CSCI 4200

Design of Operating System

Assignment 3

100 Points

Instruction: Use any programming language of your choice. Make sure to put comments at each line of our code. If the solution does not have any comments, you will be deducted 25% of your grade.

Question 1: 20 Points

Suppose we have a class:

```
public class Foo {
  public void first() { print("first"); }
  public void second() { print("second"); }
  public void third() { print("third"); }
}
```

The same instance of Foo will be passed to three different threads. Thread A will call first(), thread B will call second(), and thread C will call third(). Design a mechanism and modify the program to ensure that second() is executed after first(), and third() is executed after second().

Note:

We do not know how the threads will be scheduled in the operating system, even though the numbers in the input seem to imply the ordering. The input format you see is mainly to ensure our tests' comprehensiveness.

Example 1:

Input: nums = [1,2,3]

Output: "firstsecondthird"

Explanation: There are three threads being fired asynchronously. The input [1,2,3] means thread A calls first(), thread B calls second(), and thread C calls third(). "firstsecondthird" is the correct output.

Example 2:

Input: nums = [1,3,2]

Output: "firstsecondthird"Explanation: The input [1,3,2] means thread A calls first(), thread B calls third(), and thread C calls second(). "firstsecondthird" is the correct output.

Use the following starter code:

Java

```
class Foo {

    public Foo() {

    }

    public void first(Runnable printFirst) throws InterruptedException {

        // printFirst.run() outputs "first". Do not change or remove this line.
        printFirst.run();
    }

    public void second(Runnable printSecond) throws InterruptedException {

        // printSecond.run() outputs "second". Do not change or remove this line.
        printSecond.run();
    }

    public void third(Runnable printThird) throws InterruptedException {

        // printThird.run() outputs "third". Do not change or remove this line.
        printThird.run();
    }
}
```

C++

```
class Foo {
public:
    Foo() {

    }

    void first(function<void()> printFirst) {

        // printFirst() outputs "first". Do not change or remove this line.
        printFirst();
    }
```

```
    void second(function<void()> printSecond) {

        // printSecond() outputs "second". Do not change or remove this line.
        printSecond();
    }

    void third(function<void()> printThird) {

        // printThird() outputs "third". Do not change or remove this line.
        printThird();
    }
};
```

Python:

```
class Foo:
    def __init__(self):
        pass


    def first(self, printFirst: 'Callable[[], None]') -> None:

        # printFirst() outputs "first". Do not change or remove this line.
        printFirst()


    def second(self, printSecond: 'Callable[[], None]') -> None:

        # printSecond() outputs "second". Do not change or remove this line.
        printSecond()


    def third(self, printThird: 'Callable[[], None]') -> None:

        # printThird() outputs "third". Do not change or remove this line.
        printThird()
```

C#:

```
public class Foo {

    public Foo() {

    }
```

```
    public void First(Action printFirst) {

        // printFirst() outputs "first". Do not change or remove this line.
        printFirst();
    }

    public void Second(Action printSecond) {

        // printSecond() outputs "second". Do not change or remove this line.
        printSecond();
    }

    public void Third(Action printThird) {

        // printThird() outputs "third". Do not change or remove this line.
        printThird();
    }
}
```

Question 2: 20 Points

You have the four functions:

printFizz that prints the word "fizz" to the console,

printBuzz that prints the word "buzz" to the console,

printFizzBuzz that prints the word "fizzbuzz" to the console, and

printNumber that prints a given integer to the console.

You are given an instance of the class FizzBuzz that has four functions: fizz, buzz, fizzbuzz and number. The same instance of FizzBuzz will be passed to four different threads:

Thread A: calls fizz() that should output the word "fizz".

Thread B: calls buzz() that should output the word "buzz".

Thread C: calls fizzbuzz() that should output the word "fizzbuzz".

Thread D: calls number() that should only output the integers.

Modify the given class to output the series [1, 2, "fizz", 4, "buzz", ...] where the ith token (1-indexed) of the series is:

"fizzbuzz" if i is divisible by 3 and 5,

"fizz" if i is divisible by 3 and not 5,

"buzz" if i is divisible by 5 and not 3, or

i if i is not divisible by 3 or 5.

Implement the FizzBuzz class:

FizzBuzz(int n) Initializes the object with the number n that represents the length of the sequence that should be printed.

void fizz(printFizz) Calls printFizz to output "fizz".

void buzz(printBuzz) Calls printBuzz to output "buzz".

void fizzbuzz(printFizzBuzz) Calls printFizzBuzz to output "fizzbuzz".

void number(printNumber) Calls printnumber to output the numbers.

Example 1:

Input: n = 15

Output: [1,2,"fizz",4,"buzz","fizz",7,8,"fizz","buzz",11,"fizz",13,14,"fizzbuzz"]

Example 2:

Input: n = 5

Output: [1,2,"fizz",4,"buzz"]

Constraints:

1 <= n <= 50

Question 3. 20 Points

Suppose you are given the following code:

```
class FooBar {
  public void foo() {
    for (int i = 0; i < n; i++) {
      print("foo");
    }
  }
  public void bar() {
    for (int i = 0; i < n; i++) {
      print("bar");
    }
  }
```

}

The same instance of FooBar will be passed to two different threads:

thread A will call foo(), while

thread B will call bar().

Modify the given program to output "foobar" n times.

Example 1:

Input: n = 1

Output: "foobar"

Explanation: There are two threads being fired asynchronously. One of them calls foo(), while the other calls bar().

"foobar" is being output 1 time.

Example 2:

Input: n = 2

Output: "foobarfoobar"

Explanation: "foobar" is being output 2 times.

Constraints:

1 <= n <= 1000

Question 4: 20 Points

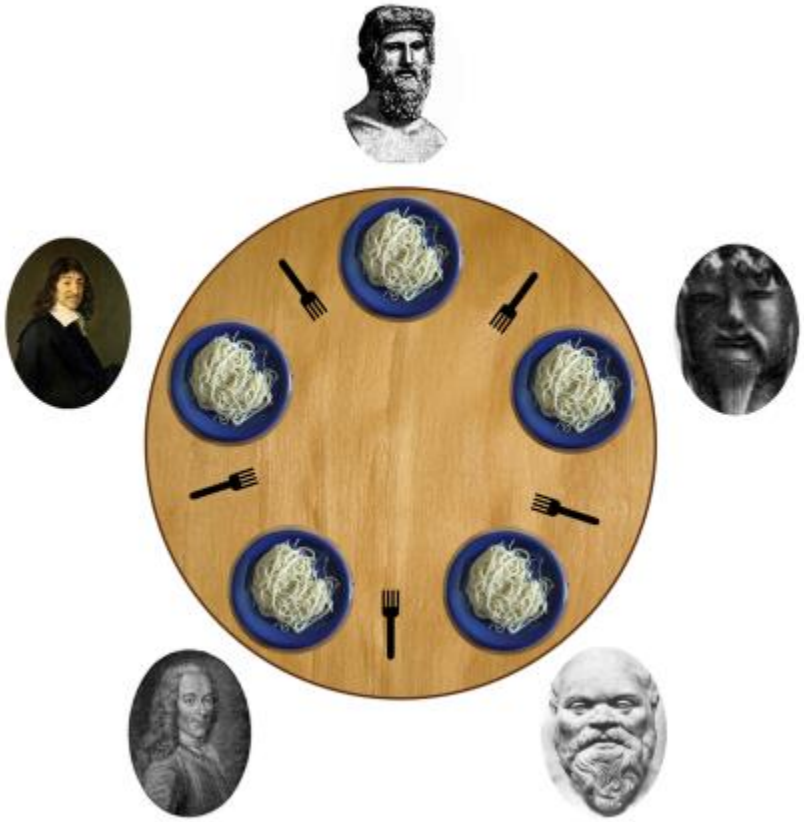
Implement the Dining Philosophers problem:

Five silent philosophers sit at a round table with bowls of spaghetti. Forks are placed between each pair of adjacent philosophers.

Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when they have both left and right forks. Each fork can be held by only one philosopher and so a philosopher can use the fork only if it is not being used by another philosopher. After an individual philosopher finishes eating, they need to put down both forks so that the forks become available to others. A philosopher can take the fork on their right or the one on their left as they become available, but cannot start eating before getting both forks.

Eating is not limited by the remaining amounts of spaghetti or stomach space; an infinite supply and an infinite demand are assumed.

Design a discipline of behaviour (a concurrent algorithm) such that no philosopher will starve; i.e., each can forever continue to alternate between eating and thinking, assuming that no philosopher can know when others may want to eat or think.



- *The philosophers' ids are numbered from 0 to 4 in a clockwise order. Implement the function void wantsToEat(philosopher, pickLeftFork, pickRightFork, eat, putLeftFork, putRightFork) where:*
- *philosopher is the id of the philosopher who wants to eat.*
- *pickLeftFork and pickRightFork are functions you can call to pick the corresponding forks of that philosopher.*
- *eat is a function you can call to let the philosopher eat once he has picked both forks.*
- *putLeftFork and putRightFork are functions you can call to put down the corresponding forks of that philosopher.*
- *The philosophers are assumed to be thinking as long as they are not asking to eat (the function is not being called with their number).*

- *Five threads, each representing a philosopher, will simultaneously use one object of your class to simulate the process. The function may be called for the same philosopher more than once, even before the last call ends.*

Example 1:

Input: n = 1

Output:
[[4,2,1],[4,1,1],[0,1,1],[2,2,1],[2,1,1],[2,0,3],[2,1,2],[2,2,2],[4,0,3],[4,1,2],[0,2,1],[4,2,2],[3,2,1],[3,1,1],[0,0,3],[0,1,2],[0,2,2],[1,2,1],[1,1,1],[3,0,3],[3,1,2],[3,2,2],[1,0,3],[1,1,2],[1,2,2]]

Explanation:

n is the number of times each philosopher will call the function.

The output array describes the calls you made to the functions controlling the forks and the eat function, its format is:

output[i] = [a, b, c] (three integers)

- a is the id of a philosopher.

- b specifies the fork: {1 : left, 2 : right}.

- c specifies the operation: {1 : pick, 2 : put, 3 : eat}.

Constraints:

1 <= n <= 60

Starter Code:

```
class DiningPhilosophers {

    public DiningPhilosophers() {

    }

    // call the run() method of any runnable to execute its code
    public void wantsToEat(int philosopher,
                           Runnable pickLeftFork,
                           Runnable pickRightFork,
                           Runnable eat,
                           Runnable putLeftFork,
                           Runnable putRightFork) throws InterruptedException {

    }
}
```

Question 5: 20 Points

Create a multithreaded program in any programming language that simulates a bank account with deposits and withdrawals. Multiple threads should be able to access the account concurrently, and the program should handle potential race conditions**.**