1. [6 points]
   
a. If \(((x-1) \& x) == 0\) for an int value \(x > 0\), what can you say about \(x\)?
   
   \(x\) is a power of 2. Since \(x > 0\), it has the form \(2^n + y2^{n+1}\) for some value \(y \geq 0\) and \(n \geq 0\). That is, the \(n^{th}\) least significant digit is the first 1 bit in \(x\). Therefore, \(x - 1\) must have the form \(2^m - 1 + y2^{n+1}\), so that it consists of a 0 in the \(2^n\) place, 1's in all places to the right, and all other bits the same as for \(x\). Therefore, \(((x-1)\&x)\) has a 0 in the \(2^n\) place (because \(x - 1\) does), 0's in the positions to its right (because \(x\) does), and because the other bits are equal to those of \(x\), the result is \(y2^{n+1}\). Since the result is known to be 0, we see that \(y = 0\), which makes \(x\) a power of 2.
   
b. Suppose that Obj is some class. Assuming that the following segment of code compiles and runs without errors or exceptions, what might it print and why?
   
   ```java
   Obj P, Q;
   P = new Obj();
   Q = P;
   f(P);
   if (Q == P)
       System.out.println("Equal");
   else
       System.out.println("Unequal");
   ```
   
   Assuming that the constructor and call to \(f\) don't throw an exception, this must print "Equal." Because Java passes all parameters by value, nothing that \(f\) does to its formal parameter will alter the actual parameter \(P\). Because \(P\) is a local variable, \(f\) has no other access to it.
   
c. In the program below, assuming that calls to other functions run in constant time, what can you say about \(C(N)\), defined as “the worst-case execution time of \(\text{explore}\) executed when it is called with \(k = N\)?”
   
   ```java
   static boolean explore (int x, int k) {
       if (k <= 0)
           return false;
       if (explore (f(x), k-1))
           return true;
       if (explore (g(x), k-1))
           return true;
       return false;
   }
   ```
   
   \(C(N) \in \Theta(2^N)\). Each call to \(\text{explore}\) makes two recursive calls in the worst case, so counting calls to \(\text{explore}\), we see that \(C(0) = 1\), and \(C(N) = 1 + 2C(N-1)\), for \(N > 0\). Expanding, \(C(N) = 1 + 2(1 + 2(1 + \cdots)) \in \Theta(2^N)\).
d. The project 1 skeleton defined

```java
public class ExitException extends RuntimeException {
    ...
}
```

In the text below, what does a call to `start()` print?

```java
void start () {
    try {
        f (2);
    } catch (ExitException e) {
        System.out.println ("Handler #1");
    }
}
```

```java
void f (int k) {
    try {
        if (k == 0) {
            throw new ExitException ();
        }
        f (k-1);
    } catch (RuntimeException e) {
        System.out.printf ("Handler #2, k = %d\n", k);
        throw new ExitException ();
    }
}
```

*Answer:*

```
Handler #2, k = 0
Handler #2, k = 1
Handler #2, k = 2
Handler #1
```

e. If I match the pattern `([\d,]+?),([\d,]+),([\d,]+)` (or as a String literal, 

```
"([\d,]+?),([\d,]+),([\d,]+)"
```

) against the string "12,345,6,789,1011,12", what will be `group(1)`, `group(2)`, and `group(3)` of the resulting match?

*Answer:*

```
group(1) is "12", group(2) is "345,6,789,1011", group(3) is "12"
```
f. When executed, what does the main program in B print?

```java
abstract class A {
    int var = 1;
    static void s (A x) {
        System.out.printf ("A.s, x.f() = %d%n", x.f ());
    }
    int f () {
        return 10;
    }
    void h1 (C x) {
        System.out.printf ("A.h1, x.var=%d, var=%d%n", x.var, var);
        x.s (x);
    }
    void h2 (A x) {
        System.out.printf ("A.h2, x.var=%d, var=%d%n", x.var, var);
        x.s (x);
    }
    int g () {
        return 10;
    }
}

class B extends A {
    int var = 2;
    static void s (A x) {
        System.out.printf ("B.s, x.f() = %d%n", x.f ());
    }
    int f () {
        return 11;
    }
    static public void main (String[] args) {
        B y = new C ();
        y.h1 ((C) y);
        y.h2 (y);
    }
}

class C extends B {
    int var = 3;
    static void s (A x) {
        System.out.printf ("C.s, x.f() = %d%n", x.f ());
    }
    int f () {
        return 12;
    }
}
```

Solution:

- `A.h1, x.var=3, var=1`
- `C.s, x.f() = 12`
- `A.h2, x.var=1, var=1`
- `A.s, x.f() = 12`
2. [1 point] I was walking one evening some time ago about an hour after sunset, and saw a fairly bright planet (magnitude < 0.5) near the eastern horizon. Which planet(s) could it have been?

   Mars, Jupiter, or Saturn. (Uranus and Neptune are too dim. Venus and Mercury can never be that far from the sun.

3. [8 points] Using the following class definitions:

   ```java
   class IntList {
       public final int head;
       public IntList tail;
       public IntList (int head, IntList tail) {
           this.head = head; this.tail = tail;
       }
   }
   ```

   ```java
   class IntList2 {
       public IntList head;
       public IntList2 tail;
       public IntList2 (IntList head, IntList2 tail) {
           this.head = head; this.tail = tail;
       }
   }
   ```

   fill in the method below to agree with its comment. Define any additional methods you’d like.

   ```java
   /** Given the list of lists L, in which all lists have the same * length, returns the list formed by concatenating all the first * elements of the lists in L (in order), then the second elements, * and so forth. For example, if L contains * [ [1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12] ] * then the returned list contains * [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ] * This operation may destroy the contents of the original list. It * must create no new IntList objects. */
   */
   static IntList zip (IntList2 L) {
       if (L == null || L.head == null) return null;
       IntList result = L.head, end;
       end = null;
       while (L.head != null) {
           for (IntList2 p = L; p != null; p = p.tail) {
               if (end != null)
                   end.tail = p.head
               end = p.head;
               p.head = p.head.tail;
           }
           end.tail = null;
           return result;
       }
   }
   ```

Copyright (c) 2008 by P. N. Hilfinger. Reproduction prohibited.
4. [6 points] This problem involves filling in the implementation of a system that allows one to describe and simulate networks of components, such as the one pictured here:

```
A * B
```

```
C * D
```

which is supposed to indicate a device with four inputs \((A, B, C, D)\) that computes \(A \cdot B\) and \(C \cdot D\), then adds the resulting values, and then outputs the final result. To simplify our life, all devices will take two inputs and produce one output (at their pointy ends), but any particular kind of device might choose never to output anything or to use just one input (e.g., OUT in the diagram). The values that “flow” over the lines in the diagram will always be ints.

I want to be able to set up the system in the diagram with this program:

```
Device D1 = makeMultiplier (), D2 = makeMultiplier (),
        D3 = makeAdder (),
        D4 = makeOutput ();
D1.connect (D3, 0); // Connect D1's output to D3's input #0.
D2.connect (D3, 1); // Connect D2's output to D3's input #1.
D3.connect (D4, 0); // Connect D3's output to D4's input #0.
```

and then send in inputs like this:

```
D1.input(0, 15); // Send input value 15 to D1's input #0.
D1.input(1, 2);
D2.input(0, 4);
D2.input (1, 3); // At this point, the OUT device will print 42
D1.input(1, 6);
D1.input(0, 5); // Inputs can arrive in either order.
D2.input(0, 7);
D2.input(1, 8); // At this point, the OUT device will print 86
```

and so forth. To make it work, assume that the file with the above statements starts with

```
package mysystem;
import devices.Device;
import static devices.Devices.*;
```

Fill in the definitions on the next pages to implement this behavior. Your definitions must have the following properties:

- The resulting devices must work for any setup, not just the example.

- Your answer must contain no \texttt{if}, \texttt{while}, \texttt{for}, \texttt{switch} or \texttt{try} statements.

- If we later decide to implement a new kind of device, the only change we should make to \texttt{Devices} is to add a method like \texttt{makeMultiplier}, containing just a single return statement, and we must not have to change the class \texttt{Device} at all.
The program follows. This is mostly an exercise in understanding a design; you should not have a lot to write. Again, no if, while, for, switch or try statements.

    package devices;

    public class Devices {
        public static Device makeMultiplier () {

            return new Multiplier ();

        }

        // You don’t have to implement adders (essentially the same) or outputs.
    }

    public abstract class Device {
        /** The current values of my operands. */
        protected int[] operands;
        /** haveOperand[k] is true if I have received operand #k and not * yet used it to compute an output. */
        private boolean[] haveOperand;
        /** Device my output is connected to (null if output is unused). */
        protected Device outputDestination = null;
        /** Which input of outputDestination my output is connected to. */
        protected int destinationInputNumber;

        /** The total number of inputs I must receive before I produce a * value (that is, I need operands numbered 0 to numInputs ()-1). */
        protected abstract int numInputs ();
        /** Compute using my operands, and, if I produce outputs, transmit * the result to the Device my output is connected to. */
        protected abstract void compute ();

        protected Device () {
            outputDestination = null; destinationInputNumber = 0;

            operands = new int[numInputs ()];
            haveOperand = new boolean[numInputs ()];
        }

        continues on next page
/** Set input #INPUTNUMBER to VALUE. Compute result and set up for
* next round if have all inputs. Final: cannot be overridden. */
public final void input (int inputNumber, int value) {
    operands[inputNumber] = value; haveOperand[inputNumber] = true;
    for (boolean b : haveOperand)
        if (! b)
            return;
    compute ();
    java.util.Arrays.fill (haveOperand, false); /* Set all to false */
}

/** Connect my output to input #INPUTNUMBER of DEST. Final: cannot
* be overridden. */
public final void connect (Device dest, int inputNumber) {
    outputDestination = dest;
    destinationInputNumber = inputNumber;
}

// ADDITIONAL CLASS (in another package)
class Multiplier extends devices.Device {
    protected int numInputs () {
        return 2;
    }
    protected void compute () {
        outputDestination.input (destinationInputNumber,
                operands[0] * operands[1]);
    }
}