# Taller 2 - Sincronización de Threads - Exclusión mutua

El propósito de este taller es entender cómo funciona la exclusión mutua y realizar algunos ejemplos prácticos. De manera específica, en este taller vamos a realizar la búsqueda del valor máximo global en una matriz, así como una serie de ejercicios que permitirán afianzar los conocimientos en sincronización de Threads. Con ello en mente tenemos:

## Ejercicio 1 – Búsqueda concurrente del valor máximo en una matriz

Escriba un programa en Java que revise una matriz numérica y busque, de manera concurrente, el valor máximo de la matriz. Para ello, guíese con el diseño de la solución presente en las diapositivas de clase que incluye las clases **Máximo y T.** Incluya mecanismos de exclusión mutua para resolver la condición de carrera del laboratorio anterior.

## Ejercicio 2 – Asignación de Ids en forma dinámica

A partir del ejercicio anterior, queremos cambiar la forma en la que asignamos identificadores a cada uno de los Threads. En vez de asignar identificadores de forma secuencial cuando los threads son creados, cada thread debe averiguar qué identificador le corresponde.

Para esto cree una clase y un objeto "id" de esa clase. Este objeto será encargado de llevar la secuencia y responder a los threads cuando estos pidan un identificador. Este objeto tendrá un método *darId()* que retornará números de 0 a n-1 en secuencia. Observe que al momento de crear los threads en el *main(),* los threads necesitarán conocer la referencia a "id" para poder invocar el método *darId()* y así conseguir su identificador (que se utilizará como número de fila a analizar de la matriz). Tenga en cuenta que el método darId debe ser sincronizado. ¿por qué?

Después de tener el identificador, cada thread podrá empezar a buscar el máximo como en el ejercicio anterior.

## Ejercicio 3 – Pastelería

Se quiere modelar el proceso de entrega de un pastel personalizado que se encargó a una panadería por medio de threads. Para ello, escriba un programa en Java donde un thread representa al cliente y otro thread al pastelero. El cliente ordena su pastel y queda en espera para ir a recogerlo. El cliente solo podrá recoger el pastel en el momento en el que el pastelero avise que el pastel está terminado.

Para comunicarse, el cliente y el pastelero deben compartir un objeto de clase Pastel. Un pastel tiene dimensiones (alto y radio), sabor y color de la cubierta - recuerde que todos los atributos deben ser privados. El pastel también debe tener los métodos hacerPedido (para configurar los atributos del pastel), consultarDetallesPedido (para consultar los atributos del pastel) y consultarEstadoPedido (que indica si el cliente ya configuró los atributos del pastel). Observe que estos métodos deben ser sincronizados, ¿por qué?

El pastelero debe implementar espera activa: revisar constantemente si el cliente ya definió el pastel que quiere. Si aún no lo ha hecho, entonces espera 5 segundos (por medio de sleep) y luego vuelve a revisar. Cuando encuentra que el cliente ya definió el pastel que quiere, hace el pastel; simularemos este proceso usando sleep con un tiempo aleatorio entre 5 y 15 segundos. Cuando el pastelero termina avisa al cliente que el pastel está listo y termina su ejecución.

El cliente debe implementar espera pasiva: configura el pastel que quiere y luego espera sobre el pastel hasta que el pastelero le avisa que el pastel está listo. Para implementar espera pasiva use las instrucciones “wait” y “notify”.

Adicione mensajes que le permitan verificar el avance del cliente y del pastelero.

Nota: Para realizar este ejercicio necesitará leer primero la sección **1.4.2 Eventos** de las notas del curso.

# Historial de revisiones

**Registro de cambios**

| Fecha | Autor(es) | Versión | Referencia de Cambios |
|---|---|---|---|
| **...** | **Carlos Eduardo Gómez Montoya<br>Jairo Emilio Bautista Mora<br>Geovanny Andrés González Rodríguez** | **1.0** | **Versión Inicial** |
| 01/08/2022 | Andrés Hernández León | 2.0 | Adaptación por temas de clase |

**Revisores**

| Nombre | Cargo | Institución |
|---|---|---|
| Sandra Rueda | Profesor Asociado Departamento de Ingeniería de Sistemas y Computación | Universidad de los Andes |
| Harold Castro | Profesor Titular Departamento de Ingeniería de Sistemas y Computación | Universidad de los Andes |