Opgave 11.3

1. Begrænset stak (sekventiel version)

```
public class BoundedStack {
    protected Object[] stack;
    protected int size;
    protected int count = 0;

    public BoundedStack(int size) {
        this.size = size;
        stack = new Object[size];
    }

    public boolean isEmpty() {
        return count == 0;
    }

    public boolean isFull() {
        return count == size;
    }

    public void push(Object obj) {
        if (obj != null && !isFull())
            stack[count++] = obj;
    }

    public Object pop() {
        return count > 0 ? stack[--count] : null;
    }
}
```

2. Trådsikker begrænset stak med bevogtet suspendering

```
public class BoundedStackWithGuard extends BoundedStack {
    public BoundedStackWithGuard(int size) {
        super(size);
    }

    synchronized public boolean isEmpty() {
        return super.isEmpty();
    }

    synchronized public boolean isFull() {
        return super.isFull();
    }

    synchronized public void push(Object obj) {
        try {
            while (isFull())
                wait();
        } catch (InterruptedException e) {}
        super.push(obj);
        notify();
    }

    synchronized public Object pop() {
        try {
            while (isEmpty())
                wait();
        } catch (InterruptedException e) {}
        Object result = super.pop();
        notify();
        return result;
    }

    public static void main(String args[]) {
        BoundedStackWithGuard stack =
                new BoundedStackWithGuard(5);
        new Producer(stack, 15).start();
        new Consumer(stack, 10).start();
    }
}
```

3. Producent

```
public class Producer extends Thread {
    protected BoundedStack stack;
    protected int n;

    public Producer(BoundedStack stack, int n) {
        this.stack = stack;
        this.n = n;
    }

    public void run() {
        for (int i = 0; i < n; i++) {
            System.out.println("produce: " + i);
            stack.push(new Integer(i));
            try {
               sleep((int)(Math.random() * 100));
            }  catch (InterruptedException e) {}
        }
    }
}
```

4. Forbruger

```
public class Consumer extends Thread {
    protected BoundedStack stack;
    protected int n;

    public Consumer(BoundedStack stack, int n) {
        this.stack = stack;
        this.n = n;
    }

    public void run() {
        for (int i = 0; i < n; i++) {
            Object obj = stack.pop();
            System.out.println("\tconsume: "+obj);
            try {
                sleep((int)(Math.random() * 400));
            } catch (InterruptedException e) {}
        }
    }
}
```