***Problem 1*** [18 points] *Short-answer: course policies, fundamental concepts, literals, operators, expression statements*

***1a*** [5 points] Fill in the following blanks:

Programming is automating problem solving.

MATLAB stands for Matrix Laboratory .

Java is strongly typed.

David I. Schwartz's office is in 5137 Upson Hall.

***1b*** [6 points] Fill in the blanks for the following code fragment that swaps the contents of **a** with **b** without directly assigning 1 to **b** and 2 to **a**. Hint: Use **tmp**:

```
int a=1, b=2, tmp;

tmp = a ; a = b ; b = tmp ;
```

***1c*** [1 point] Fill in the blank for the output for the following statement:

```
System.out.println("Output c: " + ( true && ( false || !( true || false ) ) ) );
```

```
Output c: false
```

***1d*** [1 point] Fill in the blank for the output for the following statement:

```
System.out.println("Output d: " + (1-2-3-4) );
```

```
Output d: -8
```

***1e*** [3 points] Show the output for the following statement:

```
System.out.println("Hi\nBye!");
```

```
Hi
Bye!
```

***1f*** [2 points] Fill in the blanks for the output for the following statements:

```
int x=2, y=1;
x = y-- -x;
System.out.println("x: "+x+" y: "+y);
```

```
x: -1  y: 0
```

***Problem 2*** [38 points] *Selection statements, processing input, remainder operator, Strings, relations, user I/O*

Complete the following code in class **`Problem2`** by filling in the blanks and boxes. The user is prompted to enter an integer-valued **`temperature`**, which is rated using Strings according to the following criteria:

| `temperature` | `rating` |
|---|---|
| 90 – 92 | `"almost hot"` |
| 80 – 82 | `"almost warm"` |
| 70 – 72 | `"OK"` |
| 60 – 62 | `"very cool"` |
| 50 – 52 | `"very cold"` |

| `temperature` | `rating` |
|---|---|
| 93 – 96 | `"hot"` |
| 83 – 86 | `"warm"` |
| 73 – 76 | `"OK"` |
| 63 – 66 | `"cool"` |
| 53 – 56 | `"cold"` |

| `temperature` | `rating` |
|---|---|
| 97 – 99 | `"very hot"` |
| 87 – 89 | `"very warm"` |
| 77 – 79 | `"OK"` |
| 67 – 69 | `"almost cool"` |
| 57 – 59 | `"almost cold"` |

To determine the **`rating`**, your program will:

- obtain a user input **`temperature`**. The user must enter an integer. If the user enters an out-of-bounds value, the program finishes. Assume that the user does not enter an illegal type.
- determine an initial **`rating`** based on initial temperature ranges: 90 – 99, 80 – 89, 70 – 79, 60 – 69, and 50 – 59.
- modify **`rating`** based on the specific temperature range, as demonstrated in the chart above. Note that the **`rating`** modifications of **`"warm"`** and **`"hot"`** are different from the modifications of **`"cool"`** and **`"cold"`**. For instance, given a **`temperature`** of 81, the **`rating`** is **`"almost warm"`**, but for a **`temperature`** of 51, the **`rating`** is **`"very cold"`**. Hint: A remainder operation (using **`%`**) can help identify the specific portion of the initial range.
- print `Done!` when finished processing input.

**Example session**:

```
Enter temperature: 52
Your temperature is very cold.
Enter temperature: 67
Your temperature is almost cool.
Enter temperature: 99
Your temperature is very hot.
Enter temperature: 32
Done!
```

---

```
public class Problem2 {
   public static void main(String[] args) {
      // Initialize variables. Assume user enters integers:
         TokenReader in = new TokenReader(System.in);  // input reader
         System.out.print("Enter temperature: ");      // prompt for temperature
         int temperature = in.readInt();              // user-input temperature
         String rating;                               // temperature rating

      // Rate temperatures until user enters out-of-bounds integer:

         while (temperature > 49 && temperature < 100) {

            // Apply initial temperature rating:

               if      ( temperature >= 90 ) rating = "hot";

               else if ( temperature >= 80 ) rating = "warm";

               else if ( temperature >= 70 ) rating = "OK";

               else if ( temperature >= 60 ) rating = "cool";

               else                          rating = "cold";
```

```
            // Modify $rating$ based on specific portion of temperature range:

              int rem = temperature % 10; // portion of temperature range

              // warm and hot conditions:
                 if (temperature >= 80) {
                     if (rem < 3)
                         rating = "almost" + " " + rating;
                     else if (rem > 6)
                         rating = "very" + " " + rating;
                 }

              // cool and cold conditions:
                 else if (temperature <= 69) {
                     if (rem > 6)
                         rating = "almost" + " " + rating;
                     else if (rem < 3)
                         rating = "very" + " " + rating;
                 }

            // Output the modified temperature rating:

               System.out.println("Your temperature is " + rating + ".");

            // Process next temperature:

               System.out.print("Enter temperature: ") ;

               temperature = in.readInt() ;
         } // end while

      System.out.println("Done!"); // finished processing

   } // method main
} // class Problem2
```

***Problem 3*** [44 points] *Algorithms, Repetition: conditional update, accumulation*

**Background**: *DIS.com* needs to ship boxes to consumers but is too cheap to hire more than one worker. A random amount of boxes (between 1 and 3, inclusive) spews forth from a chute into a bin with no boxes initially inside. The lonely worker must take boxes out of this bin and place them in a truck for shipping. Because of sporadic back pains, the worker will take a random amount of boxes (between 1 and 3, inclusive) out of the bin. Unfortunately, the bin may hold a maximum of 7 boxes. To prevent the bin from filling up and shutting down the whole operation, the worker must keep working. Although the worker starts completely refreshed, all this work tires the worker. So, every four trips between the bin and truck the worker's efficiency drops by 25% of the previous value. Eventually the worker will cease carrying enough boxes, and thus, the bin will fill up.

**Algorithm**: You will simulate this problem with a program that has this algorithm:

- Set up initial values.
- Attempt to add boxes to bin.
- If the bin is not full:
  - remove boxes that the worker is able to extract.
  - obtain more boxes from the chute.
  - repeat.
- Otherwise, stop the simulation and report the results.

**Tasks**: Fill in the blanks below to complete the code that performs the simulation:

- Initialize variables that represent these initial amounts: boxes to add to the bin (**`boxesToAdd`**), boxes taken from the bin (**`boxesTaken`**), boxes inside the bin (**`boxesInBin`**), and total amount boxes extracted by the worker (**`totalBoxes`**).
- Use a loop to perform each cycle of the algorithm. Each cycle consists of checking if the bin is full, updating the count of cycles and amounts of boxes (added, taken, total), decreasing worker efficiency (if necessary), and obtaining a new batch of boxes to add to the bin.
- Report the count of cycles of the simulation and the total amount of boxes taken by the worker.

**Notes**: Remember that you may *not* use arrays. You must use the code and blanks supplied for you!

---

```
public class Problem3 {

   public static void main(String[] args) {

      // Initialize variables:

         int MIN      = 1;    // minimum number of boxes for chute and worker

         int MAX      = 3;    // maximum number of boxes for chute and worker

         int MAXBOXES = 7;    // maximum number of boxes that the bin can hold

         // random amount of boxes (MIN to MAX, inclusive) for adding to bin:

            int boxesToAdd =

            (int) (Math.random()*(MAX-MIN+1) + MIN) ;

         int boxesInBin    = 0 ;   // current # of boxes in bin

         int boxesTaken    = 0 ;   // current # of boxes taken from the bin

         int totalBoxes    = 0 ;   // total # of boxes taken so far

         double eff        = 1 ;   // worker efficiency, which starts at 100%

         int count         = 0 ;   // count of cycles so far
```

```
      // Process each cycle of the simulation:
         while ( (boxesInBin + boxesToAdd) <= MAXBOXES) ) {

            count++ ;                                  // increment # of cycles

            boxesInBin += boxesToAdd ;                 // increment # of boxes in bin

            // Determine current # of boxes taken from bin. The worker may not
            // extract more boxes than are already in the bin:
               boxesTaken =

               (int) (eff * (Math.random()*(MAX-MIN+1) + MIN)) ;

               if ( boxesTaken <= boxesInBin ) {

                  totalBoxes += boxesTaken ;

                  boxesInBin -=  boxesTaken ;

               } else {

                  totalBoxes += boxesInBin ;

                  boxesInBin = 0 ;

               }

            // Reduce worker's efficiency by 25% every 4th cycle of the simulation:

               if ( count % 4 == 0 )

                  eff = eff * 0.75 ;

            // Obtain new boxes to attempt adding in bin for the next cycle:

                boxesToAdd = (int) (Math.random()*(MAX-MIN+1) + MIN) ;

         } // end while

      // Report results:

         System.out.println("Total boxes taken:\t" + totalBoxes );

         System.out.println("Number of cycles of simulation:\t" + count );

   } // method main

} // class Problem3
```