**Can we provide thread-count as negative value for Parallel Execution?**

No, you cannot provide a negative value for the thread-count in parallel execution.

In frameworks like **TestNG**, the thread-count parameter defines the number of threads to be used for parallel execution. This value must be a positive integer because:

1. **Negative Values are Invalid**: A negative thread count doesn't make sense in terms of thread management. It would lead to undefined behavior or errors during execution.
2. **Validation in TestNG**: TestNG explicitly checks for valid configurations. Providing a negative thread-count would result in an error or exception.
3. **Logical Meaning**: A thread pool cannot have a negative number of threads—it must have at least one thread to perform execution.

**What Happens if You Provide a Negative Value?**

* If a negative value is used for thread-count in TestNG, it will throw an **IllegalArgumentException** or a similar runtime error, depending on the implementation.

**How TestNG Manages More Threads than Tests?**

### Example:

Suppose you have:

* 5 test methods in total.
* thread-count="10" (more threads than tests).

#### Parallel Execution Flow:

1. TestNG starts with 10 threads.
2. Each of the 5 test methods gets assigned to a thread.
3. The remaining 5 threads will stay idle.

### Implications:

* **Wasted Resources**: The additional threads consume memory and other system resources without contributing to execution.
* **No Performance Benefit**: The execution time will not improve with unused threads.
* **Overhead**: In some cases, creating and managing excessive threads can introduce unnecessary overhead.

**What is invocationCount attribute in TestNG?**

The **invocationCount** attribute in **TestNG** specifies the number of times a particular test method should be executed. It is useful for scenarios where you want to repeatedly run a test to verify stability, handle concurrency, or perform stress testing.

**Key Features:**

1. **Repeat Execution**: The test method is executed as many times as specified by the invocationCount.
2. **Supports Parallelism**: When used with threadPoolSize, the invocations can be executed in parallel.
3. **Default Value**: The default value is 1, meaning the test method runs only once unless explicitly set.

**Syntax:**

**@Test**(invocationCount = 5)

public void testMethod() {

System.out.println("Executing test method");

}

### Advanced Usage with threadPoolSize:

You can combine invocationCount with threadPoolSize to execute the invocations in parallel.

**Example:**

**@Test**(invocationCount = 10, threadPoolSize = 5)

public void testMethod() {

System.out.println("Thread: " + Thread.currentThread().getId());

}

#### Behavior:

* The test method runs 10 times (invocationCount = 10).
* Up to 5 threads (threadPoolSize = 5) will execute the invocations in parallel.

**What is invocationTimeOut attribute TestNG?**

The **invocationTimeOut** attribute in **TestNG** sets the maximum time (in milliseconds) allowed for executing all invocations of a test method specified by the invocationCount attribute. If the total execution time of all invocations exceeds this timeout, the test will fail.

**Key Features:**

1. **Time Limit**: Ensures that repeated executions of a test method complete within a specific time frame.
2. **Works with invocationCount**: It is meaningful only when invocationCount is set to a value greater than 1.
3. **Use Case**: Useful for performance testing or ensuring that a method doesn't hang or take too long during repeated executions.

**Syntax:**

**@Test**(invocationCount = 5, invocationTimeOut = 2000)

public void testMethod() throws InterruptedException {

Thread.sleep(500); // Simulates some processing time

System.out.println("Test method executed");

}

**Behavior:**

1. **invocationCount = 5**: The test method runs 5 times.
2. **invocationTimeOut = 2000**: All 5 invocations must complete within 2 seconds.

**What Happens if Timeout is Exceeded?**

If the total execution time of all invocations exceeds the specified timeout:

* TestNG **fails the test**.
* A failure message like "java.lang.ThreadTimeoutException" is thrown.

Example with Timeout Exceeded:

**@Test**(invocationCount = 5, invocationTimeOut = 2000)

public void testMethod() throws InterruptedException {

Thread.sleep(600); // Simulates longer processing time

System.out.println("Test method executed");

}