Question 10.2–1: (Solution, p 5) Define the following terms as they relate to processes and threads.

a. *race*

b. *deadlock*

Question 10.2–2: (Solution, p 5)

At right is a class for representing a long-term computation. It attempts to implement two methods.

`void process()`
: Performs a long-term computational process.

`boolean abort()`
: Initiates an abort of the computation if the computation is in progress, returning `true` if the computation had to be aborted prematurely.

```
class LongProcess {
    boolean in_progress = false;
    boolean abort_requested = false;
    int value;

    void process() {
        in_progress = true;
        value = 0;
        while(!abort_requested && value >= 0) {
            value++; // (just a placeholder for
                     // long-term computation)
        }
        in_progress = false;
abort_requested = false;
    }

    boolean abort() {
        if(in_progress) {
            abort_requested = true;
return true;
        } else {
return false;
}
    }
}
```

a. Assuming that only one thread calls `process` at a time, and only one thread calls `abort`, describe a situation where the given attempt fails to work as specified.

b. Repair the code so that it works correctly in all situations; you can simply indicate your modifications in the code above. Your repairs should not depend on the specific characteristics of the long-term computation (particularly, the repeated incrementing of `i` that appears in this code).

Question 10.3–1: (Solution, p 5) Explain what happens when a program calls the `wait` method inherited from the Object class in Java.

Question 10.3–2: (Solution, p 5) Consider the following code implementing a Web server.

```
 1 public class WebServer {
 2     private class Transaction extends Thread {
 3         Socket browser;
 4         public Transaction(Socket browser) {
 5             this.browser = browser;
 6         }
 7         public void run() {
 8             // (read and handle browser's request)
 9             transactions.remove(this);
10         }
11     }

12     private LinkedList transactions = new LinkedList();

13     public void run() {
14         ServerSocket server = new ServerSocket()
15         while(true) {
16             Socket browser = server.accept();
17             Transaction t = new Transaction(browser);
18             transactions.add(t);
19             t.start();
20         }
21     }

22     public static void main(String[] args) {
23         new WebServer().run();
24     }
25 }
```

Suppose we wanted to alter this Web server so that it handles only 5 browser requests at a time. (We might want to do this to avoid having the server overload the computer.) How could you alter the above code to accomplish this?

Question 10.3–3: (Solution, p 6)

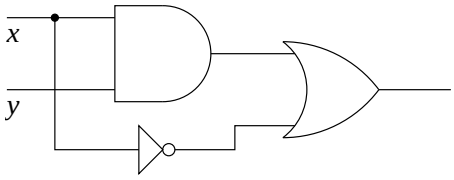
The skeleton at right describes a Java class with two two methods, `enter` and `release`. Complete the outlined methods so they work as follows:

`void enter()`
: Stalls the thread calling the method indefinitely, until it is released by another thread via a `release` message.

`void release(int n)`
: Releases `n` of the threads stalled within the `enter` method. (Do not worry about what happens if there are fewer than n threads in `enter` at that time.)

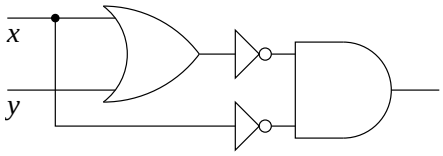
```
class Lock {

    public synchronized void enter() {




    }
    public synchronized void release(int n) {



    }
}
```

Question 11.1–1: (Solution, p 6) For the following circuit, write a truth table tabulating the circuit's output for each combination of inputs.



Question 11.1–2: (Solution, p 6) How deep is the circuit appearing in Question 11.1–1?

Question 11.2–1: (Solution, p 6) For the following circuit, write the Boolean expression that most closely corresponds to the circuit.



Question 11.2–2: (Solution, p 6) Draw a circuit representing each of the following Boolean expressions.

a. $\overline{x + y} + x$

b. $\overline{xy + \bar{x}\bar{y}}$

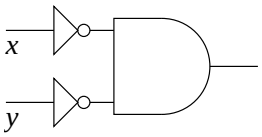
Question 11.2–3: (Solution, p 6) For each of the following Boolean expressions, write a truth table tabulating the expression's value for each combination of variable values.

a. $x + \overline{x + y}$

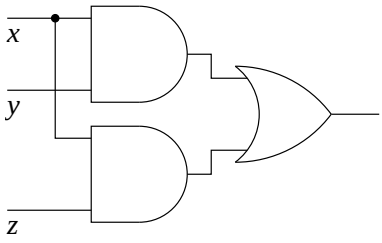
b. $xy + \overline{x + z}$

Question 11.2–4: (Solution, p 7) For each of the following, draw a smaller circuit (i.e., fewer gates) accomplishing the same task as the circuit given.

a.



b.



Question 11.2–5: (Solution, p 7) What is the *unsimplified* sum-of-products expression for the following truth tables? (Use multiplication for AND, addition for OR.)

a.

x	y	out
0	0	1
0	1	0
1	0	1
1	1	0

b.

x	y	out
0	0	1
0	1	1
1	0	0
1	1	1

c.

x	y	z	out
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Question 11.3–1: (Solution, p 7) Simplify the following sum-of-products expressions. For those that cannot be simplified using the technique from Section 2.3, you may simply state this fact.

a. $xy + \bar{x}y + x\bar{y}$

b. $\bar{x}yz + x\bar{y}z + xy\bar{z} + \bar{x}\bar{y}\bar{z}$

c. $\bar{x}yz + x\bar{y}\bar{z} + x\bar{y}z + xy\bar{z} + xyz$

Question 11.3–2: (Solution, p 7) Construct a *simplified* Boolean expression corresponding to the following truth table.

x	y	z	output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Question 11.3–3: (Solution, p 7) Using the Boolean identities, convert each of the following into a sum-of-products expression. Show each step of your simplification, and give the name of the identity used for that step. (You do not need to include steps involving the commutative or distributive laws.)

a. $\overline{x + yz}$

b. $(x + y)(\bar{x} + \bar{y})$

Solution 10.2–1: (Question, p 1)

a. A *race* occurs when the behavior of a program depends on the scheduling of concurrently running processes or threads.

b. *Deadlock* occurs when there is a set of processes or threads waiting on locks held by each other. For example, if one thread holds a lock on `a` and wants a lock on `b`, and another thread holds a lock on `b` and wants a lock on `a`, there is deadlock.

Solution 10.2–2: (Question, p 1)

a. The aborting thread could determine that a thread is currently processing, and then a context switch could occur before that thread sets the `abort_requested` variable. The processing thread could then complete during its time slice. Then the aborting thread could pick up the CPU again and set `abort_requested` to `true`. Now, if a thread tries the computation much later, it will be aborted immediately.

b. To fix the first problem, we place a synchronized block around the following in the `process` method.

```
        synchronized(this) {
            in_progress = false;
abort_requested = true;
        }
```

Also, make the `abort` method synchronized.

```
    synchronized boolean abort() {
```

Solution 10.3–1: (Question, p 1) The thread that calls the `wait` method releases the lock on the object for which `wait` was called. The machine then puts that thread into a wait queue for this object, so that the thread is stalled within the `wait` method. It is not removed from this object's wait queue until somebody calls `notify` or `notifyAll` on the object, at which time the waiting thread stalls until it can reacquire its lock on the object. Once the thread gets this lock, the thread returns from the `wait` method.

Solution 10.3–2: (Question, p 2) Replace line 18 with the following.

```
15          while(true) {
16              Socket browser = server.accept();
17              Transaction t = new Transaction(browser);
18a             synchronized(this) {
18b                 while(transactions.size() > 5) {
18c                     try { wait(); } catch(InterruptedException e) {}
18d                 }
18e                 transactions.add(t);
18f             }
19              t.start();
```

This prevents the server from starting a thread until there are only 5 transactions in the pool.
But we must have some code for awaking the web server when a thread terminates. Therefore, we should replace line 9 with the following.

```
9a              synchronized(WebServer.this) {
9b                  transactions.remove(this);
9c                  WebServer.this.notifyAll();
9d              }
```

Solution 10.3–3: (Question, p 2)

```
class Lock {
    int released = 0;

    public synchronized void enter() {
        while(released <= 0) {
            try {
                wait();
            } catch(InterruptedException e) {}
        }
        --released;
    }
    public synchronized void release(int n) {
        released += n;
        notifyAll();
    }
}
```

Solution 11.1–1: (Question, p 3)

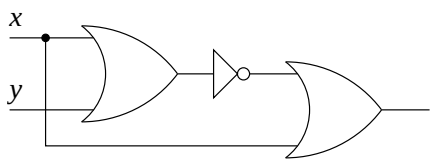
x	y	output
0	0	1
0	1	1
1	0	0
1	1	1

Solution 11.1–2: (Question, p 3) 2 (all paths from x or y to the output go through two gates)

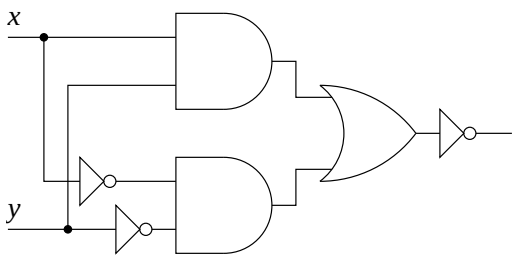
Solution 11.2–1: (Question, p 3) $(\overline{x+y})\,\bar{x}$

Solution 11.2–2: (Question, p 3)

a.



b.



Solution 11.2–3: (Question, p 3)

a.

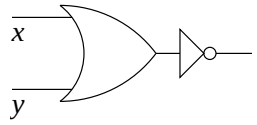
x	y	output
0	0	1
0	1	0
1	0	1
1	1	1

b.

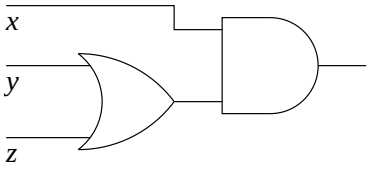
x	y	z	output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Solution 11.2–4: (Question, p 3)

a. This involves an application of the DeMorgan's law $\overline{x}\overline{y} = \overline{x + y}$.



b. This involves an application of the distributive law $xy + xz = x(y + z)$.



Solution 11.2–5: (Question, p 3)

a. $\overline{x}\overline{y} + x\overline{y}$
b. $\overline{x}\overline{y} + \overline{x}y + xy$
c. $\overline{x}\overline{y}\overline{z} + \overline{x}\overline{y}z + x\overline{y}z$

Solution 11.3–1: (Question, p 4)

a. $x + y$
b. cannot be simplified using the technique from Section 2.3.
c. $yz + x$

Solution 11.3–2: (Question, p 4) $\overline{x}\overline{z} + \overline{y}\overline{z} + xyz$ is good. (One could go further, though, and write $(\overline{x} + \overline{y})\overline{z} + xyz$ or even $\overline{x\overline{y}\overline{z}} + xyz$.)

Solution 11.3–3: (Question, p 4)

a.

$\overline{x + yz}$	original expression
$\overline{x}\,\overline{yz}$	DeMorgan's law
$\overline{x}(\overline{y} + \overline{z})$	DeMorgan's law
$\overline{x}\overline{y} + \overline{x}\overline{z}$	distributive law

b.

$(x + y)(\overline{x} + \overline{y})$	original expression
$(x + y)\overline{x} + (x + y)\overline{y}$	distributive law
$x\overline{x} + y\overline{x} + x\overline{y} + y\overline{y}$	distributive law
$0 + y\overline{x} + x\overline{y} + 0$	inverse law
$\overline{x}y + x\overline{y}$	identity law