4. La creación de hilos o su gestión por las aplicaciones está permitida

. a. Verdadero b. Falso

35. Which of the following statement are true:

* string builder es generalmente más rápido qué string buffer.

* string buffer is threadsafe; stringbuildder is not.

2. Hilos (Threads) Considera el siguiente bloque de código:

```
class MyThread extends Thread {
 public void run() {
System.out.println("Thread is running");
}
 }
public class Main {
public static void main(String[] args) {
 Thread t1 = new MyThread();
 Thread t2 = new MyThread();
t1.start();
 t2.start();
 }
 }
```

¿Cuál sería la salida en consola al ejecutar este código?

1. Thread is running (impreso una vez)

2. Thread is running (impreso dos veces)

3. Thread is running (imp res o dos veces , en orden aleatorio)

4. Compilation error

6. Hilos y Sincronización

Considera el siguiente bloque de código:

```
class Counter {
   private int count = 0;

   public synchronized void increment() {
      count++;
   }

   public int getCount() {
      return count;
   }
}

class MyThread extends Thread {
   private Counter counter;

   public MyThread(Counter counter) {
      this.counter = counter;
   }

   public void run() {
      for (int i = 0; i < 1000; i++) {
         counter.increment();
      }
   }
}
public class Main {
   public static void main(String[] args) throws InterruptedException {
      Counter counter = new Counter();
      Thread t1 = new MyThread(counter);
      Thread t2 = new MyThread(counter);
      t1.start();
      t2.start();
      t1.join();
      t2.join();
      System.out.println(counter.getCount());
```

```
    }
}
```

¿Cuál sería la salida en consola al ejecutar este código?

[redacted]

1. 2 00 0
2. 1000
3. Variable count is not synchronized
4. Compilation error

Explicación:

1. La clase Counter tiene una variable count y un método **sincronizado** increment(), lo que significa que **solo un hilo a la vez puede ejecutar este método**.
2. La clase MyThread recibe un objeto Counter y lo comparte entre dos hilos (t1 y t2). Cada hilo ejecuta counter.increment() **1000 veces** en un bucle.
3. En el main(), creamos una única instancia de Counter y dos hilos (t1 y t2), que **comparten** esa misma instancia.
4. Se llama a t1.start(); y t2.start(); para iniciar los hilos en paralelo.
5. Se usa t1.join(); y t2.join(); para asegurarse de que ambos hilos terminen antes de imprimir counter.getCount().
6. Como el método increment() está **sincronizado**, no hay condiciones de carrera y el resultado siempre será **2000** (1000 incrementos por cada hilo).

9. Concurrencia y Sincronización

Considera el siguiente bloque de código:

```
class SharedResource {
   private int count = 0;

   public synchronized void increment() {
      count++;
   }

   public synchronized void decrement() {
      count--;
   }

   public int getCount() {
      return count;
   }
}

class IncrementThread extends Thread {
   private SharedResource resource;

   public IncrementThread(SharedResource resource) {
      this.resource = resource;
   }

   public void run() {
      for (int i = 0; i < 1000; i++) {
         resource.increment();
      }
   }
}

class DecrementThread extends Thread {
   private SharedResource resource;

   public DecrementThread(SharedResource resource) {
      this.resource = resource;
```

```
    }

    public void run() {
        for (int i = 0; i < 1000; i++) {
            resource.decrement();
        }
    }
}

public class Main {
    public static void main(String[] args) throws InterruptedException {
        SharedResource resource = new SharedResource();
        Thread t1 = new IncrementThread(resource);
        Thread t2 = new DecrementThread(resource);
        t1.start();
        t2.start();
        t1.join(); t2.join();
        System.out.println(resource.getCount());
    }
}
```

¿Cuál sería la salida en consola al ejecutar este código?

1. 1000
2. 0
3. -1000
4. Compilation error

1. La clase SharedResource tiene una variable count que se modifica mediante los métodos sincronizados increment() y decrement().
2. La clase IncrementThread incrementa count 1000 veces.
3. La clase DecrementThread decrementa count 1000 veces.
4. En el main(), se crea una única instancia de SharedResource, compartida por dos hilos (t1 y t2).
5. Se inicia t1 (incrementa 1000 veces) y t2 (decrementa 1000 veces) en paralelo.
6. t1.join(); y t2.join(); aseguran que ambos hilos terminen antes de imprimir resource.getCount().
7. Como cada incremento tiene su correspondiente decremento 8pero no se sabe como se va intercalando la ejecución de cada hilo, ya que esto ira cambiando de acuerdo a la el valor final de count será 0.