# Lab: Asynchronous Programming

Problems for exercises and homework for the ["Java Advanced" course @ SoftUni](https://softuni.bg/courses/java-advanced).

# Part I: Single and Multi-Threading

## Single Thread

Create a **task** that prints the numbers from 1 to 10. **Start a thread** executing the task.

**Optional:** Add System.exit(1) at the end of your program.

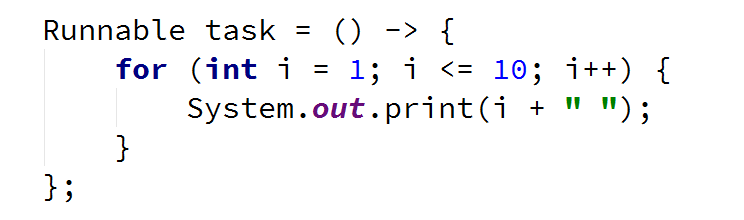
**Optional:** Experiment with thread.join()

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| *no input* | 1 2 3 4 5 6 7 8 9 10 |

### Solution

Create a new Runnable that will define the code for the task:



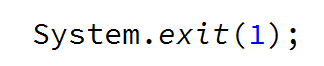
Create a new Thread that will execute the task



Start the thread:



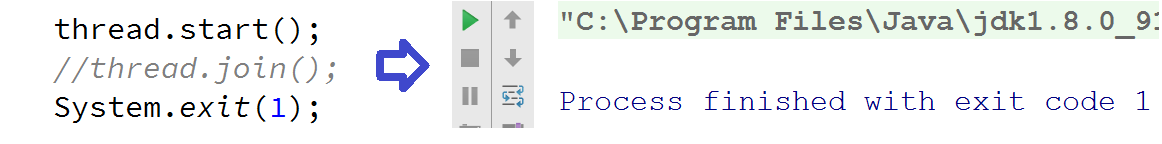
**Optional:** Add System.exit(1) at the end of your program



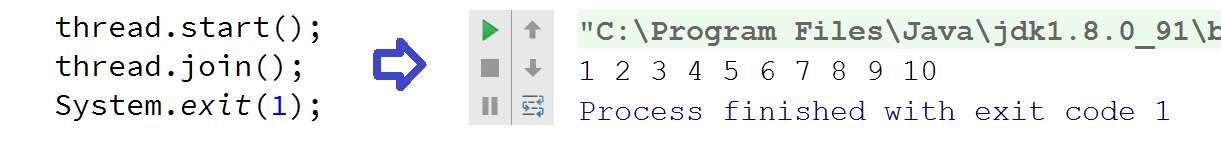
**Optional:** Experiment with thread.join()



**Example:** If you include System.exit(1) and in the same time omit thread.join(), it is **possible** that the **main thread closes the additional thread** before the additional thread is done with its task



**Example:** By including thread.join() it is guaranteed that the **main thread will wait for the thread it has started** (thread.join() blocks the calling thread)



## Multi-Thread

Create a task that prints the numbers from 1 to 10. **Start 5 threads** executing the same task.

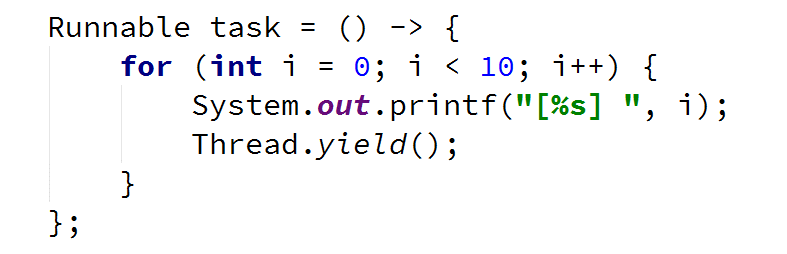
After each printing, add Thread.yield() statement. **Join all threads**.

### Examples

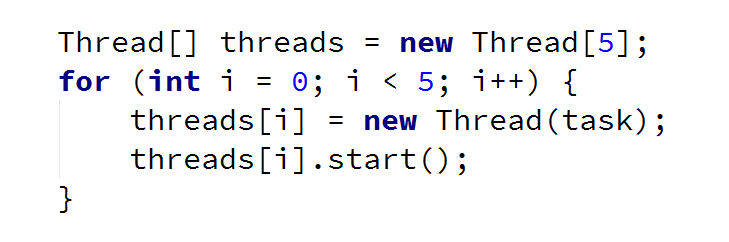
|  |  |
| --- | --- |
| **Input** | **Output** |
| *no input* | *(Output can vary)*  [0] [0] [0] [0] [0] [1] [1] [1] [2] [3] [2] [1] … |

### Solution

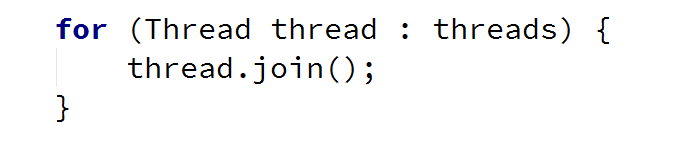
Create a new Runnable which prints the numbers and yields after each print. Thread.yield() will make the effect of thread switching more obvious.



Create an array for all 5 threads and for each of them start a new task:



Join all 5 threads:



## Responsive UI

Create a program that prints the **primes from 0 to N**. Implement a **responsive UI**, e.g. user can stop the program at any time.

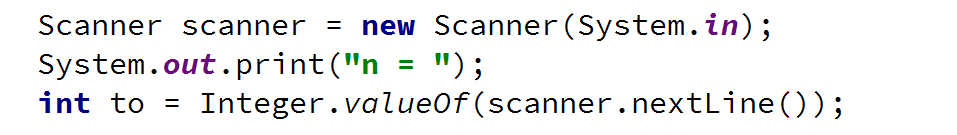
If stopped, show appropriate message

### Examples

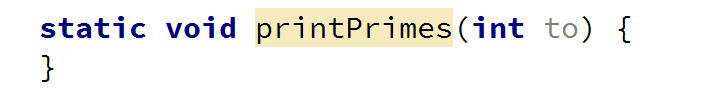
|  |  |
| --- | --- |
| **Input** | **Output** |
| 13 | [2, 3, 5, 7, 11]...  5 primes calculated. |
| 9999999  stop | [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]...  169922 primes calculated. |

### Solution

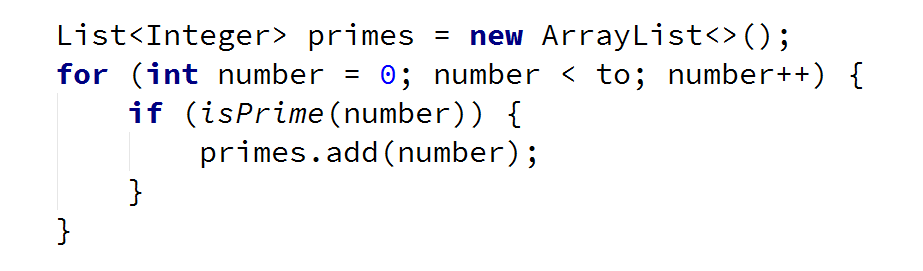
Read N, the upper bound:



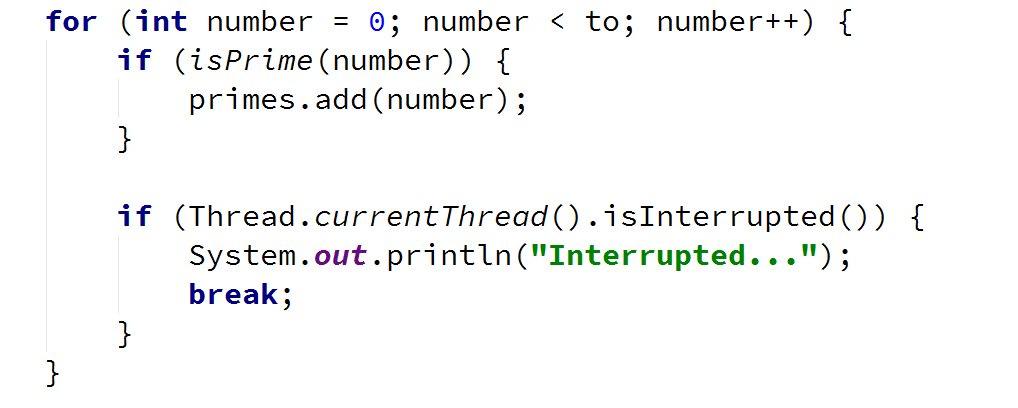
Create a method printPrimes() which you will use as a task:



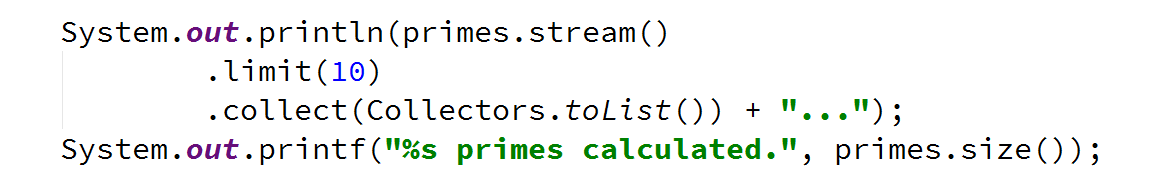
Create a List<Integer> for storing all prime numbers:



Inside the for loop, define a condition for thread interruption:



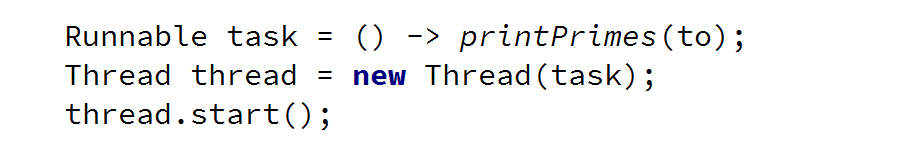
Print some of the primes and the count of all primes you have discovered:



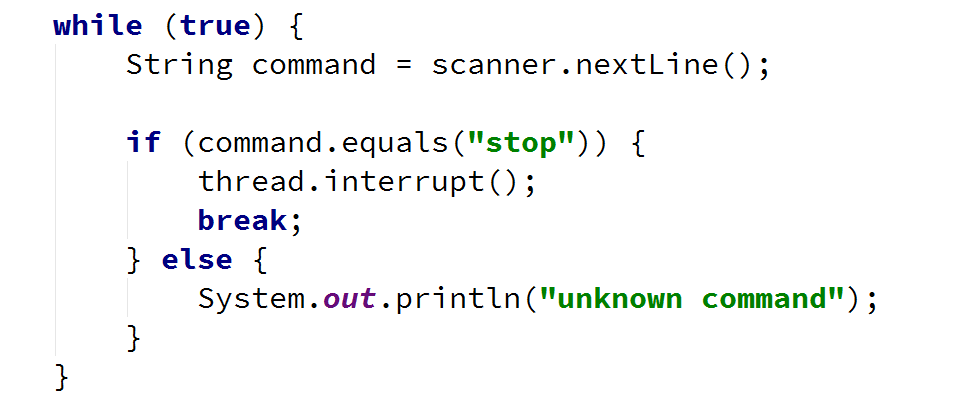
Implement the method isPrime() yourself. It should evaluate a single number:



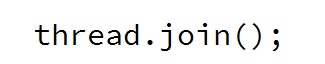
In the main(), create a new task with printPrimes() and start it:



Create a loop for user input:



* Wait for the thread to finish execution:



## Benchmarking

Test every number in the range [0...N] if it is prime or not. Spread the calculation over 2 or 4 threads.

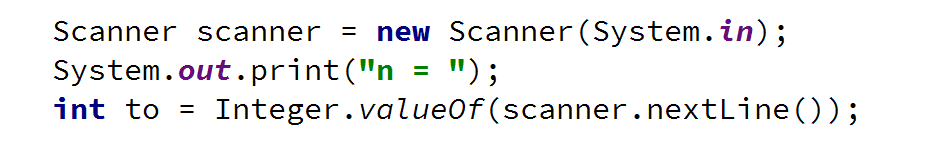
Benchmark and compare the difference over one thread. Benchmark both efficient and inefficient isPrime().

### Examples

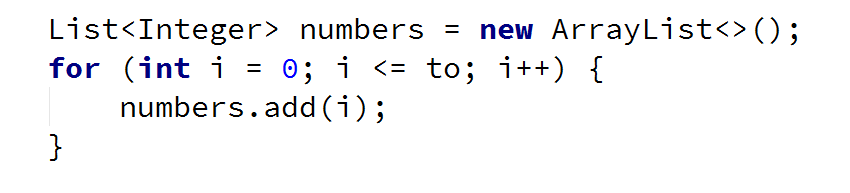
|  |  |
| --- | --- |
| **Input** | **Output** |
| 1000 | *(Output guaranteed to vary)*  Execution time: 184503539 |
| 999999 | *(Output guaranteed to vary)*  Execution time: 3274639906 |

### Solution

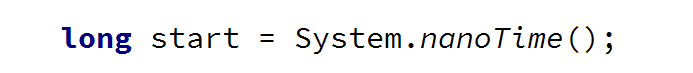
Read N, the upper bound



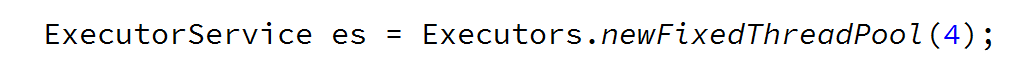
Create a List<Integer> with all numbers



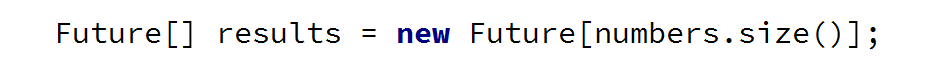
Start a clock for benchmarking:



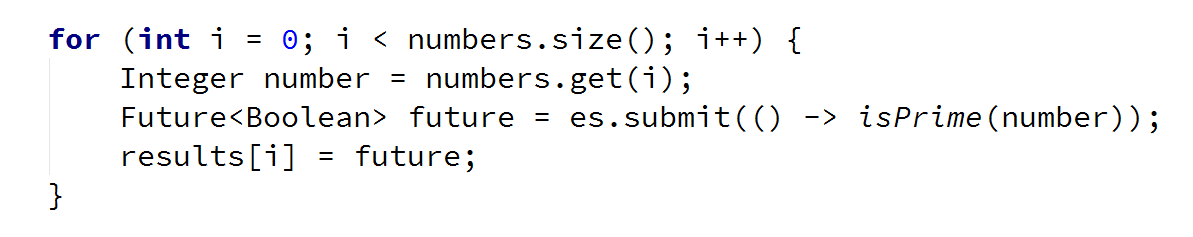
Create a new ExecutorService with a fixed thread pool



Create a Future[] with the size of all numbers



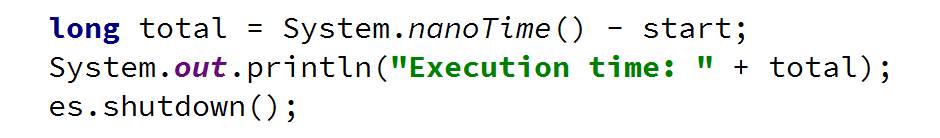
Test each number



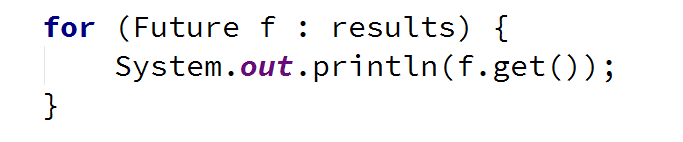
Await all tasks to finish



Stop the benchmark and **print the results**. Make sure to **shut down** the executor service



If you want the result for each number, you can get it from the Future array



# Part II: Resource Sharing

## Transactions

Create a simple BankAccount class with the following characteristics:

* Properties:
  + Integer balance
* Methods:
  + void deposit(int sum)

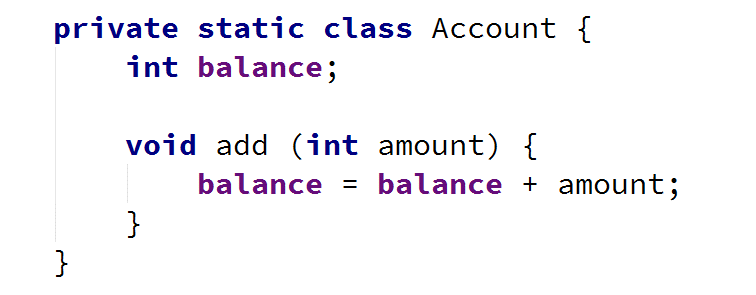
Create a **multi-threaded program** that simulates 100 transactions, each deposing 100 times 1 to the balance.

### Examples

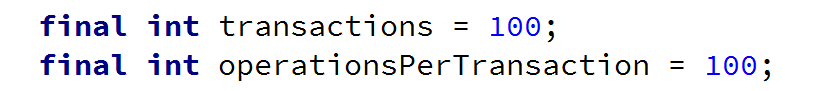
|  |  |
| --- | --- |
| **Input** | **Output** |
| *no input* | *(Output should vary)*  9559 |
| *no input* | *(Output should vary)*  9905 |

### Solution

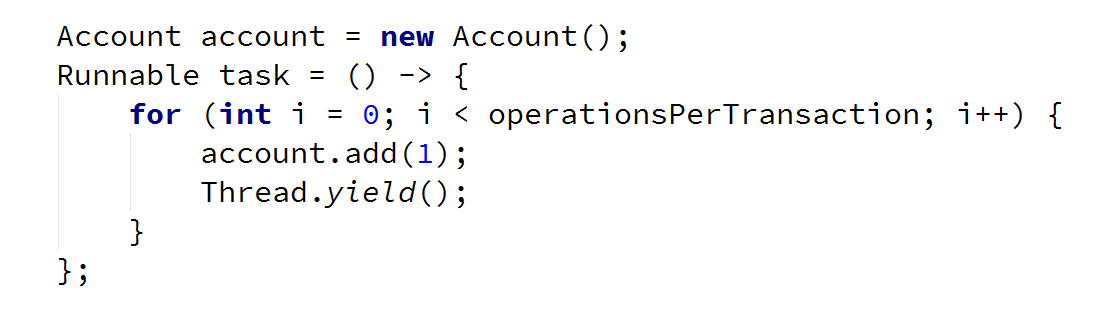
Create the class



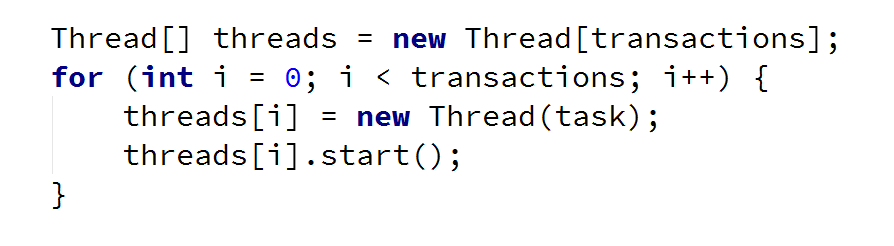
Create constants for the **number of transactions** and for **number of operations per transaction**



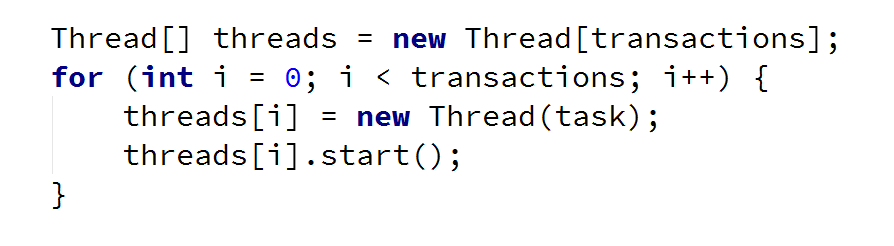
Create an **instance of the class** and a **task**



Create a new thread for each transaction



**Join** all threads



**Print** the results



Start the program multiple times and observe the results

## Thread Safe Transactions

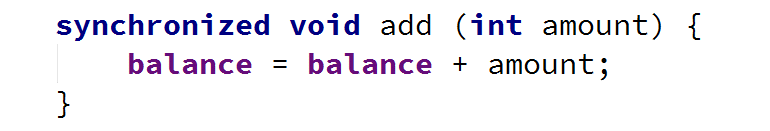
Make the previous application **thread safe**, e.g. you should get the **same result every time**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| *no input* | *(Output should not vary)*  10000 |
| *no input* | *(Output should not vary)*  10000 |

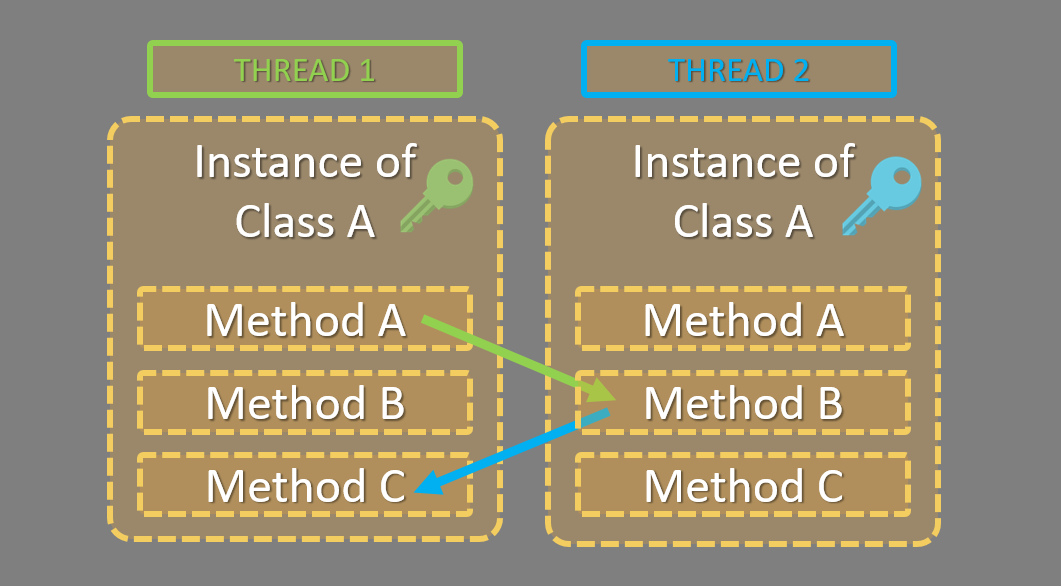
### Solution

Make the method add() synchronized



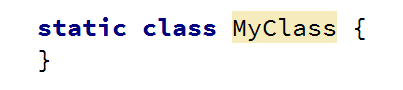
## \* Deadlock

Reproduce the following **deadlock scenario**:

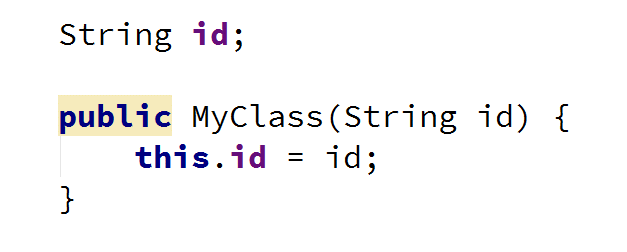


### Solution

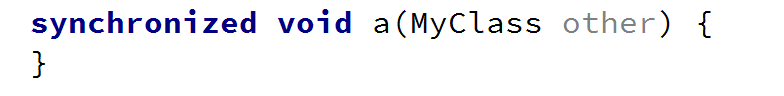
Create a class that will hold methods A, B and C



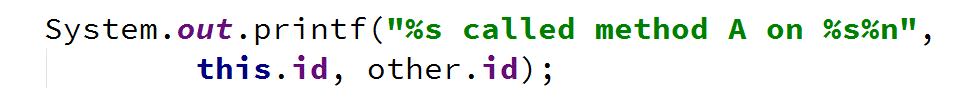
Add a property id and a constructor, setting the property



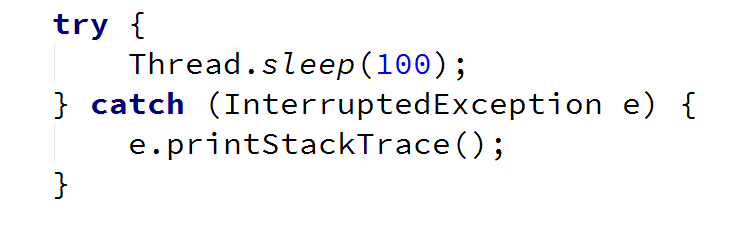
Create a methoda(), which should take a reference to the other instance of the class. Make sure it is declared with the synchronized keyword



Print a message, that the method was called



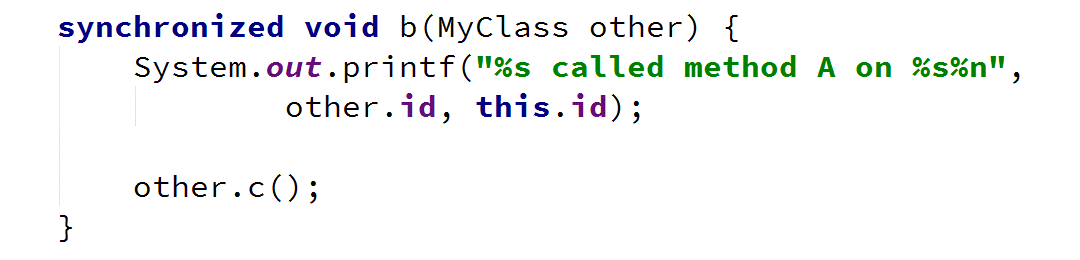
Sleep the thread for some milliseconds to ensure that the two methods will be called at the same time by the threads



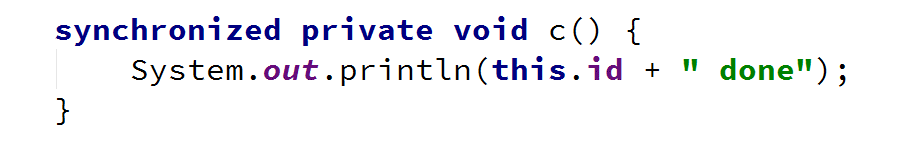
Call the b() method of the other instance and pass a reference to the current object



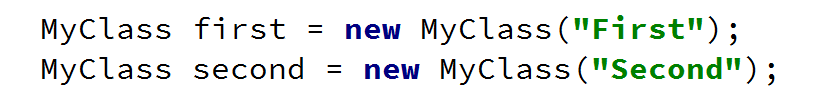
Create the b() method, which should also print a message and call the other objects c() method



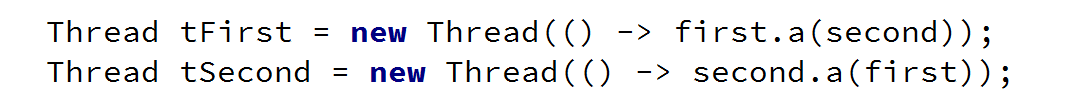
Create the c() method



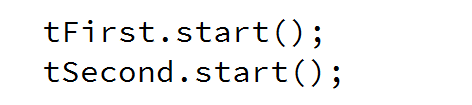
In the main method, create two instances of the class



Create two threads that start a new task



Start the threads



You should get a deadlock

