# Programación Concurrente 2025

## Trabajo Práctico Final

## Condiciones

- El trabajo es grupal, de 5 alumnos (grupo de 4 es la excepción).
- La defensa del trabajo se coordinará con el grupo respecto a modalidad presencial o videoconferencia.
- La evaluación es individual (hay una calificación particular para cada integrante).
- Solo se corrigen los trabajos que hayan sido subidos al aula virtual (LEV).
- Los problemas de concurrencia deben estar correctamente resueltos y explicados.
- El trabajo debe implementarse en lenguaje Java.
- Se evaluará la utilización de objetos y colecciones, como así también la explicación de los conceptos relacionados a la programación concurrente.

### Red de Petri Sistema de Procesamiento de Datos

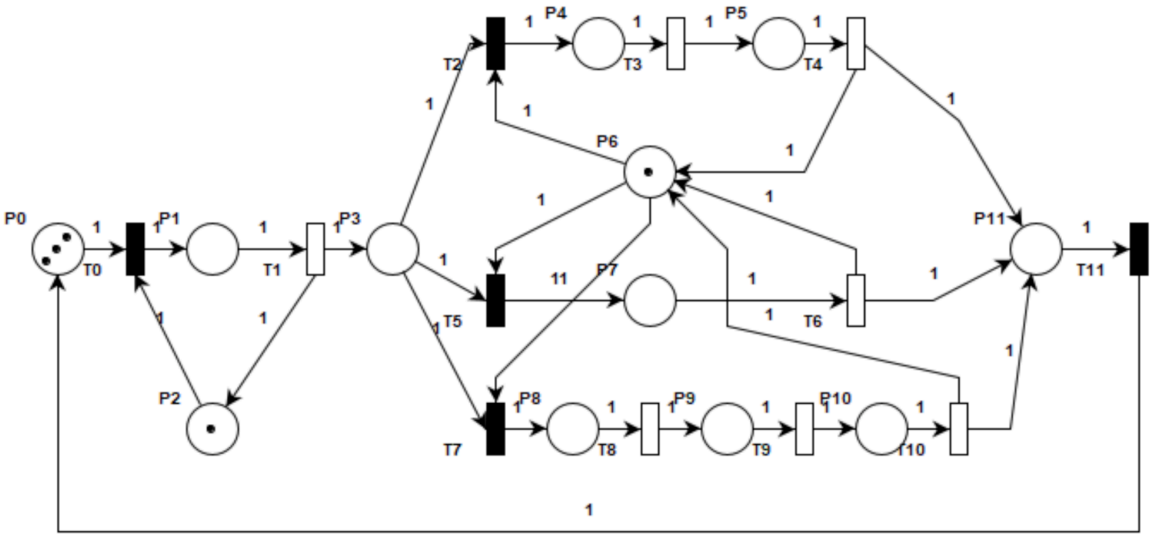


***Figura 1***

*Link descarga archivo red de Petri (PIPE).*

## Enunciado

En la ***Figura 1*** se observa una red de Petri que modela un sistema de procesamiento de datos.
Las plazas {P2, P6} representan recursos compartidos en el sistema:

- La plaza {P2} representa el bus de acceso al buffer.
- La plaza {P6} representa la unidad de procesamiento.

Las plazas {P3, P11} representan buffers:

- La plaza {P3} representa el buffer de los datos a procesar.
- La plaza {P11} representa el buffer de salida de los datos.

La plaza {P0} es una plaza idle que corresponde a la cola de arribo de datos del sistema.

La plaza {P1} representa un dato accediendo al buffer a través del bus.
Cada dato puede procesarse por uno de los siguientes tres modos de procesamiento:

- Modo de complejidad simple: una sola etapa llevada a cabo en la plaza {P7}
- Modo de complejidad media: dos etapas llevadas a cabo en las plazas {P4, P5}
- Modo de complejidad alta: tres etapas llevadas a cabo en las plazas {P8, P9, P10}

# Propiedades de la Red

- Es necesario determinar con PIPE las propiedades de la red (deadlock, vivacidad, seguridad).
- Indicar cual o cuales son los invariantes de plaza y los invariantes de transición de la red. Realizar una breve descripción de lo que representan en el modelo.

# Implementación

Es necesario implementar un monitor de concurrencia para la simulación y ejecución del modelo:

- Realizar una tabla, con los estados del sistema.
- Realizar una tabla, con los eventos del sistema.
- Determinar la cantidad de hilos necesarios para la ejecución del sistema con el mayor paralelismo posible (ver Referencias *1):
    - Caso 1: si el invariante de transición tiene un conflicto, con otro invariante, debe haber un hilo encargado de la ejecución de la/s transición/es anterior/es al conflicto y luego un hilo por invariante.
    - Caso 2: si el invariante de transición presenta un join, con otro invariante de transición, luego del join debe haber tantos hilos, como token simultáneos en la plaza, encargados de las transiciones restantes dado que hay un solo camino.
    - Realizar un gráfico donde pueda observarse las responsabilidades de cada hilo con diferentes colores. A modo de ejemplo se observa en la ***Figura 2*** una red y cómo presentar lo solicitado, las flechas coloreadas representan cada tipo y cantidad de hilos.

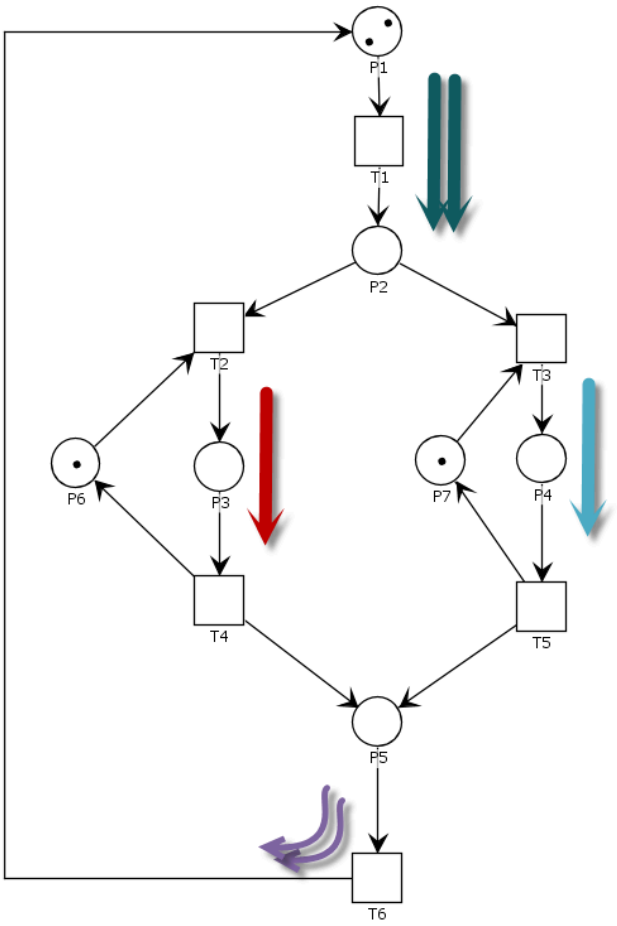


***Figura 2***

El programa debe tener la siguiente interfaz, y debe ser implementada por la clase Monitor desarrollada.

```
public interface MonitorInterface {
   boolean fireTransition(int transition);
}
```

CONSIDERACIONES IMPORTANTES:

- El método ***fireTransition*** del monitor debe ser el único método público que expone la clase Monitor.
- La clase monitor, no puede contener ninguna referencia a transiciones puntuales de la red en cuestión. Es decir, el monitor debe ser agnóstico a la red que está ejecutando. De esta manera, cambiando la red de Petri, la clase Monitor no sufre ningún cambio.

# Tiempo

Una vez implementado el monitor de concurrencia, con el sistema funcionando, se deberá implementar la semántica temporal. Las transiciones {T1, T3, T4, T6, T8, T9, T10} son

transiciones temporales. Implementarlas y asignarles un tiempo (a elección del grupo) en milisegundos.
Se debe hacer un análisis temporal tanto analítico como práctico (ejecutando el proyecto múltiples veces), justificando los resultados obtenidos. Además, es necesario también variar los tiempos elegidos, analizar los resultados y obtener conclusiones.

## Políticas

Es necesario para el modelado del sistema implementar políticas que resuelvan los conflictos. Se requiere considerar dos casos (ejecutados y analizados por separado e independientes uno de otro):

1. **Una política aleatoria:**
    - Los datos se procesan por cualquiera de los tres modos de procesamiento, de manera aleatoria.
2. **Una política de procesamiento priorizada:**
    - Se debe priorizar el modo de procesamiento simple.

## Requerimientos

1) El proyecto debe ser modelado con objetos en Java, haciendo uso de un monitor de concurrencia para guiar la ejecución de la red de Petri.
    a) El programa debe poseer una clase Main que al correrla, inicie el programa.
    b) Es un requisito que el programa debe finalizar, no pueden quedar hilos activos al terminar la ejecución.
2) Implementar un objeto Política que cumpla con los objetivos establecidos en el apartado Políticas.
3) Hacer el diagrama de clases que modele el sistema.
4) Hacer el diagrama de secuencia que muestre el disparo exitoso de una transición que esté sensibilizada, mostrando el uso de la política.
5) Indicar la cantidad de hilos necesarios para la ejecución y justificar de acuerdo a lo mencionado en el apartado Implementación (ver Referencias *1).
6) Realizar múltiples ejecuciones con 200 invariantes completados (para cada ejecución), y demostrar con los resultados obtenidos:
    a) El cumplimiento de las políticas implementadas en la distribución de la carga en los invariantes.
    b) La cantidad de cada tipo de invariante, justificando el resultado.
7) Registrar los resultados del punto **7)** haciendo uso de un archivo de log para su posterior análisis.
8) Hacer un análisis de tiempos, de acuerdo a lo mencionado en el apartado Tiempo.
    a) El programa debe demorar entre 20 y 40 segundos.
9) Mostrar e interpretar los invariantes de plazas y transiciones que posee la red.
10) Verificar el cumplimiento de los invariantes de plazas luego de cada disparo de la red.
11) Verificar el cumplimiento de los invariantes de transiciones mediante el análisis de un archivo log de las transiciones disparadas al finalizar la ejecución. El análisis de los

invariantes debe hacerse mediante expresiones regulares. Tip: www.regex.com - www.debuggex.com.

## Entregables

a) Un archivo de imagen con el diagrama de clases, en buena calidad.
b) Un archivo de imagen con el diagrama de secuencias, en buena calidad.
c) El código fuente Java (proyecto) de la resolución del ejercicio.
d) Un informe obligatorio que documente lo realizado, explique el código, los criterios adoptados y que explique los resultados obtenidos.

## Subir al LEV el trabajo **TODOS** los participantes del grupo.

# Fecha de entrega

17 de Junio de 2025

### Referencias bibliográficas

*1 : Artículo cantidad de hilos.
https://www.researchgate.net/publication/358104149_Algoritmos_para_determinar_cantidad_y_responsabilidad_de_hilos_en_sistemas_embebidos_modelados_con_Redes_de_Petri_S_3_PR1