emergencia** mediante el proceso de *triage*.

Los miembros del personal médico, al iniciar su jornada laboral, se registran como disponibles para atender un nivel de emergencia máximo, pudiendo atender a los pacientes que tengan una emergencia de ese nivel o de uno menor.

Para la atención de los pacientes la sala de emergencia puede incorporar **consultorios** a medida que se necesiten. El encargado de la sala de emergencia decidirá que un paciente será atendido por un médico en un consultorio determinado.

Al finalizar la atención de la emergencia el médico deberá indicarlo en el sistema, dejándolo disponible para atender otra, salvo que indique que se tomará un receso.

Servicios Remotos y Clientes

El sistema requiere el desarrollo de **los siguientes servicios remotos con sus respectivos clientes**.

Es importante destacar que, aunque en este enunciado se mencionan servicios y clientes de manera conjunta por motivos de claridad y brevedad, **la implementación debe respetar la correcta separación entre servicio y cliente**. Para ello deben analizar si la lógica a implementar corresponde que esté del lado del servicio, del cliente o de ambas. Aunque en este trabajo práctico especial deberán implementar un único cliente de línea de comandos para cada uno de los servicios presentados considerar que un mismo servicio podría ser invocado por varios clientes implementados en diversas tecnologías (por ejemplo un cliente de línea de comandos y un cliente de una aplicación móvil).

1. Servicio de Administración

- Funcionalidad: Agregar consultorios, agregar médicos y administrar la disponibilidad de los médicos
- Usuario: Encargado de la Sala de Emergencia
- Cliente: 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 **`administrationClient.sh`** y el resultado se debe imprimir en pantalla.

```
$> sh administrationClient.sh -DserverAddress=xx.xx.xx.xx:yyyy
-Daction=actionName [ -Ddoctor=doctorName | -Dlevel=levelNumber |
-Davailability=availabilityName ]
```

donde

- `xx.xx.xx.xx:yyyy` es la dirección IP y el puerto donde está publicado el servicio de administración
- `actionName` es el nombre de la acción a realizar (que se detallan en las siguientes secciones)

1.1 Agregar un consultorio

- Funcionalidad: Agrega un consultorio. El consultorio inicia libre. La numeración de los consultorios es incremental donde el primero tiene el número 1.
- No falla
- Ejemplo de cliente de invocación: Agregar el consultorio “1”

```
$> sh administrationClient.sh -DserverAddress=10.6.0.1:50051
-Daction=addRoom
        Room #1 added successfully
```

1.2 Agregar un médico

- Funcionalidad: Agrega un médico a partir del nombre y el nivel máximo de emergencias que puede atender. Los médicos se agregan con disponibilidad “*Unavailable*” o no disponible.
- Falla si:

    - Ya existe un médico con ese nombre
    - El nivel es inválido
- Ejemplo de cliente de invocación: Agregar el médico “John” con un nivel “3”.

```
$> sh administrationClient.sh -DserverAddress=10.6.0.1:50051
-Daction=addDoctor -Ddoctor=John -Dlevel=3
Doctor John (3) added successfully
```

1.3 Definir la disponibilidad de un médico

- Funcionalidad: Definir si un médico está disponible (“Available”) o no disponible (“Unavailable”) a partir del nombre.
- Falla si:
    - No existe un médico con ese nombre
    - El médico está atendiendo a un paciente, esto es, la disponibilidad actual del médico es atendiendo (“Attending”)
- Ejemplo de cliente de invocación: Definir la disponibilidad del médico “John” como disponible

```
$> sh administrationClient.sh -DserverAddress=10.6.0.1:50051
-Daction=setDoctor -Ddoctor=John -Davailability=available
Doctor John (3) is Available
```

Para definir la disponibilidad del médico “John” como no disponible

```
$> sh administrationClient.sh -DserverAddress=10.6.0.1:50051
-Daction=setDoctor -Ddoctor=John -Davailability=unavailable
Doctor John (3) is Unavailable
```

1.4 Consultar la disponibilidad de un médico

- Funcionalidad: Consultar si un médico está disponible (“Available”), no disponible (“Unavailable”) o atendiendo (“Attending”) a partir del nombre.
- Falla si:
    - No existe un médico con ese nombre
- Ejemplo de cliente de invocación: Consultar la disponibilidad del médico “John” que está atendiendo

```
$> sh administrationClient.sh -DserverAddress=10.6.0.1:50051
-Daction=checkDoctor -Ddoctor=John
Doctor John (3) is Attending
```

2. Servicio de Sala de Espera

- Funcionalidad: Registrar pacientes que ingresan a la sala, actualizar el nivel de sus emergencias y consultar la espera aproximada de los pacientes para ser atendidos.
- Usuario: Encargado de la Sala de Espera
- Cliente: 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 sala de espera **`waitingRoomClient.sh`** y el resultado se debe imprimir en pantalla.

```
$> sh waitingRoomClient.sh -DserverAddress=xx.xx.xx.xx:yyyy
-Daction=actionName [ -Dpatient=patientName | -Dlevel=levelNumber ]
```

donde

- xx.xx.xx.xx:yyyy es la dirección IP y el puerto donde está publicado el servicio de sala de espera
- actionName es el nombre de la acción a realizar (que se detallan en las siguientes secciones)

2.1 Registrar un paciente

- Funcionalidad: Registrar un paciente que ingresa en la sala a partir del nombre y el nivel de emergencia que tiene
- Falla si:
  - Ya existe un paciente con ese nombre esperando en la sala
  - Ya existe un paciente con ese nombre que está siendo atendido o que ya fue atendido
- Ejemplo de cliente de invocación: Ingresa el paciente “Foo” con una emergencia de nivel “3”

```
$> sh waitingRoomClient.sh -DserverAddress=10.6.0.1:50051
-Daction=addPatient -Dpatient=Foo -Dlevel=3
        Patient Foo (3) is in the waiting room
```

2.2 Actualizar el nivel de emergencia de un paciente

- Funcionalidad: Actualizar el nivel de emergencia de un paciente que está esperando en la sala a partir del nombre y el nuevo nivel de emergencia que tiene
- Falla si:
  - No existe un paciente con ese nombre esperando en la sala
  - El nivel es inválido
- Ejemplo de cliente de invocación: Actualiza el nivel de emergencia del paciente “Foo” que está esperando a ser atendido a un nivel “4”

```
$> sh waitingRoomClient.sh -DserverAddress=10.6.0.1:50051
-Daction=updateLevel -Dpatient=Foo -Dlevel=4
        Patient Foo (4) is in the waiting room
```

2.3 Consultar la espera aproximada de un paciente

- Funcionalidad: Consultar, para un paciente que está esperando en la sala, cuántos serán atendidos antes que él, asumiendo que existen las salas y los médicos necesarios, a partir del nombre.
- Falla si:
  - No existe un paciente con ese nombre esperando en la sala
- Ejemplo de cliente de invocación: Consultar la espera aproximada del paciente “Foo” indicando que hay 7 pacientes que serán atendidos antes que él.

```
$> sh waitingRoomClient.sh -DserverAddress=10.6.0.1:50051
-Daction=checkPatient -Dpatient=Foo
```

```
Patient Foo (4) is in the waiting room with 7 patients ahead
```

3. Servicio de Atención de Emergencias

- Funcionalidad: Iniciar y finalizar la atención de emergencias en consultorios
- Usuario: Encargado de la Sala de Emergencia
- Cliente: 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 atención de emergencias **`emergencyCareClient.sh`** y el resultado se debe imprimir en pantalla.

```
$> sh emergencyCareClient.sh -DserverAddress=xx.xx.xx.xx:yyyy
-Daction=actionName [ -Droom=roomNumber | -Ddoctor=doctorName |
-Dpatient=patientName ]
```

donde

- `xx.xx.xx.xx:yyyy` es la dirección IP y el puerto donde está publicado el servicio de atención de emergencias
- `actionName` es el nombre de la acción a realizar (que se detallan en las siguientes secciones)

3.1 Iniciar la atención de una emergencia en un consultorio

- Funcionalidad: Iniciar la atención de una emergencia en un consultorio libre a partir del número del consultorio. El consultorio libre ahora estará ocupado, un médico disponible (“Available”) ahora tendrá una disponibilidad atendiendo (“Attending”) y un paciente que estaba esperando en la sala ahora estará siendo atendido hasta que se indique la finalización de la atención de la emergencia con el método **3.3 Finalizar la atención de una emergencia.**
  - **Algoritmo determinístico:**
    - **i.** Se elige primero el **paciente** que está esperando en la sala, por orden descendente del nivel de emergencia (se deben atender a los de mayor nivel primero) y a igual nivel por orden de llegada (se deben atender primero a los que llegaron antes).
    - **ii.** Para ese paciente se elige ahora el **médico** que lo atenderá, por orden ascendente del máximo nivel de emergencia (mejor que una emergencia de nivel 2 sea atendida por un médico de nivel 2 que por uno de nivel 5) y a igual nivel por orden alfabético.
    - **iii.** En caso de conseguir un médico que pueda atender la emergencia del paciente se inicia la atención. De lo contrario se repite el proceso con el resto de los pacientes que están esperando en la sala hasta lograr iniciar la atención de una emergencia.
- Falla si:
  - No existe un consultorio con ese número
  - El consultorio existe pero no se encuentra libre
  - Habiendo intentado el algoritmo con todos los pacientes no se puede iniciar la atención de una emergencia (por ejemplo si no hay médicos disponibles, o si los que están disponibles tienen un nivel máximo menor a los niveles de emergencia de los pacientes que están esperando en la sala).

- Ejemplo de cliente de invocación: Iniciar la atención de una emergencia en el consultorio "2" indicando que el paciente "Foo" será atendido por el médico "John".

```
$> sh emergencyCareClient.sh -DserverAddress=10.6.0.1:50051
-Daction=carePatient -Droom=2
    Patient Foo (4) and Doctor John (3) are now in Room #2
```

En caso de que no se pueda iniciar la atención de una emergencia se muestra el siguiente mensaje:

```
$> sh emergencyCareClient.sh -DserverAddress=10.6.0.1:50051
-Daction=carePatient -Droom=2
    Room #2 remains Free
```

3.2 Iniciar la atención de emergencias en los consultorios libres

- Funcionalidad: Iniciar la atención de todas las emergencias posibles utilizando todos los consultorios libres. En función de los pacientes que están esperando y de la disponibilidad de los médicos los consultorios libres ahora estarán ocupados, los médicos disponibles ("Available") ahora tendrán una disponibilidad atendiendo ("Attending") y los pacientes que estaban esperando en la sala ahora estarán siendo atendidos hasta que se indique la finalización de la atención de cada una de las emergencias con múltiples invocaciones al método **3.3 Finalizar la atención de una emergencia.**
  - **Algoritmo determinístico:**
    - **i.** Se elige primero el **consultorio** libre, por orden ascendente por número del consultorio
    - **ii.** Para ese consultorio se aplica el algoritmo determinístico de **3.1 Iniciar la atención de una emergencia en un consultorio**
    - **iii.** Se repite el proceso para todos los consultorios libres
- No falla
- Ejemplo de cliente de invocación: Atender emergencias en los consultorios "1" y "3" indicando los nombres de los pacientes que serán atendidos y los médicos que atenderán esas emergencias.

```
$> sh emergencyCareClient.sh -DserverAddress=10.6.0.1:50051
-Daction=careAllPatients
    Patient Bar (5) and Doctor Paul (5) are now in Room #1
    Room #2 remains Occupied
    Patient Foo (1) and Doctor Melanie (4) are now in Room #3
    Room #4 remains Free
```

3.3 Finalizar la atención de una emergencia en un consultorio

- Funcionalidad: El médico informa la finalización de una emergencia que estaba atendiendo a partir del número del consultorio, el nombre del médico y el nombre del paciente. El consultorio ahora estará libre y el médico estará disponible ("Available").
- Falla si:
  - No existe un médico con ese nombre
  - El médico no estaba atendiendo una emergencia en ese consultorio

- Ejemplo de cliente de invocación: El médico “John” informa la finalización de la atención de la emergencia del paciente “Foo” en el consultorio “2”.

```
$> sh emergencyCareClient.sh -DserverAddress=10.6.0.1:50051
-Daction=dischargePatient -Droom=2 -Ddoctor=John -Dpatient=Foo
    Patient Foo (4) has been discharged from Doctor John (4) and the Room
#2 is now Free
```

4. Servicio de Notificación al Personal

- Funcionalidad: Registrar a los médicos para que sean notificados de los eventos relacionados a las emergencias, anular ese registro y consultar el historial de eventos que sucedieron
- Usuario: Médicos de la Sala de Emergencias
- Cliente: 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 notificación al personal **`doctorPagerClient.sh`** y el resultado se debe imprimir en pantalla.

```
$> sh doctorPagerClient.sh -DserverAddress=xx.xx.xx.xx:yyyy
-Daction=actionName -Ddoctor=doctorName
```

donde

- `xx.xx.xx.xx:yyyy` es la dirección IP y el puerto donde está publicado el servicio de notificación al personal
- `actionName` es el nombre de la acción a realizar (que se detallan en las siguientes secciones)

4.1 Registrar a un médico para ser notificado

- Funcionalidad: Registrar a un médico a partir de su nombre para que sea notificado de los eventos relacionados a las emergencias que debe atender y las finalizaciones de las mismas. **Se considera un evento:**
    - Se **registró correctamente** para recibir notificaciones

```
Doctor John (4) registered successfully for pager
```

    - Definió su **disponibilidad** (a partir de 1.3 Definir la disponibilidad de un médico)

```
Doctor John (4) is Available
```

    - **Inició la atención** de una emergencia (a partir de 3.1 Iniciar la atención de una emergencia en un consultorio o 3.2 Iniciar la atención de emergencias en los consultorios libres)

```
Patient Foo (4) and Doctor John (4) are now in Room #2
```

    - **Finalizó la atención** de una emergencia (a partir de 3.3)

```
Patient Foo (4) has been discharged from Doctor John (4) and the Room #2
is now Free
```

- Falla si:
  - No existe un médico con ese nombre
  - El médico ya se registró para recibir notificaciones
- Ejemplo de cliente de invocación: Registrar al médico “John” para recibir notificaciones, mostrando el evento inicial de registro, donde el cliente no termina y muestra las notificaciones a medida que van ocurriendo

```
$> sh doctorPagerClient.sh -DserverAddress=10.6.0.1:50051
-Daction=register -Ddoctor=John
    Doctor John (4) registered successfully for pager
    ...
```

4.2 Anular el registro de un médico

- Funcionalidad: Anular el registro de un médico para no recibir más notificaciones. Además finaliza la ejecución del cliente de notificación al personal.
- Falla si:
  - El médico no se registró para recibir notificaciones
- Ejemplo de invocación: Anular el registro del médico “John” para no recibir más notificaciones mostrando el evento final tanto en el cliente que invocó a 4.1 Registrar a un médico para ser notificado (que además finaliza) como en este cliente

```
t2$> sh doctorPagerClient.sh -DserverAddress=10.6.0.1:50051
-Daction=unregister -Ddoctor=John
      Doctor John (4) unregistered successfully for pager
t2$>

t1$> sh doctorPagerClient.sh -DserverAddress=10.6.0.1:50051
-Daction=register -Ddoctor=John
      Doctor John (4) registered successfully for pager
      ...
      Doctor John (4) unregistered successfully for pager
t1$>
```

4.3 Consultar el historial de notificaciones (***Exclusivo para el grupo de cuatro integrantes***)

- Funcionalidad: Listar todos los eventos (desde el inicio del servicio) que sucedieron hasta el momento, correspondientes a un médico.
- Falla si:
  - No existe un médico con ese nombre
- Ejemplo de cliente de invocación: Consultar el historial de notificaciones del médico “John”.

```
$> sh doctorPagerClient.sh -DserverAddress=10.6.0.1:50051
-Daction=history -Ddoctor=John
Doctor John (4) registered successfully for pager
Doctor John (4) unregistered successfully for pager
```

```
      Doctor John (4) registered successfully for pager
      Patient Foo (4) and Doctor John (4) are now in Room #2
      Patient Foo (4) has been discharged from Doctor John (4) and the Room #2
is now Free
      Doctor John (4) is Unavailable
      ...
```

5. Servicio de Consulta

- Funcionalidad: Consultar el estado actual de los consultorios, el listado de pacientes que están esperando en la sala y el historial de atenciones de emergencias finalizadas.
- Usuario: Área de Auditoría del Hospital
- Cliente: 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 consulta **`queryClient.sh`** y el resultado se debe imprimir en pantalla.

```
$> sh queryClient.sh -DserverAddress=xx.xx.xx.xx:yyyy -Daction=actionName
-DoutPath=filePath.csv [ -Droom=roomNumber ]
```

donde

- `xx.xx.xx.xx:yyyy` es la dirección IP y el puerto donde está publicado el servicio de consulta
- `actionName` es el nombre de la ***query*** a realizar (que se detallan en las siguientes secciones)
- `filePath.csv` es el *path* del archivo de salida (absoluto o relativo) con los resultados de la *query* elegida en formato CSV (separado por comas)

5.1 Estado actual de los consultorios

- Funcionalidad: Consultar el estado actual de los consultorios, en orden ascendente por número, indicando para cada uno su número, si está libre u ocupado, y en caso de estar ocupado el paciente y el médico correspondiente.
- Falla si:
  - No se agregó al menos un consultorio y en ese caso el archivo de salida no debe crearse
- Ejemplo de invocación: Consultar los consultorios indicando por ejemplo que el consultorio "1" está libre y el consultorio "2" está ocupado por el médico "John" atendiendo al paciente "Foo".

```
$> sh queryClient.sh -DserverAddress=10.6.0.1:50051 -Daction=queryRooms
-DoutPath=../query1.csv
$> cat ../query1.csv
   Room,Status,Patient,Doctor
   1,Free,,
   2,Occupied,Foo (2),John (4)
   3,Occupied,Bar (3),Paul (3)
   ...
```

5.2 Pacientes esperando a ser atendidos

- Funcionalidad: Consultar el listado de pacientes que están esperando en la sala, en orden descendente por nivel de emergencia y a igual nivel por orden de llegada, indicando para cada uno el nombre y el nivel de emergencia
- Falla si:
    - No hay ningún paciente esperando en la sala de espera y en ese caso el archivo de salida no debe crearse
- Ejemplo de invocación: Consultar el listado de pacientes indicando que el paciente “Abc” tiene un nivel de emergencia 4.

```
$> sh queryClient.sh -DserverAddress=10.6.0.1:50051
-Daction=queryWaitingRoom -DoutPath=query2.csv
$> cat query2.csv
   Patient,Level
   Abc,4
   Zzz,2
   ...
```

5.3 Atenciones finalizadas

- Funcionalidad: Consultar el listado de atenciones de emergencias que fueron iniciadas y luego finalizadas, en orden en que se resolvieron, indicando para cada una el número del consultorio utilizado, el paciente que fue atendido y el médico que la resolvió. Se debe ofrecer también la posibilidad de filtrar el resultado listando únicamente las atenciones que involucran un consultorio con un parámetro opcional.
- Falla si:
    - No se resolvió ninguna emergencia y en ese caso el archivo de salida no debe crearse
- Ejemplo de invocación: Consultar el listado de atenciones donde primero finalizó la atención del paciente “Foo” por el médico “John” en el consultorio “2” y luego finalizó la atención del paciente “Bar” por el médico “Paul” en el consultorio “1”.

```
$> sh queryClient.sh -DserverAddress=10.6.0.1:50051 -Daction=queryCares
-DoutPath=/tmp/query3.csv
$> cat /tmp/query3.csv
   Room,Patient,Doctor
   2,Foo (2),John (4)
   1,Bar (3),Paul (3)
   2,Abc (4),John (4)
```

Para filtrar el resultado y sólo listar las emergencias resueltas en el consultorio “2”:

```
$> sh queryClient.sh -DserverAddress=10.6.0.1:50051 -Daction=queryCares
-DoutPath=/tmp/query3.csv -Droom=2
$> cat /tmp/query3.csv
   Room,Patient,Doctor
   2,Foo (2),John (4)
   2,Abc (4),John (4)
```

Hechos y Consideraciones

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

- No es necesario que tenga persistencia. Al reiniciar el sistema se comienza de cero la operación del mismo
- Los valores de los parámetros de los clientes no contienen espacios
- Si el archivo de salida existe, reemplazar el contenido (no *appendear*)
- Los únicos valores posibles para los niveles de emergencia son: 1, 2, 3, 4 y 5 donde 5 es el mayor nivel de emergencia

Requisitos

Se requiere implementar:

- Todos los servicios utilizando gRPC, 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 de consola diferente

Muy Importante:

➔ **Respetar EXACTAMENTE**
- **Los nombres de los *scripts* y sus parámetros** (Si se pide `-Droom` no se aceptará `-DroomNumber`, `-Dconsultorio`, etc.)
- **El orden y formato de salida enunciado, tanto para los clientes de consola como para los archivos CSV de salida**

➔ **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>`**
- El `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>`**

Material a entregar

Cada grupo deberá subir al **Campus UNICAMENTE 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 cinco clientes

    - El directorio oculto `.git/` donde se encuentra la historia de commits y modificaciones. Recordar que la descarga desde github.com no incluye el directorio .git/.
    - **No se deben entregar los binarios.** Recordar de ejecutar el comando `mvn clean` antes de la entrega.
- Un **documento breve** (no más de cuatro 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 la 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: Concurrencia y gRPC
- 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 11/09**
- **Entrega del trabajo: Estará disponible hasta el jueves 26/09 a las 23:59** la actividad "TPE 1" localizada en la sección Contenido / Evaluaciones. En la misma deberán cargar el archivo compactado.
- **El día del coloquio será el miércoles 09/10**.
- **El día del recuperatorio será el miércoles 27/11**
- **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.lang.System#getProperty(java.lang.String)`
  - `java.nio.file.Files.write`

Índice