* It is the implementation of Future and CompletionStage.
* If I Call a remote service, which is totally unknown what it is doing. The result might arrive in 10ms, 100ms, 1000ms, 2s etc. I don’t have any control over when I will be getting the result.
  + Ex: String result = service.getResult(); Here the call is blocked till I receive the receive the result from service. If the service is down, then we get exception after pre-determined time.
* The best way of calling the remote service is to
  + run in background without blocking the main flow
  + run with timeout to avoid waiting for ever.

**Callable<String> task = () -> c.myRemoteService();**

**ExecutorService exec = Executors.*newFixedThreadPool*(1);**

**Future<String> future = exec.submit(task);**

* + Future promises to give a String result in Future which is not yet present, result may come in 10ms, 100ms, 1000ms, 2s etc. but gives a String handler
* **Problem with Future:** The only way to interact with future is **future.get()** which is a **blocking call**. Also you can specify a Timeout in seconds, hours or even days.
  + **future.get(1, TimeUnit.*SECONDS*).** So this code waits or blocks the main thread upto 1 sec to get the promised result from future.
* **Does Future is blocking call?** No. Future is a proxy Mechanism which wraps the actual service call. Future has methods like **isDone()**, **cancel()**. But getting the result from future, i.e future.get() is blocking call.

**future1.get(1, TimeUnit.*SECONDS*);**

**future2.get(1, TimeUnit.*SECONDS*);**

* In the above code, future2 is blocked till the future1 result arrives.
* Future get() method is blocking. Lets say we fire 2 task, task1 and task2 on Executor Service. If task2 is completed prior, we can get the result of task2 only on future2. If task1 is completed prior, we can get the result of task1 only on future1.From coding we can loop through all the future randomly for first completed result which is awkward. Execution Service doesn't provide method to access completed task first. ExecutorCompleteService provides a method take(), which returns the result of first completed task and so on. As of Java 8 you can use the CompletableFuture interface which provide a callback interface and called once a task is completed. See Code
  + ExecutorServiceDownSides.java
  + ExecutorCompleteServiceSample.java
* **Future<String> service1Result = CompletableFuture.*supplyAsync*(() -> myRemoteService1());**
  + Or
* **CompletableFuture<String> service1Result = CompletableFuture.*supplyAsync*(() -> myRemoteService1());**
* **CompletableFuture** has a callback method **thenAccept()** which will be called when the result arrives from remote service.

ExecutorService exec1 = Executors.*newFixedThreadPool*(2);

CompletableFuture<String> service1Result = CompletableFuture.*supplyAsync*(() -> myRemoteService1(),exec1);

CompletableFuture<String> service2Result = CompletableFuture.*supplyAsync*(() -> myRemoteService2(),exec1);

service1Result.thenAccept((String result) -> System.*out*.println((result)));

service2Result.thenAccept((String result) -> System.*out*.println((result)));