

**Question 11.1-1:** (Solution, p 6) Suppose we are writing a Web server. Explain how threads would be useful for our program, and describe why they are useful in this context.

**Question 11.1-2:** (Solution, p 6) Name and explain two reasons that threads are a useful in programming.

**Question 11.1-3:** (Solution, p 6)

Modify the two classes below so that when the user runs `Main`'s `main` method, it allows the user to type "begin" to initiate a `Counter` thread and "end" to stop the currently running `Counter` thread. (Don't worry about the possibility that the user might type "begin" while a `Counter` thread is going or "end" when no `Counter` thread is going.)

The `Counter` thread should display a counter as it increments each second. The following is code to accomplish this.

```
while(true) {
    ++counter;
    System.out.println(counter);
    try { Thread.sleep(1000); }
    catch(InterruptedException e) {}
}
```

```
begin
1
2
3
4
5
end
begin
1
2
3
end
```

The transcript at right illustrates how the program should work.

(Your program should not use any deprecated methods from the `Thread` class. If you stick to what we discussed in class, this won't be a problem for you.)

```
public class Counter extends Thread {
    import java.io.*;
    public class Main {
        public static void main(String[] args) {
            BufferedReader user = new BufferedReader(
                new InputStreamReader(System.in));

            while(true) {
                try {
                    String line = user.readLine();
                    if(line.equals("begin")) {

                        } else {

                    }
                } catch(IOException e) {}
            }
        }
    }
}
```

## 2 Questions

---

### Question 11.1–4: (Solution, p 6)

Suppose we were to run the program at right.  
Which of the following outputs might occur?  
(Place a checkmark to each possible output.)

- a. 1 1 2 4
- b. 1 2 3
- c. 1 2 3 4
- d. 1 2 4
- e. 3 1 2 4
- f. 4
- g. 4 3 2 1
- h. 4 4 4 4

```
public class PrintNums extends Thread {
    static int i;

    public void run() {
        System.out.print(" " + i);
    }

    public static void main(String[] args) {
        for(i = 1; i <= 4; i++)
            (new PrintNums()).start();
    }
}
```

### Question 11.2–1: (Solution, p 7)

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 11.2–2:** (Solution, p 7)

Suppose we have two threads using the code at right. One frequently calls `advance` to advance the computation of primes, while another frequently calls `output` to output the current information.

a. Describe a situation in which output may display erroneous information.

b. Edit the code at right to fix this. Your fix must allow a thread to execute `output` even when another thread is inside the `isPrime` method.

```
public class PrimeCounter {
    private int last_checked = 1;
    private int primes_found = 0;

    public void advance() {
        if(isPrime(last_checked + 1)) {
            last_checked++;
            primes_found++;
        } else {
            last_checked++;
        }
    }

    public void output() {
        System.out.println(primes_found
            + " primes <= " + last_checked);
    }

    private boolean isPrime(int n) {
        for(int i = 2; i * i <= n; i++) {
            if(n % i == 0) return false;
        }
        return true;
    }
}
```

**Question 11.2–3:** (Solution, p 7) Define the term *deadlock* as it relates to processes and threads.

**Question 11.2–4:** (Solution, p 8)

For the Java program at right, suppose we have two threads, a and b, where a executes the `runA` method and b executes the `runB` method. Give an example of code for `runA` and `runB` which could result in deadlock between the two threads.

```
public class Deadlock {
    Thread a;
    Thread b;

    public void runA() {

    }

    public void runB() {

    }
}
```

**Question 11.3–1:** (Solution, p 8) Explain what happens when a program calls the `wait` method inherited from the `Object` class in Java.

## 4 Questions

---

**Question 11.3-2:** (Solution, p 8) 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 11.3-3:** (Solution, p 9)

The skeleton at right describes a Java class with 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 12.1–1:** (Solution, p 9) Reduce the following lambda expression, showing *every* intermediate step. It should reduce to a number.

$$(\lambda f.\lambda a.f (a + 3) - (f a) - (f 3)) (\lambda x.x * x) 7$$

**Question 12.1–2:** (Solution, p 9) Simplify the following lambda expression to normal form. Show *every* intermediate step.

$$(((\lambda f.\lambda g.g f) (\lambda x.10 - x)) (\lambda y.\lambda x.y (y x)))4$$

**Question 12.2–1:** (Solution, p 10) Suppose we are using an applied lambda calculus which supports the infix arithmetic operators  $*$ ,  $+$ , and  $-$ . Reduce the following lambda expression in normal order, showing *every* intermediate step. It should reduce to a number.

$$(\lambda a.\lambda b.a (b + 1) - a b) (\lambda x.x * x * x) 2$$

*Use normal evaluation order!*

**Question 12.3–1:** (Solution, p 10) Suppose we have the Y combinator (i.e., the fixed-point combinator) defined.

$$Y \equiv \lambda f.(\lambda x.f (x x)) (\lambda x.f (x x))$$

Suppose, moreover, that we have infix operators for subtraction, addition, and comparison ( $=$ ,  $<$  and  $\leq$ ), as well as **if-then-else** expressions. Give a lambda expression to compute the  $n$ th Fibonacci number, recalling that the 0th Fibonacci is 0 and the 1st Fibonacci is 1.

**Question 12.4–1:** (Solution, p 10) Suppose we want to define a way to represent pairs (or 2-tuples) in the lambda calculus. We can do this by defining the following function *pair*, which takes two arguments and generates a representation of a pair of those two values.

$$pair \equiv \lambda x.\lambda y.\lambda f.f x y$$

For example, to represent the value (2,3), we would call *pair* 2 3, which would give us  $\lambda f.f 2 3$ .

Give a lambda expression for the function *first* that takes a pair represented as above and extracts the first item of the pair. Using your definition, I should be able to write *first* (*pair* 2 3) and get 2.

**Question 12.4–2:** (Solution, p 10) Using the Church numerals, we can define the *inc* function in the lambda calculus to take a Church numeral representing some number  $n$  and return the Church numeral representing  $n + 1$ .

$$inc \equiv \lambda a.\lambda s.\lambda z.a s (s z)$$

Give a lambda calculus definition of the addition function  $+$ , which takes two Church numerals representing two numbers  $m$  and  $n$  and returns a Church numeral representing their sum,  $m + n$ .

## 6 Solutions

---

**Solution 11.1–1:** (Question, p 1) For good performance, a Web server must communicate with many browsers simultaneously, and it is natural to have a single thread handle each individual browser connection. This technique is useful for three reasons. [One is sufficient for the solution.]

- Implementing the Web server as a single process without threads requires the programmer to write code that switches between discussions among browsers, which is difficult and conducive to programmer errors.
- If the Web server were implemented to create a new process for each Web browser connection, the high overhead of process creation would damage the program's efficiency.
- Also in the multi-process solution, having multiple processes would interfere with the possibility that the responding processes might want to share information between themselves.

**Solution 11.1–2:** (Question, p 1) Any two of the following would be good.

- A programmer using threads can conceptualize a process as doing one thing at a time, even though the process will actually be doing many things simultaneously.
- Threads use less system resources than separate processes.
- A process can continue perform computation during long I/O tasks (even when a system call is blocked).

**Solution 11.1–3:** (Question, p 1)

```
public class Counter extends Thread { import java.io.*;
    public boolean stopped = false;   public class Main {
                                     public static void main(String[] args) {
                                     BufferedReader user = new BufferedReader(
                                     new InputStreamReader(System.in));
                                     Counter thread = null;
                                     while(true) {
                                     try {
                                     String line = user.readLine();
                                     if(line.equals("begin")) {
                                     thread = new Counter();
                                     thread.start();
                                     } else {
                                     thread.stopped = true;
                                     }
                                     } catch(IOException e) {}
                                     }
                                     }
                                     }
}
```

**Solution 11.1–4:** (Question, p 2) The possible outputs are (c.), (e.), (g.), and (h.).

**Solution 11.2–1:** (Question, p 2)

- 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 11.2–2:** (Question, p 3)

- a. A thread could call `output` while the other thread is between the “`last_checked++;`” and “`primes_found++;`” lines of `advance`, in which case it would display a count that is one less than it ought to be.
- b. Modify `advance` to add a synchronized block around these two lines.

```

synchronized(this) {
    last_checked++;
    primes_found++;
}

```

Also, mark the entire `output` method as synchronized.

```

public synchronized void output() {

```

**Solution 11.2–3:** (Question, p 3) *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.

## 8 Solutions

---

### Solution 11.2–4: (Question, p 3)

```
public class Deadlock {
    Thread a;
    Thread b;

    public void runA() {
        synchronized(a) {
            synchronized(b) {
                System.out.println("I'm A");
            }
        }
    }
    public void runB() {
        synchronized(b) {
            synchronized(a) {
                System.out.println("I'm B");
            }
        }
    }
}
```

[In this example, a could get a lock on itself, and then a context-switch could allow b to get a lock on itself. When b then tries to get a lock on a, it will have to wait until a releases its own lock. And a will not be able to continue until b releases its lock. We have deadlock.]

**Solution 11.3–1:** (Question, p 3) 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 11.3–2: (Question, p 4) 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 11.3–3:** (Question, p 4)

```

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 12.1–1:** (Question, p 5)

$$\begin{aligned}
 & (\lambda f.\lambda a.f(a+3) - (fa) - (f3))(\lambda x.x * x)7 \\
 \Rightarrow & (\lambda a.(\lambda x.x * x)(a+3) - (\lambda x.x * x)a - (\lambda x.x * x)3)7 \\
 \Rightarrow & (\lambda x.x * x)(7+3) - (\lambda x.x * x)7 - (\lambda x.x * x)3 \\
 \Rightarrow & (7+3) * (7+3) - (\lambda x.x * x)7 - (\lambda x.x * x)3 \\
 \Rightarrow & (7+3) * (7+3) - 7 * 7 - (\lambda x.x * x)3 \\
 \Rightarrow & (7+3) * (7+3) - 7 * 7 - 3 * 3 \\
 \Rightarrow & 10 * (7+3) - 7 * 7 - 3 * 3 \\
 \Rightarrow & 10 * 10 - 7 * 7 - 3 * 3 \\
 \Rightarrow & 100 - 7 * 7 - 3 * 3 \\
 \Rightarrow & 100 - 49 - 3 * 3 \\
 \Rightarrow & 51 - 3 * 3 \\
 \Rightarrow & 51 - 9 \\
 \Rightarrow & 42
 \end{aligned}$$

**Solution 12.1–2:** (Question, p 5)

$$\begin{aligned}
 & (((\lambda f.\lambda g.g f) (\lambda x.10 - x)) (\lambda y.\lambda x.y (y x))) 4 \\
 \Rightarrow & ((\lambda g.g (\lambda x.10 - x)) (\lambda y.\lambda x.y (y x))) 4 \\
 \Rightarrow & ((\lambda y.\lambda x.y (y x)) (\lambda x.10 - x)) 4 \\
 \Rightarrow & (\lambda x.(\lambda x.10 - x) ((\lambda x.10 - x) x)) 4 \\
 \Rightarrow & (\lambda x.10 - x) ((\lambda x.10 - x) 4) \\
 \Rightarrow & 10 - ((\lambda x.10 - x) 4) \\
 \Rightarrow & 10 - (10 - 4) \\
 \Rightarrow & 10 - 6 \\
 \Rightarrow & 4
 \end{aligned}$$

## 10 Solutions

---

**Solution 12.2–1:** (Question, p 5)

$$\begin{aligned} & (\lambda a.\lambda b.a(b+1) - a b) (\lambda x.x * x * x) 2 \\ \Rightarrow & (\lambda b.(\lambda x.x * x * x) (b+1) - (\lambda x.x * x * x) b) 2 \\ \Rightarrow & (\lambda x.x * x * x) (2+1) - (\lambda x.x * x * x) 2 \\ \Rightarrow & (2+1) * (2+1) * (2+1) - (\lambda x.x * x * x) 2 \\ \Rightarrow & 3 * (2+1) * (2+1) - (\lambda x.x * x * x) 2 \\ \Rightarrow & 3 * 3 * (2+1) - (\lambda x.x * x * x) 2 \\ \Rightarrow & 9 * (2+1) - (\lambda x.x * x * x) 2 \\ \Rightarrow & 9 * 3 - (\lambda x.x * x * x) 2 \\ \Rightarrow & 27 - (\lambda x.x * x * x) 2 \\ \Rightarrow & 27 - 2 * 2 * 2 \\ \Rightarrow & 27 - 4 * 2 \\ \Rightarrow & 27 - 8 \\ \Rightarrow & 19 \end{aligned}$$

**Solution 12.3–1:** (Question, p 5)  $Y (\lambda f.\lambda n.\mathbf{if} \ n \leq 1 \ \mathbf{then} \ n \ \mathbf{else} \ f \ (n - 1) + f \ (n - 2))$

**Solution 12.4–1:** (Question, p 5)  $first \equiv \lambda p.p (\lambda x.\lambda y.x)$

**Solution 12.4–2:** (Question, p 5)  $\lambda m.\lambda n.n \ inc \ m$