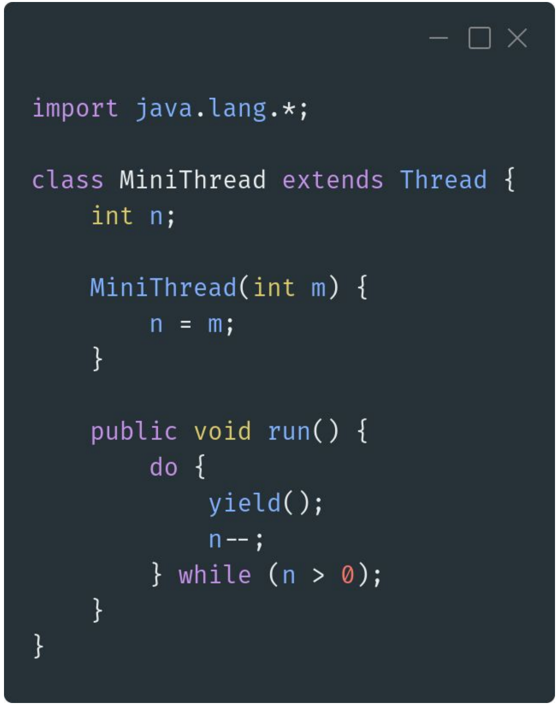
Morte por concorrência

Roberto Alves Neto

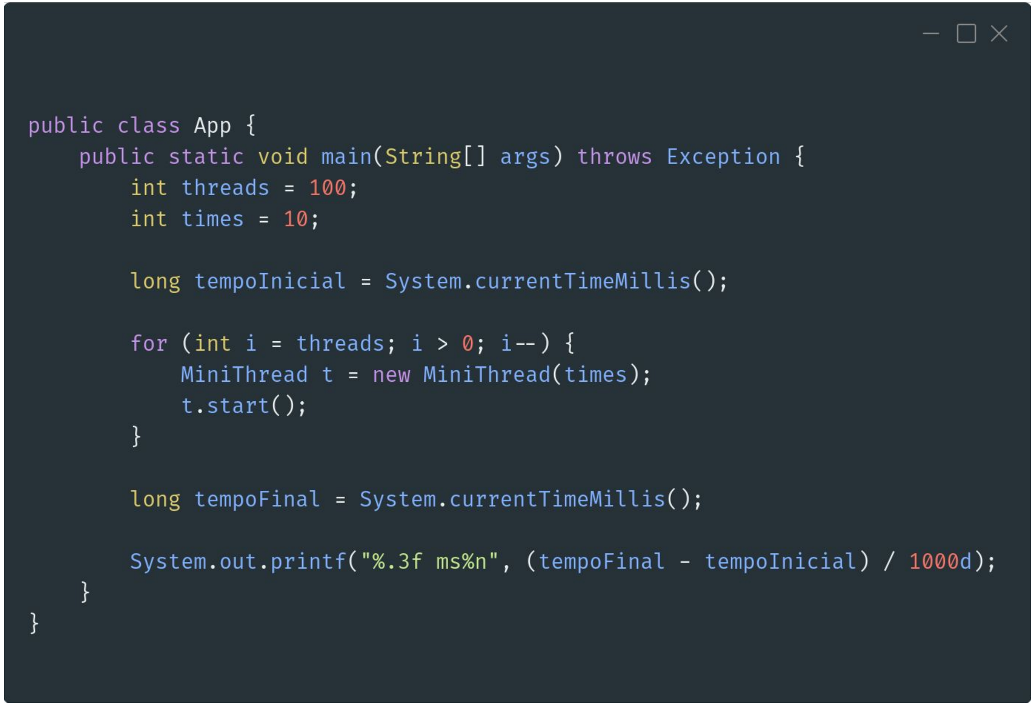


```java
import java.lang.*;

class MiniThread extends Thread {
    int n;

    MiniThread(int m) {
        n = m;
    }

    public void run() {
        do {
            yield();
            n--;
        } while (n > 0);
    }
}
```



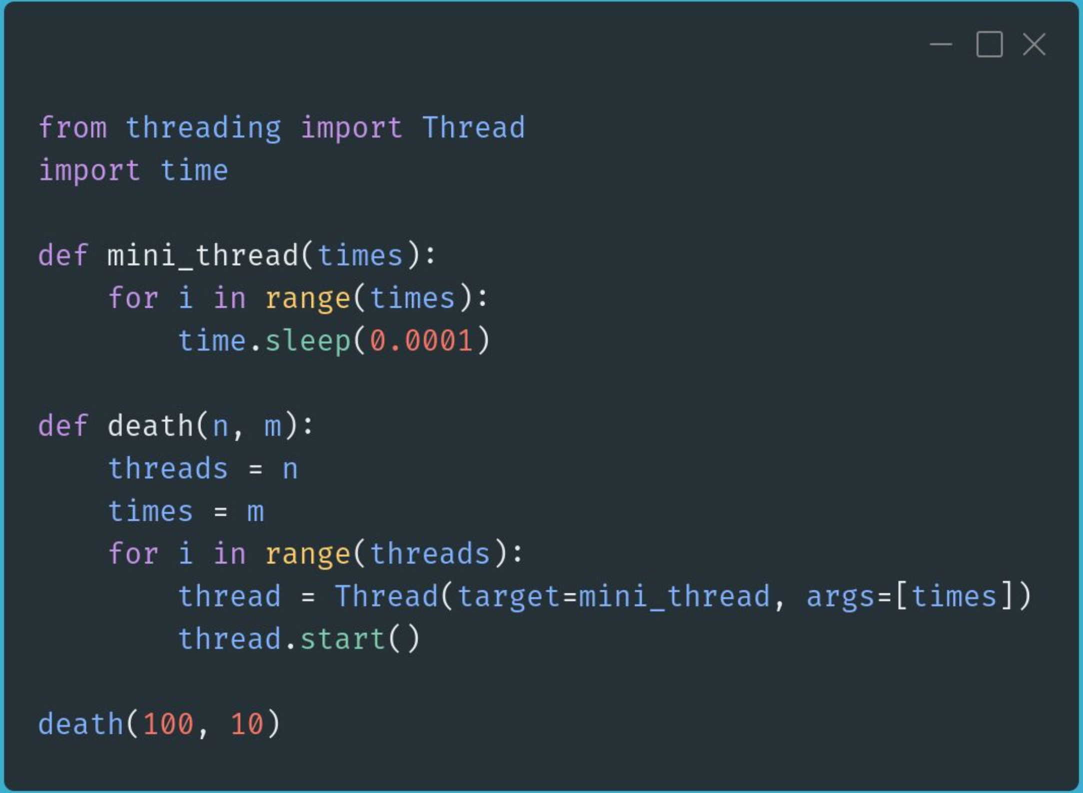
```java
public class App {
    public static void main(String[] args) throws Exception {
        int threads = 100;
        int times = 10;

        long tempoInicial = System.currentTimeMillis();

        for (int i = threads; i > 0; i--) {
            MiniThread t = new MiniThread(times);
            t.start();
        }

        long tempoFinal = System.currentTimeMillis();

        System.out.printf("%.3f ms%n", (tempoFinal - tempoInicial) / 1000d);
    }
}
```

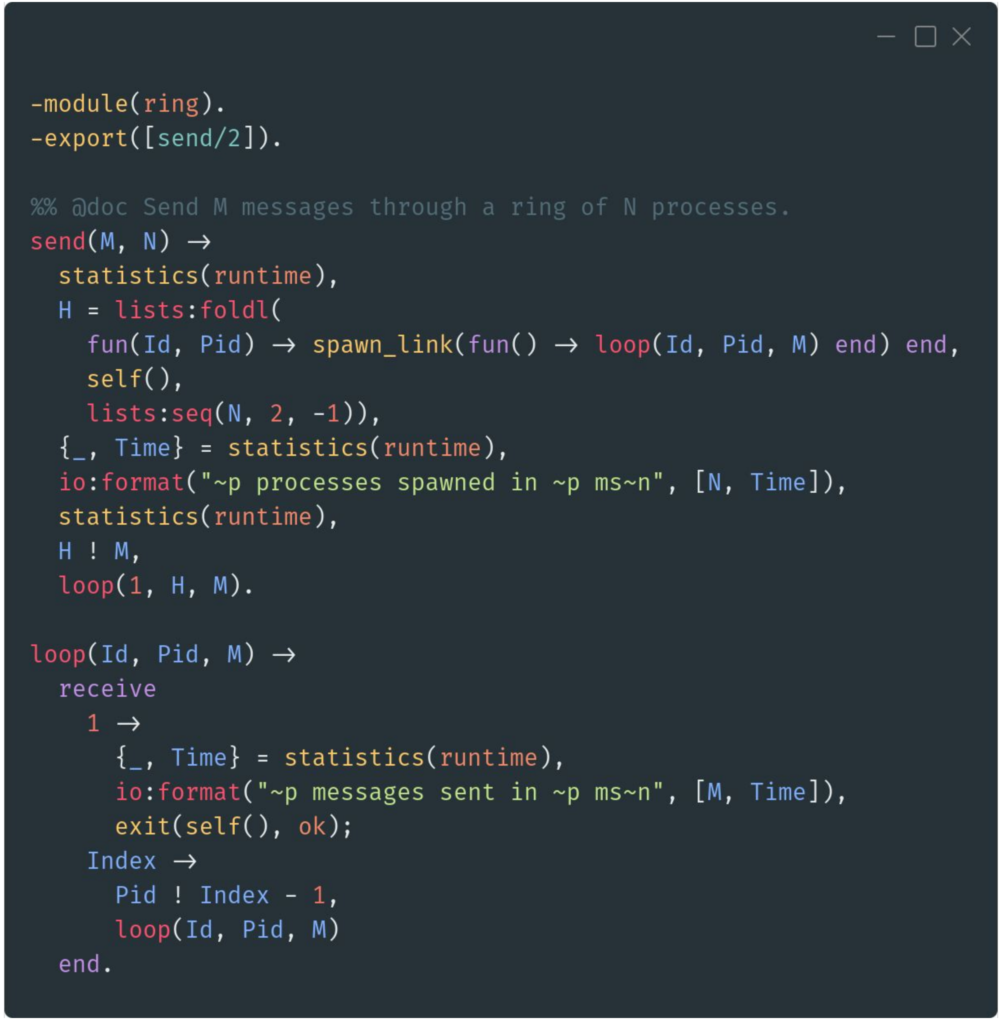


```python
from threading import Thread
import time

def mini_thread(times):
    for i in range(times):
        time.sleep(0.0001)

def death(n, m):
    threads = n
    times = m
    for i in range(threads):
        thread = Thread(target=mini_thread, args=[times])
        thread.start()

death(100, 10)
```



```erlang
-module(ring).
-export([send/2]).

%% @doc Send M messages through a ring of N processes.
send(M, N) ->
  statistics(runtime),
  H = lists:foldl(
    fun(Id, Pid) -> spawn_link(fun() -> loop(Id, Pid, M) end) end,
    self(),
    lists:seq(N, 2, -1)),
  {_, Time} = statistics(runtime),
  io:format("~p processes spawned in ~p ms~n", [N, Time]),
  statistics(runtime),
  H ! M,
  loop(1, H, M).

loop(Id, Pid, M) ->
  receive
    1 ->
      {_, Time} = statistics(runtime),
      io:format("~p messages sent in ~p ms~n", [M, Time]),
      exit(self(), ok);
    Index ->
      Pid ! Index - 1,
      loop(Id, Pid, M)
  end.
```

ERLANG

Comparação tempos

	100 P e 100 M	1000 P e 1000 M	10000 P e 1000 M	100000 P e 10000 M
Java	0,007 ms	0,014 ms	0,225 ms	2,342 ms
Python	0,059 ms	5,48 ms	61,82 ms	+10 min. Desisti
Erlang	0,001 ms	0.009 ms	0,039 ms	0,418 ms