Universidad Autónoma de Nuevo León

Facultad de Ciencias Físico Matemático

Práctica #11

Laboratorio de Base de Datos

Alumno:

Alexis Ibarra Rodriguez

Grupo:	**Matricula:**
7 a 9 am	2094647

Docente:

JOSE ANASTACIO HERNANDEZ SALDAÑA

11 de noviembre del 2025

Documentación de Conceptos de Concurrencia en Java

1. Race Condition

Archivo Clave: `CuentaBancaria.java`
Estrategia de Sincronización:
Se presenta una **condición de carrera (Race Condition)** cuando varios hilos acceden y modifican una variable compartida (por ejemplo, el saldo bancario) al mismo tiempo, provocando resultados inconsistentes.
Para solucionar este problema, se utilizó la palabra clave **`synchronized`** en los métodos `depositar()` y `retirar()`, garantizando que solo un hilo acceda al recurso compartido a la vez.

2. synchronized

Archivos Clave: `CuentaBancaria.java`, `BufferCompartido.java`
Estrategia de Sincronización:
El modificador **`synchronized`** se emplea para asegurar exclusión mutua, es decir, que únicamente un hilo pueda ejecutar un bloque o método sincronizado al mismo tiempo.
En estos archivos, se aplicó `synchronized` en los métodos críticos para evitar interferencias en el acceso a variables compartidas como el **saldo** o el **buffer**.

3. wait(), notify(), notifyAll()

Archivo Clave: `BufferCompartido.java`
Estrategia de Sincronización:
Estos métodos permiten la **comunicación entre hilos** mediante el mecanismo de **espera y notificación**.
En este caso:

- El **productor** llama a `wait()` cuando el buffer está lleno, suspendiendo su ejecución hasta que el **consumidor** libere espacio.

- Cuando el consumidor retira un elemento, se utiliza `notifyAll()` para despertar a los hilos productores que estaban esperando.
  Esto permite una coordinación eficiente entre los hilos sin desperdiciar recursos del sistema.

4. Thread vs Runnable

Archivos Clave: `CajeroThread15.java`, `ClienteRunnable04.java`
Estrategia de Sincronización:
Se demuestra la diferencia entre **extender la clase `Thread`** y **implementar la interfaz `Runnable`**:

- **Thread (Cajero):** se hereda directamente de la clase `Thread`, permitiendo una implementación más sencilla cuando la clase no necesita heredar de otra.

- **Runnable (Cliente):** se implementa la interfaz `Runnable`, lo cual es más flexible y permite que la clase pueda heredar de otras, además de fomentar la reutilización del código.

5. Deadlock

Archivo Clave: `DeadlockDemo.java`
Estrategia de Sincronización:
Se demuestra intencionalmente un **interbloqueo (Deadlock)** al intentar adquirir dos recursos (`RECURSO_A` y `RECURSO_B`) en **orden inverso** desde dos hilos distintos.
Este ejemplo muestra cómo los hilos pueden quedar bloqueados esperando indefinidamente por recursos que nunca se liberarán, resaltando la importancia del **orden consistente de adquisición de recursos**.

6. ExecutorService

Archivo Clave: `ThreadPoolDemo.java`
Estrategia de Sincronización:
Se utiliza la API de concurrencia de Java mediante **`ExecutorService`** con **`Executors.newFixedThreadPool()`** para gestionar múltiples tareas de manera eficiente.
Esta estrategia evita manejar manualmente la creación y finalización de hilos, mejorando el rendimiento y la escalabilidad del sistema al reutilizar un conjunto de hilos (pool) controlado por el programa.