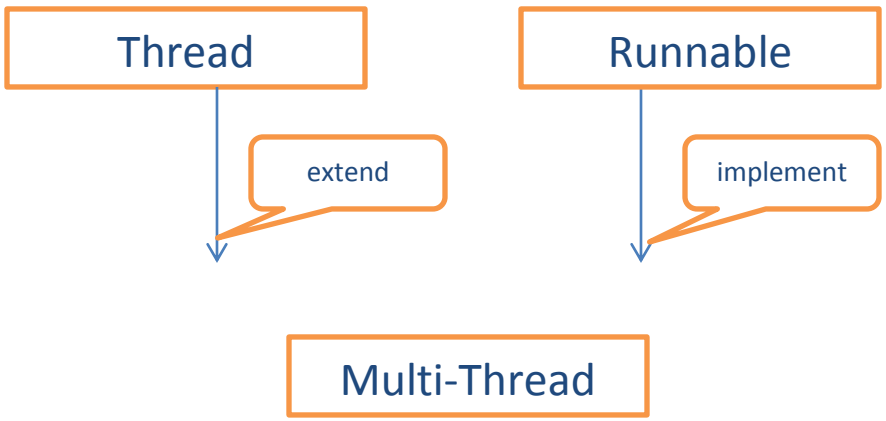
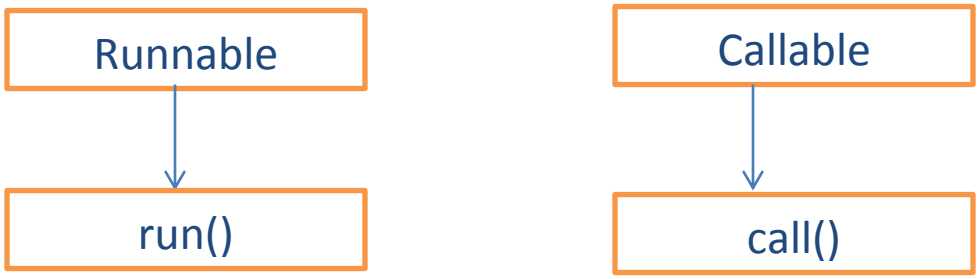
How to achieve multithreading or threading or thread :



Missing :

To create thread we need to override run() method. But run() method does not return anything when it finishes its execution. So, for supporting this feature, the **Callable interface** is present in Java 5, **java.util.concurrent package**



Note: a thread can't be created with a Callable, it can only be created with a Runnable.

- Another difference is that the call() method can throw an exception whereas run() cannot.
- The run() method does not return anything, while for a Callable, the call() method needs to be implemented which returns a result on completion

```
public interface Callable<V> {
    V call() throws Exception;
}
```

Let's have a look at calculating the factorial of a number:

```
class FactorialTask implements Callable<Integer> {
	int number;
	FactorialTask(int number){
		this.number=number;
	}
	// standard constructors

	public Integer call() throws Exception {
		int fact = 1;
		// ...
		for (int count = number; count > 1; count--) {
			fact = fact * count;
		}

		return fact;
	}
	public void whenTaskSubmitted_ThenFutureResultObtained() throws
Exception{
		FactorialTask task = new FactorialTask(5);
		ExecutorService  executorService =
		Executors.newSingleThreadExecutor();
		Future<Integer> future = executorService.submit(task);
		System.out.println(future.get().intValue());
	}
}
```

The result of *call()* method is returned within a *Future* object:

Here is the code for an example Callable, which will return a random number after a delay of around 0 – 4 seconds.

```
		// Java program to illustrate Callable
		// to return a random number
		import java.util.Random;
		import java.util.concurrent.Callable;
		import java.util.concurrent.FutureTask;

		public class CallableEg implements Callable {

		public Object call() throws Exception {
		// Create random number generator
		Random generator = new Random();

		Integer randomNumber = generator.nextInt(5);

		// To simulate a heavy computation,
		// we delay the thread for some random time
		Thread.sleep(randomNumber * 1000);

		return randomNumber;
		}
		}
```

Future :

How will you identify that a particular thread has been executed and done its task. Your answer will be content of run() method. But if we say that a particular thread will return such results that can be stored in an object known to the main thread, so that the main thread can know about the result that the thread returned.

Question : How will the program store and obtain this result later?

A Future object can be used. Think of a Future as an object that holds the result. Thus, a Future is basically one way the main thread can keep track of the progress and result from other threads.

Callable and Future do two different things – Callable is similar to Runnable, in that it encapsulates a task that is meant to run on another thread, whereas a Future is used to store a result obtained from a different thread. In fact, the Future can be made to work with Runnable as well.

To create the thread, a Runnable is required. To obtain the result, a Future is required.

The Java library has the concrete type **FutureTask**, which implements Runnable and Future, combining both functionality conveniently. A FutureTask can be created by providing its constructor with a Callable. Then the FutureTask object is provided to the constructor of Thread to create the Thread object. Thus, indirectly, the thread is created with a Callable. For further emphasis, note that ***there is no way to create the thread directly with a Callable.***

```
//Java program to illustrate Callable and FutureTask
//for random number generation
import java.util.Random;
import java.util.concurrent.Callable;
import java.util.concurrent.FutureTask;

class CallableExample implements Callable {

	public Object call() throws Exception {
		Random generator = new Random();
		Integer randomNumber = generator.nextInt(5);

		Thread.sleep(randomNumber * 1000);

		return randomNumber;
	}

}

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

		// FutureTask is a concrete class that
		// implements both Runnable and Future
```

```
		FutureTask[] randomNumberTasks = new FutureTask[5];

		for (int i = 0; i < 5; i++) {
			Callable callable = new CallableExample();

			// Create the FutureTask with Callable
			randomNumberTasks[i] = new FutureTask(callable);

			// As it implements Runnable, create Thread
			// with FutureTask
			Thread t = new Thread(randomNumberTasks[i]);
			t.start();
		}

		for (int i = 0; i < 5; i++) {
			// As it implements Future, we can call get()
			System.out.println(randomNumberTasks[i].get());

			// This method blocks till the result is obtained
			// The get method can throw checked exceptions
			// like when it is interrupted. This is the reason
			// for adding the throws clause to main
		}
	}
}
```

Runnable tasks can be run using the *Thread* class or ***ExecutorService*** whereas ***Callables*** can be run only using the latter.