Bloque 2. Paradigmas de la Programación

Tarea 3: Definir el concepto de thread-safe. Poner un ejemplo de código thread-safe y otro que no lo sea.

1.Definición Thread-Safe:

Llamamos thread-safe a un fragmento de código o clase que es capaz de funcionar sin generar errores mientras está siendo accedida por varios hilos que se están ejecutando de forma simultánea.
En Java tendríamos como ejemplo la clase StringBuffer, siendo thread-safe. Esto quiere decir que una instancia de dicha clase podrá ser utilizada por varios hilos en paralelo. Por otro lado,la clase StringBuilder, por ejemplo, no es thread-safe.
La mayoría de los problemas derivan de una mala planificación respecto a las variables compartidas.

Para escribir códigos thread-safe podemos hacer uso de alguna de estas herramientas:

- Mediante exclusión mutua: el evitar el libre acceso a las variables compartidas por parte de los hilos, por ejemplo, haciendo uso de semáforos, cerrojos, o con el método synchronized. De esta manera garantizamos el acceso a una variable por parte de un único hilo en un mismo instante.
- Hacer uso de variables locales, para que de esta manera, una variable no pueda ser accesible desde cualquier hilo.
- Usar constantes (finally).

Nuestro principal objetivo es el de evitar los Deadlocks, los livelocks, la inanición y las condiciones de carrera, y lograr un resultado determinista.

2.Ejemplo básico de código Thread-Safe:

En este ejemplo haremos uso del método synchronized, de forma que sólo un hilo podrá ejecutar la función.

```
class ThreadSafe {
	private static int contador = 0;

	public static synchronized int Contar() {
		return contador++;
	}
}
```

3.Ejemplo básico de código NO Thread-Safe:

```
class ThreadNoSafe {
	private static int contador = 0;

	public static int getContar() {
		return contador++;
	}
}
```

Este método funcionará correctamente mientras un único hilo haga uso del código. En el caso de que sean más, nos podrá generar problemas.
Supongamos un hilo h1 que accede al método en el instante t1 y empieza a hacer uso del método.
Supongamos que aparece otro hilo h2 que requiere también hacer uso del método, al no haber exclusión mutua, nada le impide acceder al método a la par que h1 sigue trabajando con él.
Esto nos podrá generar disparidad en los resultados en función de qué hilo llegue antes y modifique la información.

Ejemplo completo con Ruby, NO thread safe:

```ruby
class DoorLock
 def initialize(locked)
   @locked = locked
 end
  def open?
   !@locked
 end

 def unlock!
   unless open?
     puts "Opening the door!"
     @locked = false
   end
 end
end

door_lock = DoorLock.new(true)
 5.times.map do
 Thread.new do
   unless door_lock.open?
     door_lock.unlock!
   end
 end
end.each(&:join)
```

Algunos resultados que podemos obtener:

```
ruby main.rb
Opening the door!
Opening the door!
Opening the door!
Opening the door!
Opening the door!
```

```
ruby main.rb
Opening the door!
```

Confirmamos que este código no es determinista, ya que el resultado de su ejecución depende del hilo que llegue y se ejecute antes. También destacaremos que, precisamente por esta carencia de exclusión mutua, este código no es thread-safe.

Ahora, el código thread-safe en Ruby podría consistir en:

```ruby
class DoorLock
 def initialize(locked)
   @locked = locked
   @mutex = Mutex.new
 end
  def open?
   @mutex.synchronize { !@locked }
 end

 def unlock!
   unless open?
     @mutex.synchronize do
       puts "Opening the door!"
       @locked = false
     end
   end
 end
end
```

En este código estamos empleando Mutex (Mutual Exclusion), que hace uso de un cerrojo que sólo permitirá el acceso a un hilo de una vez, de forma que podremos controlar el acceso a las variables compartidas.

4.Bibliografía empleada:

thread safe code : Java Glossary. (2020). Retrieved 28 October 2020, from https://www.mindprod.com/jgloss/threadsafe.html
Consultas en:
https://stackoverflow.com/questions/261683/what-is-the-meaning-of-the-term-thread-safe
Medium. 2020. Multithreaded Ruby — Synchronization, Race Conditions And Deadlocks. [online] Available at: [Accessed 29 October 2020].