Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2025-2

Publicación: 16 de octubre de 2025
Actividad evaluada

Actividad 5
Threading

Entrega

- **Lugar:** Repositorio personal de GitHub — Carpeta: `Actividades/AC5`
- **Fecha máxima de entrega:** 16 de octubre 17:20
- **Ejecución de actividad**: La Actividad será ejecutada **únicamente** desde la terminal del computador. **Los *paths* relativos utilizados en la Actividad deben ser coherentes con esta instrucción, y no pueden modificarse.**

Importante: Antes de comenzar, comprueba que Git este funcionando correctamente en tu repositorio privado. Para esto, **sube los archivos base de la actividad de inmediato** (*add*, *commit*, *push*). Se espera que en esta actividad (así como en las demás actividades y tareas) utilices Git a lo largo de **todo tu desarrollo** como una herramienta, no sólo como un método de entrega. Es por esto que recomendamos enfáticamente que vayas subiendo tus cambios constantemente (***push***), ya que **problemas de último minuto** relacionados con la entrega y Git **no serán considerados**.

Objetivo de la actividad

- Aplicar conocimientos de *Threading* para permitir la concurrencia de código.
- Aplicar conceptos de *Lock* para controlar el acceso a código crítico.
- Utilizar señales y el método `join` para permitir la interacción entre *threads* cuando corresponda.
- Probar código mediante la ejecución de *test* y el archivo `main.py`.

Introducción

Gracias a la diversidad de canciones que aparecen en las ayudantías, discotecas –como la *BlonDCC*– se han vuelto famosas entre los estudiantes del DCC que cada vez disfrutan más de escuchar música y bailar.

Los ayudantes de Programación Avanzada, viendo una oportunidad de negocio, deciden crear su propia discoteca –*DCClub*–, por lo que deberás hacer un programa para simular el funcionamiento de este establecimiento.

Flujo del programa

El programa consiste un club al que pueden asistir múltiples clientes. El programa deberá encargarse de preparar todos lo necesario para que el *DCClub* pueda abrir y manejar la llegada y salida de los distintos clientes que quieran acceder al establecimiento.

Debido a las restricciones establecidas durante la pandemia, los clubes y discotecas presentan capacidades máximas de cuantos clientes pueden atender a la vez. Por lo que el programa deberá asegurarse de que contador capacidad no se vea afectado por problemas de concurrencia.

Esta actividad consta en completar 2 clases utilizando los contenidos de *Threading*. La primera clase modela el *DCClub*, mientras que la segunda modela los clientes. Ambas clases serán corregidas exclusivamente mediante el uso de *tests*.

Finalmente, debes asegurarte de entregar, como mínimo, el archivo que tenga el *tag* de Entregar en la siguiente sección. Los demás archivos no es necesario subir, pero tampoco se penalizará si se suben al repositorio personal.

Archivos y entidades

- Entregar Modificar `main.py`: Archivo principal a ejecutar, se encarga de llamar a todas las clases a implementar. Contiene la definición de las clases `DCClub` y `Cliente`.

  La clase `DCClub` corresponde a un *thread* que representa el establecimiento. Presenta los siguientes atributos:

  - `self.capacidad_actual` Entero que indica la capacidad actual del establecimiento.
  - `self.capacidad_maxima` Entero que indica la capacidad máxima del establecimiento.
  - `self.senal_abierto` `Event` que representa señal de apertura del establecimiento.
  - `self.lock_capacidad` `Lock` que asegura que solo 1 cliente pueda entrar o salir del establecimiento a la vez. Solo el `DCClub` tiene acceso a este *lock*.
  - `self.artista` Texto que indica el artista que se presenta esta noche en el establecimiento.
  - `self.instrumentos` Lista de textos que indica los instrumentos que deben ser preparados para la presentación de esta noche.

  La clase `Cliente` corresponde a un *thread* que representa a los clientes que asisten al establecimiento. Presenta los siguientes atributos:

  - `self.daemon` Booleano que determina si el *thread* es *daemon*.
  - `self.senal_abrio_club` `Event` que representa señal de apertura del *DCClub*.
  - `self.entrar_al_club` Método de `DCClub` que le permite al cliente entrar al *DCClub*.
  - `self.salir_del_club` Método de `DCClub` que le permite al cliente salir del *DCClub*.
  - `self.id` Identificador único del cliente.

  Tanto `DCClub` como `Cliente` poseen distintos atributos de clase que presentan constantes que serán utilizadas por cada una de ellas. Los métodos de cada una de estas clases serán explicados con mayor detalle en las distintas partes del enunciado.

Parte I. DCClub

Para hacer que `DCClub` sea funcional, deberás completar, modificar y utilizar los siguientes métodos.

Hint: Para algunos de los métodos de esta clase necesitaras el siguiente conocimiento:

> El método `acquire()` de la clase `Lock` recibe como parámetro el booleano *blocking*, que le indica si debe quedarse esperando a que se libere el *lock* (`True`) o si debe intentarlo solo una vez y seguir su ejecución (`False`).
>
> El método `acquire()` de la clase `Lock` retorna `True` si el *lock* fue adquirido correctamente y `False` en caso contrario.
>
> Para más información sobre `Lock` y `acquire`, puedes revisar la documentación de `Lock`.

- No modificar `def __init__(self, capacidad: int, artista: str, instrumentos: list) -> None:`

  Inicializador de la clase. Asigna los argumentos recibidos en los atributos que le corresponden. Adicionalmente, define otros atributos necesarios para el *thread*.

- No modificar `def revisar_mesa_sonido(self) -> bool:`

  Método encargado de preparar la mesa de sonido. Durante la preparación de esta mesa pueden suceder eventos imprevistos, por lo que el tiempo que demora en prepararse es incierto. Una vez que termina de preparar la mesa de sonido retorna `True`.

- No modificar `def revisar_instrumentos(self) -> bool:`

  Método encargado de revisar los instrumentos necesarios para la presentación de la noche. El tiempo de revisión de cada instrumento es variable, por lo que el tiempo de demora en realizarse toda la revisión es incierto. Una vez que termina de revisar todos los instrumentos retorna `True`.

- Modificar `def preparar_escenario(self) -> bool:`

  Método encargado de preparar el escenario para que pueda abrir el establecimiento. Revisa **al mismo tiempo** la mesa de sonido y los instrumentos necesarios para realizar la presentación. Una vez que termina de preparar la mesa de sonido y los instrumentos retorna `True`.

- No modificar `def llegada_cliente(self, cliente: Cliente) -> None:`

  Método encargado de aumentar el contador de capacidad e imprimir el mensaje correspondiente a la llegada de un cliente.

- No modificar `def salida_cliente(self, cliente: Cliente) -> None:`

  Método encargado de disminuir el contador de capacidad e imprimir el mensaje correspondiente a la llegada de un cliente.

- Modificar `def manejar_llegada_cliente(self, cliente: Cliente) -> bool:`

  Método encargado de manejar la llegada de un cliente. Primeramente intenta adquirir el `lock_capacidad`, pero **NO** debe quedarse esperando a que se libere el *lock* si no logra obtenerlo.

En caso de obtener el *lock* y que no se haya alcanzado la capacidad máxima del establecimiento, debe llamar al método `llegada_cliente`, liberar el *lock* y retornar `True`.

En caso de obtener el *lock*, pero que sí se haya alcanzado la capacidad máxima, libera el *lock* y retorna `False`.

Finalmente, en caso de no obtener el *lock*, retorna `False`.

- **Modificar** `def manejar_salida_cliente(self, cliente: Cliente) -> bool:`

  Método encargado de manejar la salida de un cliente. Primeramente intenta adquirir el `lock_capacidad`, pero **NO** debe quedarse esperando a que se libere el *lock* si no logra obtenerlo.

  En caso de obtener el *lock*, debe llamar al método `salida_cliente`, liberar el *lock* y retornar `True`.

  En caso de no obtener el *lock*, retorna `False`.

- **Modificar** `def run(self) -> None:`

  Método encargado de la ejecución del *thread*. Realiza los siguientes pasos:

  1. Al iniciarse la ejecución del *thread*, el establecimiento empieza a preparar el escenario.
  2. Una vez preparado el escenario, abre el establecimiento y se le avisa a los clientes.
  3. Una vez abierto el establecimiento, este estará abierto durante `DURACION_CLUB_ABIERTO` segundos.
  4. Finalizado este tiempo, el establecimiento deberá cerrar y terminar la ejecución del *thread*.

  El método incluye unos *prints* que te ayudarán a orientarte sobre las distintas acciones que suceden durante el `run`.

Parte II. Cliente

Para que los clientes puedan disfrutar del *DCClub*, deberás completar, modificar y utilizar los siguientes métodos:

- **Modificar** `def __init__(self, senal_abrio_club: Event, metodo_llega: Callable, metodo_salida: Callable) -> None:`

  Inicializador de la clase. Asigna los argumentos recibidos en los atributos que le corresponden. Adicionalmente, define el identificador único del cliente.

  Debes modificar el atributo `self.daemon` para asegurar que el programa principal no espere que este *thread* finalice para terminar con su ejecución.

- **No modificar** `def calcular_tiempo_en_club(self) -> float:`

  Método encargado de calcular el tiempo en segundos que permanecerá el cliente en el *DCClub*. Este tiempo es calculado de forma aleatoria a partir de un rango de tiempo y tiempo mínimo.

- **Modificar** `def intentar_entrar_dcclub(self) -> bool:`

  Método encargado de que el cliente intente entrar al *DCClub*. Hasta que el cliente logre entrar el método llamará el método almacenado en `entrar_al_club`.

  Si logra entrar (el método llamado retorna `True`), entonces retorna `True`. Si no logra entrar (el método llamado retorna `False`), entonces espera `TIEMPO_ESPERA_NUEVO_INTENTO` segundos y vuelve a intentar entrar al *DCClub*.

- **Modificar** `def intentar_salir_dcclub(self) -> bool:`

  Método encargado de que el cliente intente salir del *DCClub*. Hasta que el cliente logre salir el método llamará el método almacenado en `salir_del_club`.

  Si logra salir (el método llamado retorna `True`), entonces retorna `True`. Si no logra salir (el método llamado retorna `False`), entonces inmediatamente vuelve a intentar salir al *DCClub*.

- **Modificar** `def run(self) -> None:`

  Método encargado de la ejecución del *thread*. Realiza los siguientes pasos:

  1. Al iniciarse la ejecución del *thread*, espera a que el *DCClub* abra.
  2. Una vez haya abierto el *DCClub*, el cliente intenta entrar al establecimiento.
  3. Una vez dentro del establecimiento, el cliente estará adentro `calcular_tiempo_en_club` segundos.
  4. Una vez haya terminado el tiempo, el cliente intentará salir del establecimiento.
  5. Finalmente, cuando haya salido del establecimiento, terminará la ejecución del `thread`.

  El método incluye unos *prints* que te ayudarán a orientarte sobre las distintas acciones que suceden durante el `run`.

Notas

- **No puedes hacer *import* de otras librerías externas a las entregadas en el archivo.**
- Recuerda que la ubicación de tu entrega es en **tu repositorio de Git**. En la rama (*branch*) por defecto del repositorio: `main`.
- Se recomienda completar la actividad en el orden del enunciado.
- Recuerda que esta evaluación presenta corrección **automatizada**. Si entregas un código que se cae al momento de correr los *tests*, será evaluado con 0 puntos.
- Si aparece un error inesperado, ¡léelo y revisa el código del *test*! Intenta interpretarlo y/o buscarlo en Google.
- Se recomienda probar tu código con los *tests* y ejecutando `main.py`, este último se ofrece un pequeño código donde se prueba el funcionamiento del *DCClub* y 5 clientes que tratan de entrar en él.

Ejecución de *tests*

En esta actividad se provee de varios archivos `.py` los cuáles contiene diferentes *tests* que ayudan a validar el desarrollo de la actividad. Para ejecutar estos *tests*, **primero debes posicionar tu terminal/consola en la carpeta de la actividad (`Actividades/AC5`)**. Luego, desde esta misma, debes escribir el siguiente comando para ejecutar todos los *tests* de la actividad:

- `python3 -m unittest discover tests_publicos -v -b`

En cambio, si deseas ejecutar un subconjunto de *tests*, puedes hacerlo si escribes lo siguiente en la terminal/consola:

- `python3 -m unittest -v -b tests_publicos.test_dcclub`
  Para ejecutar solo el subconjunto de *tests* de la Parte I.
- `python3 -m unittest -v -b tests_publicos.test_cliente`
  Para ejecutar solo el subconjunto de *tests* de la Parte II.

Importante: recuerda que si `python3` no funciona, probar con el comando específico de tu computador. Este puede ser `py`, `python`, `py3` o `python3.12`.