Implement a stack as a linked list in which the push, pop, and isEmpty methods can be safely accessed from multiple threads

```
import threading
class Node:
 def __init__(self, value):
    self.value = value
    self.next = None
class Stack:
  def __init__(self):
    self.top = None
    self.lock = threading.Lock()
  def push(self, value):
    with self.lock:
      node = Node(value)
      node.next = self.top
      self.top = node
 def pop(self):
    with self.lock:
      if self.top is None:
        return None
      value = self.top.value
      self.top = self.top.next
      return value
 def isEmpty(self):
    with self.lock:
      return self.top is None
def push_values(stack, values):
    for value in values:
      stack.push(value)
def pop_values(stack, num_values):
    for i in range(num_values):
      value = stack.pop()
      if value is None:
        print("Stack is empty")
        print("Popped value:", value)
stack = Stack()
push_thread = threading.Thread(target=push_values, args=(stack, [1, 2, 3, 4, 5])
pop thread = threading.Thread(target=pop values, args=(stack, 5))
push_thread.start()
pop_thread.start()
push_thread.join()
pop_thread.join()
print("Is stack empty?", stack.isEmpty())
```

Popped value: 5
Popped value: 4
Popped value: 3
Popped value: 2
Popped value: 1
Is stack empty? True

Implement a Queue class whose add and remove methods are synchronized. Supply one thread, called the producer, which keeps inserting strings into the queue as long as there are fewer than ten elements in it. When the queue gets too full, the thread waits. As sample strings, simply use time stamps new Date().toString(). Supply a sec ond thread, called the consumer, that keeps removing and printing strings from the queue as long as the queue is not empty. When the queue is empty, the thread waits. Both the consumer and producer threads should run for 100 iterations.

```
import threading
import queue
import time
class SynchronizedQueue:
  def __init__(self):
    self.queue = queue.Queue()
    self.max_size = 10
    self.lock = threading.Lock()
    self.condition = threading.Condition(self.lock)
  def add(self, item):
    with self.lock:
      while self.queue.qsize() >= self.max_size:
        self.condition.wait()
        self.queue.put(item)
        self.condition.notify_all()
  def remove(self):
    with self.lock:
      while self.queue.qsize() == 0:
        self.condition.wait()
      item = self.queue.get()
      self.condition.notify all()
      return item
def producer(queue):
  for i in range(100):
    while queue.queue.qsize() < queue.max_size:</pre>
      item = time.strftime("%H:%M:%S", time.localtime())
      queue.add(item)
      print(f"Producer added item: {item}")
    time.sleep(1)
```

```
def consumer(queue):
    for i in range(100):
        while not queue.queue.empty():
            item = queue.remove()
            print(f"Consumer removed item: {item}")
            time.sleep(1)

if __name__ == "__main__":
    synchronized_queue = SynchronizedQueue()
    producer_thread = threading.Thread(target=producer, args=(synchronized_queue,)
    consumer_thread.start()
    consumer_thread.start()
    producer_thread.start()
    producer_thread.start()
    consumer_thread.join()
    consumer_thread.join()
```

```
Streaming output truncated to the last 5000 lines.
Producer added item: 14:50:42
```

```
Producer added item: 14:50:42
```

```
Producer added item: 14:50:42
```

```
Producer added item: 14:50:42
```

```
Producer added item: 14:50:42
```

```
Producer added item: 14:50:42
```

```
Producer added item: 14:50:42
```

```
Producer added item: 14:50:42
```

```
Producer added item: 14:50:42
```

```
Producer added item: 14:50:42
```

```
Producer added item: 14:50:42
         added item.
```

```
Producer added item: 14:50:42
         5055c
```

```
Producer added item: 14:50:42
```

```
Producer added item: 14:50:42
```

```
Producer added item: 14:50:42
```

```
rioducei added item:
Producer added item: 14:50:42
Producer added item: 14:50:43
```

```
Producer added Item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
         added item.
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
rioducei added item:
Producer added item: 14:50:43
```

```
Producer added Item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
         added item.
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
rioducei added item:
Producer added item: 14:50:43
```

```
Producer added Item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
         added item.
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
rioducei added item:
Producer added item: 14:50:43
```

```
Producer added Item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
         added item.
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
Producer added item: 14:50:43
```

```
rioducei added item:
Producer added item: 14:50:43
Producer added item: 14:50:44
```

```
Producer added Item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
KeyboardInterrupt
                                           Traceback (most recent call
last)
<ipython-input-3-1e6642be8a85> in <cell line: 37>()
          producer thread.start()
     42
          consumer thread.start()
 --> 43
          producer thread.join()
     44
          consumer thread.join()
                                 🕽 1 frames -
/usr/lib/python3.9/threading.py in wait for tstate lock(self, block,
timeout)
   1078
   1079
                    if lock.acquire(block, timeout):
   1081
                        lock.release()
   1082
                        self. stop()
KeyboardInterrupt:
 SEARCH STACK OVERFLOW
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
         added item.
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
PIOUUCEI auueu ILEM:
Producer added item: 14:50:44
```

```
Producer added Item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
         added item.
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
PIOUUCEI auueu ILEM:
Producer added item: 14:50:44
```

```
Producer added Item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
         added item.
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
Producer added item: 14:50:44
```

```
PIOUUCEI auueu ILEM:
Producer added item: 14:50:44
```

```
Producer added Item: 14:50:44
```

```
Producer added item: 14:50:44
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
         added item.
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
PIOUUCEI auueu ILEM:
Producer added item: 14:50:45
```

```
Producer added Item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
         added item.
```

```
Producer added item: 14:50:45
```

```
Producer added item: 14:50:45
```

N philosophers sit at a table with a plate of spaghetti in front of them and a fork on their right and one on their left. To eat spaghetti, a philosopher needs both forks close together. Each philosopher is continuously engaged in a sequence of 3 activities: meditating, trying to acquire forks and eating. Write a program that activates N philosopher threads that execute the described loop 100 times. Meditation and the phase where the philosopher eats must be implemented with a variable delay (use for example the sleep call and the rand() function)

```
import threading
import random
import time
class Philosopher(threading.Thread):
    def __init__(self, name, left_fork, right_fork):
        super().__init__(name=name)
        self.left_fork = left_fork
        self.right_fork = right_fork
        self.eating_count = 0
```

```
def run(self):
    for i in range(100):
      self.mediate()
      self.acquire_forks()
      self.eat()
      self.release_forks()
  def mediate(self):
    print(f"{self.name} is meditating")
    time.sleep(random.uniform(1, 5))
  def acquire forks(self):
    print(f"{self.name} is trying to acquire forks")
    while True:
      if self.left_fork.acquire(blocking=False):
        if self.right fork.acquire(blocking=False):
          print(f"{self.name} acquired forks")
          return
        else:
          self.left_fork.release()
          time.sleep(random.uniform(0, 1))
  def eat(self):
    print(f"{self.name} is eating")
    time.sleep(random.uniform(1, 5))
    self.eating count += 1
  def release forks(self):
    print(f"{self.name} is releasing forks")
    self.left_fork.release()
    self.right fork.release()
if __name__ == "__main__":
  n = 5 \# number of philosophers
  forks = [threading.Lock() for i in range(n)]
  philosophers = []
  for i in range(n):
    left_fork = forks[i]
    right fork = forks[(i + 1) % n]
    philosopher = Philosopher(f"Philosopher {i}", left_fork, right_fork)
    philosophers.append(philosopher)
    philosopher.start()
  for philosopher in philosophers:
    philosopher.join()
  print("All philosophers have eaten")
```

```
Streaming output truncated to the last 5000 lines.

Producer added item: 14:54:51

Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
         added item.
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
PIOUUCEI auueu ILEM:
Producer added item: 14:54:51
```

```
Producer added Item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
         added item.
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
PIOUUCEI auueu ILEM:
Producer added item: 14:54:51
```

```
Producer added Item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
```

```
Producer added item: 14:54:51
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
         added item.
```

```
Producer added item: 14:54:52
         5055c
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
PIOUUCEI auueu ILEM:
Producer added item: 14:54:52
```

```
Producer added Item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
         added item.
```

```
Producer added item: 14:54:52
         hobbe
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
PIOUUCEI auueu ILEM:
Producer added item: 14:54:52
```

```
Producer added Item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
```

```
Producer added item: 14:54:52
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
Philosopher 3 is trying to acquire forks
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
Philosopher 4 is releasing forks
Philosopher 4 is meditatingPhilosopher 0 acquired forks
Producer added item: 14:54:53
Producer added item: 14:54:53
Producer added item: 14:54:53
Philosopher 0 is eatingProducer added item: 14:54:53
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
         added item.
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
PIOUUCEI auueu ILEM:
Producer added item: 14:54:53
```

```
Producer added Item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
                                           Traceback (most recent call
KeyboardInterrupt
last)
<ipython-input-4-2c75245da756> in <cell line: 37>()
            philosopher.start()
          for philosopher in philosophers:
            philosopher.join()
          print("All philosophers have eaten")
     49
                                1 frames
/usr/lib/python3.9/threading.py in _wait_for_tstate_lock(self, block,
timeout)
  1078
   1079
                try:
-> 1080
                    if lock.acquire(block, timeout):
  1081
                        lock.release()
   1082
                        self. stop()
KeyboardInterrupt:
 SEARCH STACK OVERFLOW
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
PIOUUCEI auueu ILEM:
Producer added item: 14:54:53
```

```
Producer added Item: 14:54:53
```

```
Producer added item: 14:54:53
```

```
Producer added item: 14:54:53
```

Reader-Writer Problem: This is a classic problem that demonstrates the use of synchronization in Java. The goal is to have multiple readers reading a shared resource simultaneously, while a writer is able to modify the resource. The challenge is to ensure that readers do not interfere with each other and that the writer has exclusive access to the resource when making modifications.

```
import threading
import time
class ReaderWriter():
 def __init__(self):
    self.rd = threading.Semaphore()
    self.wrt = threading.Semaphore()
    self.readCount = 0
 def reader(self):
    while True:
      self.rd.acquire()
      self.readCount+=1
      if self.readCount == 1:
        self.wrt.acquire()
        self.rd.release()
        print(f"Reader {self.readCount} is reading")
        self.rd.acquire()
        self.readCount-=1
      if self.readCount == 0:
       self.wrt.release()
      self.rd.release()
      time.sleep(3)
 def writer(self):
    while True:
      self.wrt.acquire()
      print("Wrting data....")
      print("-"*20)
      self.wrt.release()
     time.sleep(3)
 def main(self):
    t1 = threading.Thread(target = self.reader)
    t1.start()
    t2 = threading.Thread(target = self.writer)
    t2.start()
    t3 = threading.Thread(target = self.reader)
    t3.start()
    t4 = threading.Thread(target = self.reader)
    t4.start()
    t6 = threading.Thread(target = self.writer)
    t6.start()
```

```
t5 = threading. Thread(target = self.reader)
   t5.start()
if name ==" main ":
  c = ReaderWriter()
  c.main()
    Producer added item: 15:00:39
    Producer added item: 15:00:39
```

Draducar added item 15:00:20

```
Producer added item: 15:00:39
```

Sleeping Barber Problem: This problem is used to demonstrate the use of synchronization and inter-thread communication in Java. The goal is to model the behavior of a barber shop where customers arrive to get haircuts and the barber is responsible for cutting their hair. The challenge is to ensure that customers are served in the order in which they arrive, and that the barber does not start cutting hair until a customer is available.

```
import threading
import time
import random
MAX CUSTOMERS = 2
waiting_room = []
barber sleeping = threading.Event()
class Customer:
  def __init__(self, id):
    self.id = id
def barber():
    while True:
      print("Barber falls asleep")
      barber_sleeping.wait()
      while len(waiting room) > 0:
        customer = waiting_room[0]
        waiting room.remove(customer)
        print(f"Barber is cutting hair of customer {customer.id}")
        time.sleep(random.randint(1, 3))
        print(f"Customer {customer.id} leaves the barber shop")
      else:
        print("No more customers in the queue, barber goes back to sleep")
        barber_sleeping.clear()
def customer_arrives():
    id = 0
    while id < MAX_CUSTOMERS:
      time.sleep(random.randint(1, 4))
      customer = Customer(id)
      print(f"Customer {customer.id} arrives")
```

```
if len(waiting_room) < MAX_CUSTOMERS:</pre>
        waiting_room.append(customer)
        print(f"Customer {customer.id} takes a seat in the waiting room")
      if barber sleeping.is set():
        barber_sleeping.clear()
        print("Barber wakes up")
      else:
        print(f"Waiting room is full, customer {customer.id} leaves")
      id += 1
      print("All customers have arrived, shop is closing")
      barber_sleeping.set()
barber_thread = threading.Thread(target=barber, daemon=True)
customer thread = threading.Thread(target=customer arrives, daemon=True)
barber thread.start()
customer_thread.start()
time.sleep(5)
```

Producer added item: 15:04:37 Producer added item: 15:04:37

```
Producer added item: 15:04:37
```

Write a python program to Print alternate numbers using 2 Threads. Implement using wait and notify construct.

```
import threading
class AlternateNumbers:
  def __init__(self, n):
    self.n = n
    self.lock = threading.Lock()
    self.cond = threading.Condition(self.lock)
    self.current_number = 1
  def print number(self, is even):
    with self.lock:
      while self.current_number <= self.n:</pre>
        if is even and self.current number % 2 == 1:
          self.cond.wait()
        elif not is even and self.current number % 2 == 0:
          self.cond.wait()
        print(f"{threading.current_thread().name}: {self.current_number}")
        self.current number += 1
        self.cond.notify_all()
n = 10
alt nums = AlternateNumbers(n)
t1 = threading.Thread(target=alt_nums.print_number, args=(False,), name="Odd_Thr
```

```
t2 = threading.Thread(target=alt_nums.print_number, args=(True,), name="Even_Thr
t1.start()
t2.start()
t1.join()
t2.join()
```

```
Producer added item: 15:06:59
```

```
Producer added item: 15:06:59
```

Write a python program to implement banking account with necessary function. Ensure both winthdrawl and deposit can be carried out safely by employing concurrenct control.

```
import threading
class BankAccount:
  def init (self, balance=0):
    self.balance = balance
    self.lock = threading.Lock()
  def deposit(self, amount):
    with self.lock:
      self.balance += amount
      print(f"{threading.current_thread().name} deposited {amount} rupees")
      print(f"New balance is {self.balance} rupees\n")
  def withdraw(self, amount):
    with self.lock:
      if self.balance >= amount:
        self.balance -= amount
        print(f"{threading.current_thread().name} withdrew {amount} rupees")
        print(f"New balance is {self.balance} rupees\n")
      else:
        print(f"{threading.current_thread().name} tried to withdraw {amount}rupe
        print(f"Current balance is {self.balance} rupees\n")
bank account = BankAccount()
t1 = threading.Thread(target=bank_account.deposit, args=(1000,), name="Thread-1"
t2 = threading.Thread(target=bank_account.withdraw, args=(500,), name="Thread-2"
t3 = threading.Thread(target=bank_account.deposit, args=(200,), name="Thread-3")
t4 = threading.Thread(target=bank_account.withdraw, args=(300,), name="Thread-4"
t1.start()
t2.start()
t3.start()
t4.start()
t1.join()
t2.join()
t3.join()
t4.join()
```

Producer added item: 15:08:31 Producer added item: 15:08:31

01/04/23, 8:39 PM Week8.ipynb - Colaboratory

Producer added item: 15:08:31 Producer added item: 15:08:31 Producer added item: 15:08:31

Philosopher 1 is trying to acquire forks Philosopher 1 acquired forks

Philosopher 1 is eating

Colab paid products - Cancel contracts here

✓ 0s completed at 8:38 PM