Exercises: Futures and Promises

DCC-FCUP, University of Porto
José Proença

Concurrent Programming – Part 2

These exercises are inspired mainly from the book "*Learning Concurrent Programming in Scala*". You are required to implement operations that work correctly in concurrent settings, using Scala's notions of futures and promises. The exercises are not ordered in any particular order, but some assume earlier exercises have been done.

We recommand starting with the code that you produced for the exercises for the Java memory model.

Exercise 1. Re-implement **using futures** a `parallel` method, which takes two computation blocks, `a` and `b`, and starts each of them in a new thread. The method must return a tuple with the result values of both the computations. It should have the following signature:

```
def parallel[A, B](a: =>A, b: =>B): (A, B)
```

Do not use explicitly creation of new threads as before. Use the `log` function when testing your code.

Exercise 2. Re-implement **using futures** a `periodically` method, which takes a time interval duration specified in milliseconds, and a computation block `b`. The method starts a thread that executes the computation block `b` every duration milliseconds, even if the previous computation did not finish yet. It should have the following signature:

```
def periodically(duration: Long)(b: =>Unit): Unit
```

Exercise 3. Implement and test a `SyncVal` class **using a promise in the internal state** with the following interface:

```
class SyncVal[T] {
  def isEmpty(): Boolean = ???
  def get(): T = ???
  def put(x: T): Unit = ???
  def getWait(): T = ???
}
```

A `SyncVal` object can be used to exchange values between two or more threads. When created, the `SyncVar` object is **empty**:

- `isEmpty` returns true;
- `get` throws an exception;
- `put` adds a value to the `SyncVal` object.
- `getWait` blocks until it can return a value;

After a value is added to a `SyncVal` object, we say that it is **non-empty**:

- `isEmpty` returns false;
- `get` returns the value;
- `put` throws an exception.
- `getWait` also returns the value;

Exercise 4. Consider the code below that gets the content of a webpage as a list of strings, one for each line.

```
object WebpageSearch extends App {
  /** Gets the text from a given URL as a list of strings. */
  def getUrlLines(url: String): List[String] = {
    val f = Source.fromURL(url)
    try f.getLines.toList finally f.close()
  }

  /** Finds lines where a given keyword appears. */
  def find(lines: List[String], keyword: String): String =
    (for ((line,n) <- lines.zipWithIndex
          if line.contains(keyword)))
     yield (n,line)
    ).mkString("\n")

  ...
}
```

4.1. Use futures to get the content of the website `https://www.w3.org/Addressing/URL/url-spec.txt` into a variable `futSpec`.

4.2. Calculate a new future `futFindTelnet: Future[String]` that searches for the keyword `"telnet"` in the result of the `futSpec` using the `find` method.

4.3. Print (using `log`) a message upon finishing to collect the url-specs (saying `"reading url-spec completed"` in case of success, and `"failed to read url-spec"` in case of failure).

4.4. Print (using `log`) a message with the result from the telnet search once it is received.