```
1 - Thread
Chamada de Thread do tipo Runnable, sem ter que estar o objeto

Thread t = new Thread(runnable);
t.setName("My Thread");
t.setPriority(100);
t.setUncaughtExceptionHandler(handler);
t.start();
```

```
(doto (Thread. runnable)
  (.setName "My Thread")
  (.setPriority 100)
  (.setUncaughtExceptionHandler handler)
  (.start))
```

2 - Comparação de String

Clojure consegue comparar mesmo se a String for nula.
Em java caso for nula terá um

```
if("Hello".equals(myString)) {
….
```

```
  (if (= my-string "Hello")
```

3 – Macro

No Clojure consegue se fazer uma Macro, enquanto no Java utilizando try-catch não consegue se movimentar após finalizar o try-catch

After I learnt the macro system of Lisp, I had a real enlightenment. The macro system provides you an opportunity to extend the language; you can think that you could add a feature into the language. In fact, a macro is kind of a function which produces function(s) for you, **at compile time**.

In other words, you can get rid of duplicate codes you don't want by writing nice macros.

For example, let's suppose we have a nested function thread as follows:

```
(if-let [a 1]
  (if-let [b 2]
    (if-let [c 3]
      (+ a b c))))
=> 6
```

Now as you well know, the code is not nice at all, but if I write a macro that will produce this code at the compile time with a more classy syntax, everything will be as I want.

Let's write our macro called **if-let***:

```
(defmacro if-let*
  ([bindings then]
   `(if-let* ~bindings ~then nil))
  ([bindings then else]
   (if (seq bindings)
     `(if-let [~(first bindings) ~(second bindings)]
        (if-let* ~(vec (drop 2 bindings)) ~then ~else)
        ~else)
     then)))
```

I know it looks very complicated, but I would like to show a macro example.

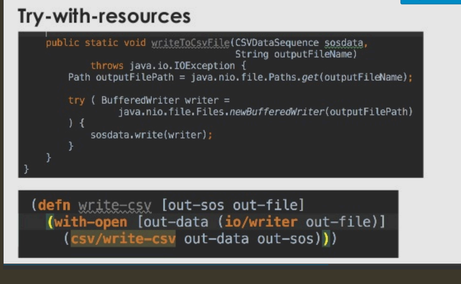
Now the nested code will be produced for me, but I will only write the following part:

```
(if-let* [a 1
          b 2
          c 3]
         (+ a b c))
=> 6
```

Of course, there can be much more complex examples. I have the chance of decreasing code on a great scale and the honor of having clearer code through using macro in Clojure projects.

Java programmers know that *try-with-resources* feature is available in Java 7 version. Therefore, IO classes which are written in **try()** are automatically closed, when their progresses are finished. For example, a macro in Clojure called **with-open** settled this progress in the standard library.

Briefly, in Clojure, you don't have to wait for minor changes on different versions as in the other languages (*Java etc.*). We can add these features by writing cool macros 🙂




```
4 - Sequences -  Retorno diferentes de nulo e java retorna false
;; Corner cases
(seq nil)   ;;=> nil
(seq '())   ;;=> nil
(seq [])    ;;=> nil
(seq "")    ;;=> nil
```