Printing FooBar n Times

This problem is about executing threads in an order for a user specified number of times.

Problem

Suppose there are two threads t1 and t2. t1 prints **Foo** and t2 prints **Bar**. You are required to write a program which takes a user input n. Then the two threads print Foo and Bar alternately n number of times. The code for the class is as follows:

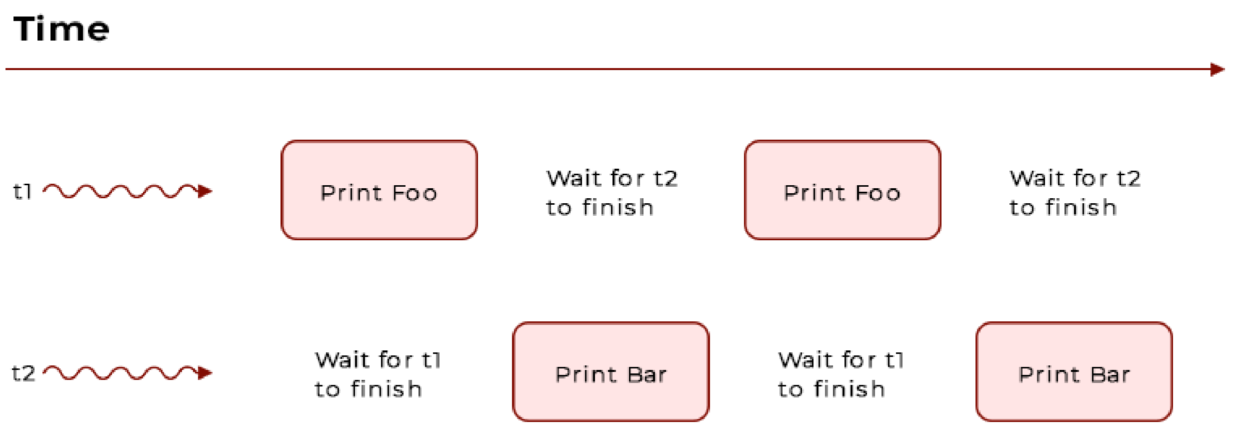
```
class PrintFooBar

    def PrintFoo()
        for i in 1..n do
        print "Foo"
        end
    end

    def PrintBar()
        for i in 1..n do
        print "Bar"
        end
    end
end
```

The two threads will run sequentially. You have to synchronize the two threads so that the functions PrintFoo() and PrintBar() are executed in an order.

The workflow of the program is shown below:



Workflow

Solution

We will solve this problem using two basic synchronization tools offered in Ruby: mutex and condition variable. Four private instances of the class are integer `n`, `mutex`, condition variable `cv` and a boolean variable `bar`. The class `Foobar` consists of two main methods `printFoo()` and `printBar()`.

```
class Foobar
    def initialize(n)
        @n = n
        @mutex = Mutex.new
        @bar = false
        @cv = ConditionVariable.new
    end

    def printFoo
    end

    def printBar
    end
end
```

In the `initialize()` method, `bar` is set to false. This ensures "Bar" is not printed before "Foo".

In `printFoo()`, the printing loop is iterated `n` (user input) number of times. In order to print “Foo” first, we will lock the printing operation in `synchronize` block. If `bar` is false then “Foo” is printed and it is set to true. Then the waiting threads are notified via `broadcast()`. If `bar` is true, then the `mutex` is acquired and other calling threads are blocked until the `mutex` is released.

```
def printFoo
    @mutex.synchronize do
        for i in 1..@n do
            if (@bar)
                @cv.wait(@mutex)
            end
            print "Foo"
            @bar = true
            @cv.broadcast
        end
    end
end
```

Similarly in `printBar()`, the printing loop is iterated `n` number of times and `mutex` is acquired for “Bar” to be printed. If `bar` is true then “Bar” will be printed otherwise the calling thread will go into `wait()`. Once "Bar" is printed, `bar` is set to false and waiting threads are notified via `broadcast()`.

```
def printBar
    @mutex.synchronize do
        for i in 1..@n do
            if (@bar != true)
                @cv.wait(@mutex)
            end
            puts "Bar"
            @bar = false
            @cv.broadcast
        end
    end
end
```

To test our code, We will create 2 threads; **t1** and **t2**. An object of `Foobar` is initialized with **5**. Both threads will be passed the same object of `Foobar`. **t1** calls `printFoo()` & **t2** calls `printBar()`

t1 calls `printFoo()` & **t2** calls `printBar()`.

```ruby
class Foobar

        def initialize(n)
                @n = n
                @mutex = Mutex.new
                @bar = false
                @cv = ConditionVariable.new
        end

        def printFoo
                @mutex.synchronize do
                        for i in 1..@n do
                                if (@bar)
                                        @cv.wait(@mutex)
                                end
                                print "Foo"
                                @bar = true
                                @cv.broadcast
                        end
                end
        end

        def printBar
                @mutex.synchronize do
                        for i in 1..@n do
                                if (@bar != true)
                                        @cv.wait(@mutex)
                                end
                                print "Bar"
                                @bar = false
                                @cv.broadcast
                        end
                end
        end
end

class Main
  foobar = Foobar.new(5)

  t1 =  Thread.new do
          foobar.printFoo
              end
  t2 =  Thread.new do
                      foobar.printBar
              end

  t2.join
  t1.join
end
```