Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2021-2

23 de Septiembre de 2021
Actividad Sumativa

Actividad Sumativa 2

Threading

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AS2/`
- **Hora del *push*:** 16:40

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

¿Aburrido de que te llegue la comida fría? ¿aburrido de que te llegen los pedidos tarde? ¡Llegó tu solución! DCComidApp, la aplicación más eficiente del mercado, cuenta con una variedad de tiendas y rápidos tiempos de entrega.

Hoy serás el encargado de programar la DCComidApp, deberás encargarte de que todo funcione de forma correcta, coordinar las acciones entre la aplicación, los Shoppers y las tiendas, y asignar correctamente los pedidos.

Flujo del programa

El programa consiste en llevar a cabo la simulación de una aplicación de despachos de comida a domicilio. La aplicación irá generando pedidos desde una base de datos, donde cada pedido será ingresado a su tienda correspondiente, para luego ser asignado a uno de los Shoppers[1] disponibles (en caso de no haber Shoppers disponibles, se esperará hasta que se desocupe uno). Apenas se desocupe un Shopper, la aplicación le asignará el pedido, entregándole también las distancias tanto a la tienda como al destino (el domicilio a donde debe llegar). Desde este momento el Shopper comienza su viaje hacia la tienda.

Después de asignar el pedido a un Shopper, la aplicación ingresará el pedido a la tienda correspondiente. La tienda solo comenzará la preparación de este si es que no está preparando otro y si es que no hay un pedido por recoger. Una vez comenzada la preparación del pedido pueden ocurrir dos cosas. La primera, es que la tienda termine el pedido antes de que llegue el Shopper, situación en la cual la tienda tendrá que esperar a que este último llegue para poder entregarle dicho pedido, y comenzar a preparar el siguiente (en caso de que haya). La segunda, es que el Shopper llegue a la tienda antes de que el pedido esté listo, situación en la cual el Shopper debe esperar a que la tienda termine de prepararlo antes de recogerlo.

La simulación ejecutará la DCComidaApp, la cual tendrá varios Shoppers y tiendas, además será la encargada de mantener la comunicación entre estas dos entidades, de manera que se asignen los pedidos correctamente y no se produzcan errores en el flujo del programa.

Archivos

Archivos de datos

- `datos/shoppers.csv`: En este archivo encontrarás los datos de todos los Shoppers. Cada línea contiene la información de los Shoppers separada por una coma de la siguiente forma forma:

  `nombre_shopper,velocidad`

  donde: `nombre_shopper` corresponde al nombre del Shopper y `velocidad` a la velocidad con la que se mueve el Shopper.

- `datos/tiendas.csv`: En este archivo encontrarás la información correspondiente a cada tienda. Cada línea contiene la información de las tiendas de la siguiente forma:

  `nombre_tienda`

- `datos/pedidos.csv`: En este archivo encontrarás la información de cada pedido que se simulará en la aplicación. Cada línea contiene la información relacionada a los pedidos separadas por coma de la siguiente forma:

  `id_pedido,nombre_tienda,descripcion`

Archivos de código

- `shopper.py`: Contiene la clase `Shopper`. **Debes modificarlo**
- `tienda.py`: Contiene la clase `Tienda`. **Debes modificarlo**
- `app.py`: Contiene la clase `DCComidaApp`. **Debes modificarlo**
- `cargar_datos.py`: Contiene la funciones `cargar_shoppers`, `cargar_tiendas` y `cargar_pedidos` para la lectura de los archivos de datos. **No debes modificarlo**

[1] Un Shopper representa a un trabajador de *delivery* de la aplicación, es decir, las personas que van a buscar el pedido a la tienda y luego se lo entregan al cliente.

- `pedido.py`: Contiene la clase `Pedido`. No debes modificarlo
- `main.py`: Contiene a la simulación del programa. No debes modificarlo

Pedidos

La clase Pedido se encuentra en el archivo `pedido.py`, tiene como objetivo simular los pedidos realizados a las tiendas. Se instancia usando los datos del archivo `pedidos.csv` en la clase `DCComidApp`.

- `class Pedido`: Representa a los pedidos realizados a las distintas tiendas mediante la aplicación. Posee el siguiente método: No debes modificarlo
  - `def __init__(self, id_: int, tienda: str, descripcion: str)`: Este es el constructor de la clase. Asigna los siguientes atributos:
    - `self.id_`: `int` con el número del pedido.
    - `self.tienda`: `str` con el nombre de la Tienda a la que se relizará el pedido.
    - `self.descripcion`: `str` con la descripción del pedido.
    - `self.entregado`: `bool` que indica si el pedido fue entregado.
    - `self.distancia_destino`:`int` que puede tomar un valor aleatorio entre 1 y 10. Representa la distancia desde la tienda hasta el lugar de entrega del pedido.
    - `self.evento_llego_repartidor`: `Event` que indica cuando el repartidor ha llegado a recoger el pedido.
    - `self.evento_pedido_listo` : `Event` que indica que el pedido ya terminó de ser preparado por la tienda y está listo para ser entregado al Shopper.

Parte I: Shopper

En esta parte deberás completar la clase Shopper que se encuentra en el archivo `shopper.py`, la cual tiene el objetivo de simular a los trabajadores de *delivery* en la entrega de pedidos. En específico debes completar los métodos `__init__`, `avanzar` y `run`.

- `class Shopper`: Representa a un Shopper o trabajador de *delivery*. Debe heredar de la clase `Thread`, incluye los siguientes métodos: Debes modificarlo
  - `def __init__(self, nombre: str, velocidad: int)`: Este es el constructor de la clase en el cual debes asignar los atributos `self.nombre` y `self.velocidad`. Además, la clase posee los siguientes atributos: Debes modificarlo
    - `self.posicion`: `int` que indica la posición actual del repartidor, comienza en 0 para cada pedido.
    - `self.distancia_tienda`: `int` que indica la distancia hasta la tienda del pedido actual. Comienza en 0.
    - `self.distancia_destino`: `int` que indica la distancia hasta el destino del pedido actual. Comienza en 0.
    - `self.pedido_actual`: Instancia `Pedido` que correspondiente al pedido en curso, `None` en caso de no tener un pedido asignado.

- `self.termino_jornada`: `bool` que indica si el repartidor ha terminado su jornada de trabajo. Comienza en `False`
- `self.ocupado`: `@property` que retorna un `bool` indicando si el Shopper tiene un pedido en curso.

- `def avanzar(self)`: Este método se encarga exclusivamente de mover al Shopper, cada vez que se llame se debe aumentar en 1 su posición actual y esperar un tiempo de 1/`self.velocidad`. Cuando haga esto, debe imprimir un mensaje que contenga el nombre del Shopper y la posición a la que avanzó. **Debes modificarlo**
- `def run(self)`: Este método representa la ejecución del *thread*. Acá debes implementar el funcionamiento del Shopper mediante el siguiente algoritmo mientras su atributo `self.termino_jornada` sea falso y no hayan pedidos en curso[2]: **Debes modificarlo**
  - Si hay un pedido asignado, se debe llamar al método `avanzar` para mover al Shopper.
  - Si la posición actual alcanzó la distancia hasta la tienda, entonces se debe informar en la consola y activar el evento del pedido `evento_llego_repartidor`, luego se debe esperar hasta que sea activado el evento del pedido `evento_pedido_listo`.
  - Por otro lado, si la posición actual alcanzó la distancia de destino, entonces se debe informar en la consola y hacer verdadero el atributo `entregado` del pedido actual. Además, se debe activar el evento `evento_disponible` de la clase, setear la posición del Shopper en 0 y quitar el pedido actual del Shopper.
- `def asignar_pedido(self, pedido: Pedido)`: Este método se encarga de asignar un nuevo pedido para el Shopper. Recibe una instancia de `Pedido` y lo guarda en su atributo `self.pedido_actual`, además se calcula la distancia a la tienda y la distancia hasta el destino. **No debes modificarlo**

Parte II: Tienda

En esta parte deberás completar la clase Tienda del archivo `tienda.py`, la cual tiene el objetivo de simular a las tiendas que preparan los pedidos.

- `class Tienda`: corresponde a las tiendas disponibles en DCComidaApp. Debe heredar de la clase `Thread` y deberás completarla de modo que su ejecución termine con el resto del programa y pueda utilizar `Lock` en su ejecución, incluye los siguientes métodos: **Debes modificarlo**
  - `def __init__(self, nombre: str)`: corresponde al constructor de la clase. Debes asignar el atributo `self.nombre` con el argumento entregado y crear un `Lock` único para cada Tienda que utilicen para alterar la cola de pedidos. Incluye los siguientes atributos **ya implementados**: **Debes modificarlo**
    - `self.nombre`: `str` que corresponde al nombre de la tienda.
    - `self.cola_pedidos`: `list` que corresponde a la cola de pedidos que la Tienda debe preparar.
    - `self.abierta`: `bool` que corresponde al estado de la Tienda, si esta está abierta o no.
  - `def ingresar_pedido(self, pedido: Pedido, shopper: Shopper)`: Este método recibe una instancia de pedido y otra de Shopper. Se encarga de ingresar el pedido a la tienda para ser rea-

[2] Puedes utilizar la *property* `self.ocupado` para verificar si hay un pedido en curso

lizado. Para ello deberás agregar una tupla que relacione el Pedido con el Shopper a la cola de pedidos que deberá realizar la Tienda. **Debes utilizar Lock en este método**. Debes modificarlo

- `def preparar_pedido(self, pedido: Pedido)`: Este método recibe una instancia de Pedido y se encarga de prepararlo. Debe generar un `int` random entre `1` y `10` que indicará el tiempo que se tardará en realizar el pedido. Debes imprimir un mensaje indicando el tiempo de demora del pedido antes de su preparación, además de un mensaje tras su finalización. Debes modificarlo
- `def run()`: Este método se encarga de iniciar el `thread` y preparar los pedidos asignados. Se debe ejecutar el siguiente algoritmo mientras la Tienda se encuentre abierta: Debes modificarlo
  - Si es que hay pedidos en la cola de pedidos, se debe seleccionar el primer pedido y sacarlo de la cola.
  - Se debe preparar el pedido seleccionado con el método `self.preparar_pedido`.
  - Cuando el pedido esté listo, se debe activar el evento `evento_pedido_listo` del objeto Pedido.
  - Luego se debe esperar a que el repartidor llegue a buscar el pedido, esperando el evento `evento_llego_repartidor` del objeto Pedido. Cuando el repartidor llegue, se debe imprimir que el pedido ha sido retirado.
  - Por último, en caso de no haber pedidos en la cola, entonces la tienda se tomará un descanso aleatorio entre 1 y 5, imprimiendo un mensaje indicando lo anterior.

  Debes utilizar Lock en este método.

Parte III: DCComidApp y Simulación

En esta parte deberás completar la clase `DCComidApp` en el archivo `app.py`, la cual tiene el objetivo de coordinar las acciones entre las tiendas y los shoppers para satisfacer los pedidos. En específico debes completar los métodos `obtener_shopper` y `run` de la clase `DCComidApp`.

- `class DCComidApp`: esta clase corresponde a la aplicación, contiene los pedidos realizados a las diferentes tiendas de la `DCComidApp` y a los Shoppers registrados en ella. Hereda de la clase `Thread` e incluye los siguientes métodos: . Debes modificarlo
  - `def __init__(self, shoppers:list, tiendas:dict, pedidos:list)`: Este es el constructor de la clase `DCComidApp`, posee los siguientes atributos: No debes modificarlo
    - `self.shoppers`: `list` que contiene a todos los Shoppers (instancias de la clase `Shopper`).
    - `self.pedidos`: `list` que contiene listas con los parámetros necesarios para instanciar los pedidos en la clase `Pedido`. Estos corresponden a los pedidos realizados a las distintas tiendas..
    - `self.tiendas`: `dict` donde la llave corresponde al nombre de la tienda y el valor a una instancia `Tienda`.
  - `def obtener_shopper(self) -> Shopper`: Este método se encarga de encontrar un Shopper disponible para ir a recoger el pedido a la tienda y llevarlo al destino. Busca en la lista de Shoppers a uno disponible, si encuentra uno lo retorna. Si todos los Shoppers están ocupados imprime un mensaje notificándolo y espera a la señal `evento_disponible` de la clase Shopper, luego imprime un mensaje notificando que se desocupó un Shopper para finalmente reiniciar la señal y buscar nuevamente al Shopper disponible. Debes modificarlo

- `def run(self)`: Este método se encarga de simular la DCComidaApp. Aca debes implementar la DCComidaApp mientras queden pedidos en la lista `self.pedidos`: **Debes modificarlo**
  - En primer lugar se extrae el primer pedido de la lista `self.pedidos` y se busca la tienda a la que se hará el pedido en `self.tiendas`,
  - Luego, se crea la instancia del pedido actual.
  - Una vez instanciado el pedido, busca a un Shopper disponible mediante el metodo `obtener_shopper` (descrito mas adelante) y le asigna el pedido haciendo uso del método del Shopper `asignar_pedido`.
  - Finalmente, ingresa el pedido a la tienda usando el método `ingresar_pedido` de la clase `Tienda`.
  - Por último, antes de ingresar el siguiente pedido se debe esperar un tiempo aleatorio entre 1 y 5 que simulará el tráfico de la red.

Notas

- Se pueden modificar el archivo `pedidos.csv` con tal de acortarlo y probarlo con menor cantidad de pedidos. Recomendamos probar con los 5 primeros pedidos para evaluar funcionalidad.
- La función `sleep()` del módulo `time` recibe como argumento el tiempo **en segundos** durante los cuales el *thread* actual se detendrá.

Requerimientos

- (2.00 pts) Clase Shopper
  - (0.5 pts) Completar `def __init__()`
  - (0.25 pts) Completar `def avanzar()`
  - (1.25 pts) Completar `def run()`
- (2.00 pts) Clase Tienda
  - (0.5 pts) Completar `def __init__()`
  - (0.5 pts) Completar `def ingresar_pedido()`
  - (0.5 pts) Completar `def preparar_pedido()`
  - (0.5 pts) Completar `def run()`
- (2.00 pts) Clase DCComidApp
  - (1 pts) Completar `def obtener_shopper()`
  - (1 pts) Completar `def run()`