package threadtest;
public class StackImpl {

    private int stackData[];
    private int stackTop;
    private int capacity;
    public StackImpl (int capacity)
    {
        stackData = new int[capacity];
        this.capacity = capacity;
        this.stackTop = -1;
    }

    synchronized public int pop ()
    {
        System.out.println(Thread.currentThread().getName() + " : popping");
        while (isEmpty())
        {
            System.out.println(Thread.currentThread().getName() + " : waiting to pop");
            try {
            wait();
            }
            catch (InterruptedException ex) {}
        }
        //end while

        int item = stackData[stackTop--];
        System.out.println(Thread.currentThread().getName() + " : notifying after pop (Popped "+ item + ")");
        notify();
        return item;
    }
}
synchronized public void push(int item)
{
    System.out.println(Thread.currentThread().getName() + 
        ": pushing " + item) ;

    while (isFull())
    {
        System.out.println(Thread.currentThread().getName() + 
            ": waiting to push"); 

        try
        {
            wait();
        } catch (InterruptedException ex) {} 

    }

    stackData[++stackTop]= item;
    System.out.println(Thread.currentThread().getName() + 
        ": notifying after push") ;

    notify();
}

public boolean isEmpty()
{
    if (stackTop < 0 ) return true;
    else return false;
}

public boolean isFull()
{
    if (stackTop >= capacity -1) return true;
    else return false;
}
package threadtest;

class Popper extends Thread {

    StackImpl stack;

    public Popper(String name, StackImpl stack) {
        super(name);
        this.stack = stack;
        System.out.println(this + " : created");
        start();
    }

    public void run() {
        while (true) {
            stack.pop();
        }
    }
}

class Pusher extends Thread {

    StackImpl stack;

    public Pusher(String name, StackImpl stack) {
        super(name);
        this.stack = stack;
        System.out.println(this+ " : created");
        start();
    }

    public void run() {
        while (true) {
            stack.push((int) (Math.random() * 100));
        }
    }
}
public class Test {

    public static void main(String[] args) {

        StackImpl aStack = new StackImpl(10);

        Pusher pusherA = new Pusher("PusherA", aStack);
        Pusher pusherB = new Pusher("PusherB", aStack);
        Popper popper = new Popper("Popper", aStack);

    }

}
Waiting and notifying SAMPLE outputs:

**Case 1: There is only a single “Popper” Thread:**

Thread[Popper,5,main]: created
**Popper: popping**
**Popper: waiting to pop**

**Case 2: There is only a single “Pusher” Thread:**

Thread[PusherA,5,main]: created
**PusherA: pushing 39**
**PusherA: notifying after push**
**PusherA: pushing 45**
**PusherA: notifying after push**
**PusherA: pushing 88**
**PusherA: notifying after push**
**PusherA: pushing 25**
**PusherA: notifying after push**
**PusherA: pushing 97**
**PusherA: notifying after push**
**PusherA: pushing 46**
**PusherA: notifying after push**
**PusherA: pushing 36**
**PusherA: notifying after push**
**PusherA: pushing 13**
**PusherA: notifying after push**
**PusherA: pushing 13**
**PusherA: notifying after push**
**PusherA: pushing 90**
**PusherA: notifying after push**
**PusherA: pushing 60**
**PusherA: waiting to push**
Case 3: There are TWO Pushers and ONE Popper:

Thread[PusherA,5,main]: created
Thread[PusherB,5,main]: created

PusherA: pushing 13
PusherA: notifying after push
PusherA: pushing 80
PusherA: notifying after push
PusherA: pushing 46
PusherA: notifying after push
Thread[Popper,5,main]: created
PusherB: pushing 56
PusherB: notifying after push

**PusherA: pushing 60**
PusherA: waiting to push
PusherB: pushing 3
**PusherB: waiting to push**

Popper: popping
Popper: notifying after pop (Popped 56)
**PusherA: notifying after push**
Popper: popping
Popper: notifying after pop (Popped 60)
**PusherB: notifying after push**
Popper: popping
Popper: notifying after pop (Popped 3)
Popper: popping
Popper: notifying after pop (Popped 46)
Popper: popping
Popper: notifying after pop (Popped 80)

.