Tarea 3: Sistema de Pedidos de Comida con Concurrencia

> Trabajo individual

Objetivo

Implementar un sistema de gestión de pedidos de comida que utilice patrones de diseño y técnicas de programación concurrente para procesar múltiples pedidos simultáneamente.

Enunciado

Debe crear un sistema que simule un restaurante donde:

- Clientes realizan pedidos de diferentes tipos de pedidos (hamburguesas y pizzas)
- Cocineros (hilos independientes) procesan los pedidos concurrentemente

A continuación se adjunta como debe verse el main que ejecuta el programa y un ejemplo de cómo podrían verse los logs.

Código base

```
# Main para prueba
if __name__ == "__main__":
    servicio = ServicioPedidos()

    # Factory para hamburguesas
    creador_hamburguesas = CreadorHamburguesas()
    for i in range(3):
        pedido = creador_hamburguesas.crear_pedido(i)
        servicio.agregar_pedido(pedido)

    # TODO: Agregar otro factory para pizzas

    servicio.procesar_pedidos()
```

Logs de dicha ejecución

> Ejemplo de salida completa esperada:

```
[COCINERO 1] Preparando pedido 0 (Hamburguesa)
[COCINERO 2] Preparando pedido 1 (Pizza)
[COCINERO 1] Pedido 0 listo: Hamburguesa 0 preparada
[COCINERO 1] Preparando pedido 2 (Hamburguesa)
[COCINERO 2] Pedido 1 listo: Pizza 1 preparada
[COCINERO 1] Pedido 2 listo: Hamburguesa 2 preparada
[SISTEMA] Todos los pedidos procesados
```

Por lo tanto, debe agregar los siguientes componentes:

1. Sistema de concurrencia con hilos

    - Modificar la clase `ServicioPedidos` para que use hilos (cada cocinero es un hilo que procesa pedidos de la cola).
    - Implementar un sistema de cola segura para threads.
    - Asegurar la sincronización adecuada para evitar condiciones de carrera.
2. Múltiples tipos de pedidos
    - Agregar un nuevo factory para pizzas y completar el de hamburguesas.
    - Permitir que el sistema maneje múltiples tipos de pedidos sin modificar la lógica de procesamiento.

Más consideraciones:

- Pueden agregar la cantidad que hilos que deseen, siendo un ejemplo recomendado 2 o 3.
- Pueden suponer y agregar lógica, métodos, atributos o demás que sea necesario para su solución. Todo puede ser aceptable con su debida justificación y que no modifique mucho el main esperado.
- Investiguen una cola "thread safe" para que los workers puedan consumir pedidos de forma correcta.

Evaluación detallada

Código (80%):

- Implementación de concurrencia (30%)
- Patrón Factory correctamente aplicado (25%)
- Sincronización y manejo de colas (15%)
- Calidad y estructura del código (10%)

Documentación (20%):

- Explicación de decisiones de diseño (10%)
- Claridad en comentarios (10%)

Entrega

1. Código completo en Python (.py o .ipynb). La entrega puede ser adjuntando un zip a mediación o lo suben a un repositorio personal.
2. Breve explicación de las decisiones de diseño (en archivo adjunto o README).