**Runnable vs Callable in Java**

Both Runnable and Callable are used to define tasks that can be executed in a thread, but there are key differences.

**✅ Runnable**

* Introduced in **Java 1.0**.
* **Does not return a result**.
* **Cannot throw checked exceptions**.
* Used with Thread or ExecutorService.

🔹 **Example of Runnable**

class MyRunnable implements Runnable {

@Override

public void run() {

System.out.println("Running a task...");

}

}

public class RunnableExample {

public static void main(String[] args) {

Thread thread = new Thread(new MyRunnable());

thread.start();

}

}

🔹 **Output**

Running a task...

**✅ Callable**

* Introduced in **Java 5** (part of java.util.concurrent).
* **Returns a result** (Future<T>).
* **Can throw checked exceptions**.
* Used with ExecutorService.submit().

🔹 **Example of Callable**

import java.util.concurrent.\*;

class MyCallable implements Callable<String> {

@Override

public String call() throws Exception {

return "Task completed!";

}

}

public class CallableExample {

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

ExecutorService executor = Executors.newSingleThreadExecutor();

Future<String> future = executor.submit(new MyCallable());

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

executor.shutdown();

}

}

🔹 **Output**

Result: Task completed!

**🚀 Key Differences**

| **Feature** | **Runnable** | **Callable** |
| --- | --- | --- |
| **Introduced in** | Java 1.0 | Java 5 |
| **Return Type** | void | T (generic return type) |
| **Checked Exception** | Cannot throw | Can throw |
| **Usage** | Thread or ExecutorService | ExecutorService |
| **Method** | run() | call() |

**🔥 When to Use Which?**

✔ **Use Runnable** if you don’t need a result.  
✔ **Use Callable** if you need a result or want to handle exceptions.

Would you like an example where Callable is used for parallel computation? 🚀