

76% completed

Search Course

Multithreading in Java

Java's Memory Model

Interview Practice Problems

Bonus Questions

Ordered Printing

Printing Foo Bar n Times

Printing Number Series (Zero, Even, Odd)

Build a Molecule

Fizz Buzz Problem

Beyond the Interview

Java Concurrency

Practice Mock Interview

Java Multithreading for Senior Engineering Interviews / ... / Printing Foo Bar n Times

Printing Foo Bar n Times

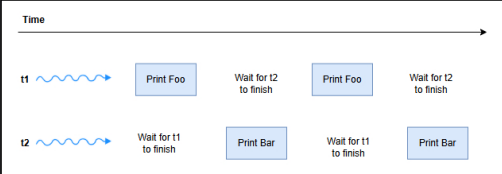
Learn how to execute threads in a specific order for a user specified number of iterations.

Problem Statement

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 {  
  
    public void PrintFoo() {  
        for (int i = 1; i <= n; i++){  
            System.out.print("Foo");  
        }  
    }  
  
    public void PrintBar() {  
        for (int i = 1; i <= n; i++){  
            System.out.print("Bar");  
        }  
    }  
}
```

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 is shown below:



Solution

We will solve this problem using the basic utilities of `wait()` and `notifyAll()` in Java. The basic structure of `FooBar` class is given below:

```
class FooBar {  
    private int n;  
    private int flag = 0;  
  
    public FooBar(int n) {  
        this.n = n;  
    }  
  
    public void foo() {  
    }  
  
    public void bar() {  
    }  
}
```

Two private instances of the class are integers `n`, and `flag`.

`n` is the user input that tells how many times "Foo" and "Bar" should be printed. `flag` is an integer based on which the words are printed. When the value of `flag` is 0, the word "Foo" will be printed and it will be incremented. This way "Bar" can be printed next. `flag` is initialized with 0 because the printing has to start with "Foo". The class consists of two methods `foo()` and `bar()` and their structures are given below:

```
public void foo() {  
    for (int i = 1; i <= n; i++) {  
        synchronized(this) {  
            while (flag == 1) {  
                try {  
                    this.wait();  
                }  
                catch (Exception e) {}  
            }  
            System.out.print("Foo");  
            flag = 1;  
            this.notifyAll();  
        }  
    }  
}
```

In `foo()`, a loop is iterated `n`(user input) number of times. For synchronization purpose, the printing operation is locked in `synchronize(this)` block. This is done to ensure proper sequence of printing. If `flag` is 0 then "Foo" is printed, then `flag` is set to 1 and any waiting threads are notified via `notifyAll()`. While the value of `flag` is 1, then `wait()` blocks the calling thread.

```

public void bar() {
    for (int i = 1; i <= n; i++) {
        synchronized(this) {
            while (flag == 0) {
                try {
                    this.wait();
                }
                catch (Exception e) {
                }
            }
            System.out.print("Bar");
            flag = 0;
            this.notifyAll();
        }
    }
}

```

Similarly in `bar()`, the loop is iterated `n` times. In every iteration, the while loop checks if the value of `flag` is 0. If it is, then it means it is not yet bar's turn to be printed and the calling thread will `wait()`. When the value of `flag` changes to 1, the waiting thread can resume execution. "Bar" is printed and `flag` is changed to 0 for "Foo" to be printed next. All the waiting threads are then notified via `notifyAll()` that this thread has finished its work.

We will create a new class `FooBarThread` that extends `Thread`. This enables us to run `FooBar` methods in separate threads concurrently. The class consists of a `FooBar` object along with a string `method` which holds the name of the function to be called. If `method` matches "foo" then `fooBar.foo()` is called. If `method` matches "bar", then `fooBar.bar()` is called.

```

class FooBarThread extends Thread {
    FooBar fooBar;
    String method;

    public FooBarThread(FooBar fooBar, String method){
        this.fooBar = fooBar;
        this.method = method;
    }

    public void run() {
        if ("foo".equals(method)) {
            fooBar.foo();
        }
        else if ("bar".equals(method)) {
            fooBar.bar();
        }
    }
}

```

To test our code, We will create two threads; `t1` and `t2`. An object of `FooBar` is initialized with `n`. Both threads will be passed the same object of `FooBar`. `t1` calls `foo()` & `t2` calls `bar()`.

The screenshot shows a Java IDE window titled 'Main.java' with the following code:

```

1 class FooBar {
2
3     private int n;
4     private int flag = 0;
5
6     public FooBar(int n) {
7         this.n = n;
8     }
9
10    public void foo() {
11
12        for (int i = 1; i <= n; i++) {

```

[← Back lesson](#)

☒ Completed

[Next →](#)

Ordered Printing

Printing Number Series (Zero, Even, Odd)