Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2020-1

07 de mayo de 2020
Actividad Sumativa

Actividad Sumativa 03

Threading

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AS03/`
- **Hora del *push*:** 16:50

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**.

Introducción

Los líderes mundiales, Dr. Pin Tong-Un y Trumpzini, están teniendo una acalorada discusión filosófica por DCCahuín, que podría definir el futuro del mundo, por lo que tú, estudiante de Programación Avanzada, con preocupación decides crear una simulación que pueda predecir el incierto futuro.

Buscando ideas para esto, ten encuentras con el *Doomsday Clock*. En pocas palabras, el *Doomsday Clock* es un reloj que indica qué tan cerca estamos de una catástrofe global ocasionada por el hombre, y cada cierto tiempo un grupo de científicos se reúne a evaluar el contexto mundial, y así avanzar o retroceder la hora según corresponda. En este reloj, la catástrofe global está representada por la medianoche, y la

hora muestra qué tan lejos estamos de ella; cabe señalar que el tiempo no utiliza una escala 1:1, es decir, no porque queden 100 minutos en el reloj nos quedarán realmente 100 minutos en el mundo.

Flujo del Programa

Para poder simular esta situación y predecir el resultado final, deberás crear a Dr. Pin Tong-Un y Trumpzini, los **Líderes Mundiales**. Normalmente, un Lider Mundial tiene la capacidad de *twitear*, sin embargo, si hay otro Lider Mundial *twiteando* en ese instante, **esperará a que el otro termine antes de enviar un *tweet***. Cada vez que los líderes *twiteen*, aumentará su enojo según el nivel de enojo del *tweet*, lo que además, hará avanzar el *Doomsday Clock* hacia la medianoche. Es por esto que mientras más *tweets* hayan, más rápido se acerca el *Doomsday Clock* a la medianoche, y más rápido nos acercaremos a una catástrofe.

Archivos

- `main.py`: Este archivo contiene la clase `Simulacion`, la cual **deberás completar** según lo pedido. Además, es el archivo principal a ejecutar.
- `lideres.py`: Contiene la clase `LiderMundial`, la cual **deberás completar** según lo pedido. También contiene la clase `Hacker`, que deberás completar para el *bonus*.
- `doomsday_clock.py`: Contiene la clase `DoomsdayClock`, la cual **deberás completar** según lo pedido.
- `parametros.py`: Contiene variables que serán importadas y usadas en los archivos anteriores. Las importaciones ya están hechas y el archivo está completo, pero eres libre de modificarlo si quieres ver como afecta la simulación.
- `cargar_tweets.py`: Contiene la función `cargar_tweets`, que carga y clasifica los *tweets* de los líderes según su enojo, guardándolos como `namedtuples` dentro de una lista. Este archivo ya está completo y **NO debes modificarlo**.
- `trumpzini_tweets.csv`: Contiene la base de datos de los *tweets* de Trumpzini y se encuentra dentro del directorio `datos/`. Cada línea define uno y viene de la forma:

  `enojo;tweet`

  donde `enojo` es el nivel de enojo que causa el tweet al líder mundial y `tweet` es el *tweet* que escribió en la plataforma. Este archivo **no debes modificarlo**.
- `pin_tweets.csv`: Contiene la base de datos de los *tweets* de Dr. Pin Tong-Un y se encuentra dentro del directorio `datos/`. Tiene el mismo formato de `trumpzini_tweets.csv`. Este archivo **no debes modificarlo**.

Doomsday Clock

En primer lugar, deberás completar la clase `DoomsdayClock`, que se encuentra en el archivo `doomsday_clock.py`, de manera de poder ir contando cuántos minutos antes de la catástrofe le quedan a la humanidad.

- `class DoomsdayClock`: Esta es la clase que implementa el *Doomsday Clock*. Debes hacer que herede de la clase `Thread`.
  - `def __init__(self, velocidad, tiempo_restante)`: Esta clase tiene como atributos a `velocidad` y `tiempo_restante`, los cuales recibe como argumentos. `velocidad` indica qué tan rápido se

acerca el reloj a medianoche, mientras que `tiempo_restante` corresponde a los minutos que quedan para una eventual catástrofe global. Además, la clase tiene el atributo `quedan_lideres`, que será un *boolean* que indica si quedan o no quedan lideres para *twitear*[1]. Este valor comienza siendo `True` y cambia a `False` cuando ya no quedan líderes.

- `def tiempo_restante(self):` Es una *property* que maneja el tiempo que queda para llegar a medianoche. Esta *property* ya está implementada, por lo que **no debes modificarla.**
- `def contar(self):` Es el método encargado de avanzar el reloj. Cada vez que se ejecuta este método, *Doomsday Clock* actualiza su hora acercándose 1 minuto hacia la medianoche. Además, imprime la hora del *Doomsday Clock* cada 5 minutos y en cada uno de los últimos 5 minutos antes de medianoche. **No debes modificar este método.**
- `def run(self):` Este es el método de ejecución del *thread*, que deberá correr mientras el `tiempo_restante` sea mayor a 0 y todavía queden lideres. Deberás completar este método de forma que que cada "1/`velocidad` segundos" el reloj avance (es decir, ejecute su método `contar`). En el caso en que el reloj llegue a la medianoche (el tiempo restante sea igual a 0) y aún queden líderes, se imprimirá una bomba nuclear, representando la catástrofe (tanto la revisión de esta condición como la impresión descrita, ya están implementadas).
- `def acelerar(self, nombre, enojo):` Es el método que modela el efecto del enojo en el avance del reloj. Recibe un `str` correspondiente al `nombre` del líder que aceleró el reloj y un `int` con el nivel de `enojo` asociado a un *tweet*, y debe aumentar la velocidad de conteo (`self.velocidad`) en `enojo`/10. Además, deberá imprimir en pantalla el `nombre` de quién aceleró el reloj y cuánto cambió la velocidad. Un ejemplo de esto es:

```
"Trumpzini ha acelerado el reloj en 0.2."
```

Para probar tus avances en este módulo, puedes correr directamente el archivo `doomsday_clock.py`. El código dentro de `if __name__ == "__main__"` crea una instancia de la clase `DoomsdayClock` y ejecuta su método `run`. Cada un segundo se llamará al método `acelerar`, lo que debería aumentar la velocidad del reloj según el enojo enviado y mostrar el `print` correspondiente. Además de esto, en ese mismo intervalo de tiempo se imprimirán los atributos actuales de la instancia de `DoomsdayClock`. Si tus métodos funcionan correctamente, el *output* debería tener un formato similar a este:

```
...
"Persona de prueba ha acelerado el reloj por 0.5."
"Velocidad actual: 5.1499999999999995"
"Minutos hasta la medianoche: 32"
"11:30"
"Persona de prueba ha acelerado el reloj por 0.4."
"Velocidad actual: 5.55"
"Minutos hasta la medianoche: 27"
"11:35"
...
```

La prueba debería terminar con la explosión, cuando el `DoomsdayClock` alcanza la medianoche.

[1] Ver *bonus*

Líderes Mundiales

Para que la simulación funcione, deberás rellenar la clase que representa a los líderes mundiales. Tanto Dr. Pin Tong-Un como Trumpzini serán instancias de la misma clase, `LiderMundial`. Esta la podrás encontrar en el archivo `lideres.py`. Cada líder mundial deberá mandar *tweets* cada vez que pueda, pero **nunca podrá hacerlo al mismo tiempo que el otro**. Además, al mandar un *tweet*, el *Doomsday Clock* se verá acelerado según el enojo del *tweet* que decidieron mandar.

- `class LiderMundial`: Esta es la clase que representa a cada líder mundial, los *tweets* que realizan y cómo estos afectan su enojo. Debes hacer que herede de la clase `Thread`.
  - `def __init__(self, nombre, tweets, enojo, reloj)`: Cada instancia debe estar creada explícitamente como *daemon* y además tendrá como atributos el `nombre` del líder (`str`), sus `tweets` (lista de `namedtuple`), el nivel inicial de `enojo` (`float`) y el `reloj` correspondiente al *Doomsday Clock* (una instancia de `DoomsdayClock`); estos atributos se recibirán como argumentos. Notar que `self.tweets` corresponde a una lista de `Tweet`, dónde cada `Tweet` es una `namedtuple` con el `enojo` (`int`) y el `texto` (*string*) asociado al *tweet*. Además, tiene el atributo `puede_twitear` (`bool`), que indica si el líder puede publicar *tweets*. Si un líder no puede publicar nuevos *tweets*, este atributo será `False` y el líder no podrá `twitear`.
  - `def enojo(self)`: Es una *property* que maneja el enojo acumulado del líder mundial. **No debes modificar esta función.**
  - `def run(self)`: Este será el método que el *thread* ejecutará al comenzar. Deberás completarlo de forma que, mientras el líder mundial `puede_twitear`, se ejecute el método propio `self.twitear()` con un intervalo de $max(5 \times (1{,}05)\hat{}(-\texttt{self.enojo}), 0{,}25)$ segundos entre cada ejecución.
  - `def twitear(self)`: Cada vez que se ejecute este método, se debe elegir un `Tweet` aleatorio[2] de la lista de `tweets` del líder, sin eliminarlo de la lista, para luego imprimir el `nombre` del líder que está mandando el *tweet* junto al `texto` de este[3]. Además, deberás aumentar el nivel de enojo del líder según el `enojo` del `Tweet` y acelerar el *Doomsday Clock* con su método `acelerar`. El método acelerar necesita como argumentos el nombre de quién twitea y el enojo del *tweet*. Debes tener en cuenta que este proceso **solo se puede hacer si no hay otro líder mundial mandando *tweets***.

Para probar tus avances en este módulo, puedes correr directamente el archivo `lideres.py`:. El código dentro de `if __name__ == "__main__"` crea una instancia de la clase `LiderMundial` y ejecuta su método `run`. Si se cumple el comportamiento del enunciado, se debería ir incrementando la velocidad del reloj cada vez que el lider ejecuta `twitear`. Si tus métodos funcionan correctamente, el output debería tener un formato similar a este:

```
...
"Dr. Pin Tong-Un: I can't wait to shake Trumpzini\'s hand.  His tiny, tiny hand."
"Dr. Pin Tong-Un ha acelerado el reloj por 6."
"Dr. Pin Tong-Un: I have everything Trumpzini wants:  Unchecked power."
"Dr. Pin Tong-Un ha acelerado el reloj por 9."
"Dr. Pin Tong-Un se enojó tanto que se rompió el reloj :(."
```

La prueba debería terminar con este último `print`, y la instancia de `LiderMundial` debería dejar de *twitear*.

[2] Puedes usar el método `choice` de la librería `random`

[3] Recuerda que cada `Tweet` es una `namedtuple` de la forma `(enojo, texto)`

Simulación

Por último, deberás llenar la clase `Simulacion` en el archivo `main.py`. Esta se encargará de realizar la simulación de los *tweets* entre los líderes mundiales que provoca que el *Doomsday Clock* se acerque hacia la medianoche. Para esto, primero se crea una instancia de tipo `DoomsdayClock`: `doomsday_clock`, y luego las instancias de tipo `LiderMundial`: `dr_pinto` (Dr. Pin Tong-Un) y `trumpzini` (Trumpzini).

- `class Simulacion`: Esta es la clase que realiza la simulación en base a los *tweets* de los líderes mundiales.
  - `def __init__(self, tweets_pinto, tweets_trumpzini)`: Esta clase recibe como argumentos dos listas de `Tweet`, cada una correspondiente a un líder distinto. La primera se utiliza para crear el atributo `dr_pinto` (Dr. Pin Tong-Un) y la segunda para el atributo `trumpzini` (Trumpzini), donde cada uno es una instancia de `LiderMundial`. Además se define el atributo `doomsday_clock` y la instancia que lo representa. **No debes modificar este método**, a no ser que implementes el *bonus*.
  - `def comenzar(self)`: Este método deberá comenzar la simulación. Para esto, deberás iniciar los *threads* de *Doomsday Clock*, Dr. Pin Tong-Un y Trumpzini. Recuerda que para que el programa funcione correctamente, el thread principal no puede continuar su ejecución hasta que finalice el `doomsday_clock`.

Bonus: Hacker (1 punto)

Como *bonus* deberás simular a Lily_Yo-Jong416, hacker de notoriedad mundial y hermana de Dr. Pin Tong-Un, quién intentará hackear a los Líderes Mundiales, para detenerlos antes de que el reloj llegue a la medianoche y así salvar el planeta. Para obtener el *bonus* deberás completar la clase `Hacker` del archivo `lideres.py`, además de instanciar la clase y empezar su *thread* en la clase `Simulacion`.

- `class Hacker`: Esta es la clase que representa a Lily_Yo-Jong416.
  - `def __init__(self, nombre, trumpzini, dr_pinto, reloj)`: Esta clase tiene como atributos a `nombre` (`str`), `trumpzini` (instancia de `LiderMundial`), `dr_pinto` (otra instancia de `LiderMundial`) y `reloj` (instancia de `DoomsdayClock`), los cuales recibe como argumentos. Además, se define explícitamente que cada instancia de `Hacker` es *daemon*. **No debes modificar este método.**
  - `def run(self)`: Este será el método que el *thread* ejecutará al comenzar. Deberás completarlo de forma que cada 0.5 segundos, siempre que no haya alguien mandando *tweets*, verifique si se cumple alguna de las siguientes condiciones y actúe como se indica:
    - Si se cumple la probabilidad `PROBABILIDAD_HACKEO` y `trumpzini puede_twitear` (o sea, el *thread* sigue funcionando), debe evitar que siga *twiteando* e imprimir un mensaje indicando quién hackeó su teléfono y que no puede seguir publicando *tweets*.

```
"{nombre} ha hackeado el teléfono de Trumpzini!"
"Trumpzini ya no podrá seguir twiteando :("
```

    - Si se cumple la probabilidad `PROBABILIDAD_DESAPARECER` y `dr_pinto puede_twitear` (o sea, su *thread* sigue funcionando), debe evitar que siga *twiteando* e imprimir un mensaje avisando quién boicoteó su cirugía y que no puede seguir publicando *tweets*.

```
"{nombre} ha boicoteado la cirugía de Dr. Pin Tong-Un!"
```

```
2 "Dr. Pin Tong-Un ya no podrá seguir twiteando :("
```

Si en algún momento se cumple la condición de que ambos líderes ya no pueden *twitear*, se detiene el `reloj` y se imprime un mensaje avisando que el mundo el *Doomsday Clock* ha sido detenido. Esta ya está implementado al final del método `run`, en conjunto con la condición del *while loop*.

Notas

- Para controlar el uso de recursos simultáneamente, **debes** usar `Lock` de `threading`.
- Para que *threads* que están corriendo terminen junto a tu programa estos deberán ser *daemon*.

Requerimientos

- (2.25 pts) *Doomsday Clock*
  - (0.75 pts) Herencia correcta como *thread* correctamente.
  - (0.75 pts) Implementa el método `run` correctamente.
  - (0.75 pts) Implementa el método `acelerar` correctamente.
- (2.5 pts) Líderes Mundiales
  - (0.50 pts) Implementa el método `__init__` correctamente.
  - (0.75 pts) Implementa clase como `Thread` correctamente.
  - (0.50 pts) Implementa el método `run` correctamente.
  - (0.50 pts) Implementa el método `twitear` correctamente.
  - (0.25 pts) Se maneja concurrencia al imprimir *tweets* con el uso de `threading.Lock`
- (1.25 pts) Simulación
  - (0.75 pts) Implementa el método `comenzar` correctamente.
  - (0.5 pts) Utiliza el método `join` correctamente.
- (1.0 pts) *Bonus*