Not exactly! **ExecutorService can use submit() to run both Runnable and Callable tasks**. The key difference is how results are handled. Let’s break it down:

**🚀 execute() vs submit() in ExecutorService**

| **Method** | **Takes** | **Returns** | **Exception Handling** |
| --- | --- | --- | --- |
| execute(Runnable task) | Runnable | void (no result) | Throws exception immediately if the task fails |
| submit(Runnable task) | Runnable | Future<?> (no actual result, but can check completion) | Exceptions are captured inside Future |
| submit(Callable<T> task) | Callable<T> | Future<T> (returns a result) | Exceptions are captured inside Future |

**✅ Example: execute() with Runnable**

import java.util.concurrent.\*;

public class ExecuteExample {

public static void main(String[] args) {

ExecutorService executor = Executors.newFixedThreadPool(2);

Runnable task = () -> {

System.out.println(Thread.currentThread().getName() + " is executing a Runnable task");

};

executor.execute(task); // No return value

executor.shutdown();

}

}

**Output**

pool-1-thread-1 is executing a Runnable task

🚀 **Key Point**: execute() just runs the task, and you **can’t check the result or handle exceptions properly**.

**✅ Example: submit() with Runnable**

import java.util.concurrent.\*;

public class SubmitRunnableExample {

public static void main(String[] args) {

ExecutorService executor = Executors.newFixedThreadPool(2);

Runnable task = () -> {

System.out.println(Thread.currentThread().getName() + " is executing a Runnable task");

};

Future<?> future = executor.submit(task); // Returns a Future object

executor.shutdown();

}

}

🚀 **Key Point**: Even though Runnable has no return value, submit() **returns a Future<?>**, which can be used to check if the task is done.

**✅ Example: submit() with Callable**

import java.util.concurrent.\*;

public class SubmitCallableExample {

public static void main(String[] args) throws Exception {

ExecutorService executor = Executors.newFixedThreadPool(2);

Callable<String> task = () -> {

return Thread.currentThread().getName() + " executed a Callable task";

};

Future<String> future = executor.submit(task); // Returns Future<String>

System.out.println("Result: " + future.get()); // Blocks until result is available

executor.shutdown();

}

}

**Output**

Result: pool-1-thread-1 executed a Callable task

🚀 **Key Point**: submit() with Callable lets you **retrieve a result and handle exceptions properly**.

**🔥 Summary**

| **Case** | **Use execute()** | **Use submit()** |
| --- | --- | --- |
| **Runnable (no result needed)** | ✅ | ✅ (but with Future<?>) |
| **Runnable (need exception handling)** | ❌ | ✅ (exceptions are captured inside Future) |
| **Callable (return result)** | ❌ | ✅ (returns Future<T>) |

**Best Practice**

* Use **execute()** for simple fire-and-forget tasks.
* Use **submit()** when you need to **handle exceptions or retrieve results**.

Would you like an example of handling exceptions using Future? 🚀