Lab Questions
Big Java, Late Objects / Java for Everyone, 2e
Chapter Number: 4 Loops

1.1) Loops provide a mechanism for repeating a block of code called the *loop body*. We begin this lab by experimenting with *while loops*, the simplest form of loop code. Many loops are controlled with a single variable, which we will refer to as the *loop control variable* or *the loop index*.

Consider the code below. What is the output the program produces?

```
/**
   A simple program that prints a loop control variable.
*/
public class SimpleLoop
{
   public static void main(String[] args)
   {
      int i = 0;
      int limit = 6;
      while (i < limit)
      {
         System.out.println("i = " + i);
         i++;
      }
   }
}
```

1.2) Consider the code below again. What happens if you comment out the line that increments `i`? Will the program ever stop looping?

```
/**
   A simple program that prints a loop control variable.
*/
public class SimpleLoop
{
   public static void main(String[] args)
   {
      int i = 0;
      int limit = 6;
      while (i < limit)
      {
         System.out.println("i = " + i);
         i++;
      }
   }
}
```

1.3) Manipulating the loop control variable is a critical skill in learning to write code with loops. Modify the program in Lab 4.1.1 so that it produces the following output:

```
i = 6
i = 8
i = 10
i = 12
i = 14
i = 16
i = 18
...
i = 98
```

2.1) There is a famous story about a primary school teacher who wanted to occupy his students' time by making the children compute the sum of `1 + 2 + 3 + … + 100` by hand. As the story goes, the teacher was astounded when one of the children immediately produced the correct answer: `5050`. The student, a child prodigy, was Carl Gauss, who grew up to be one of the most famous mathematicians of the eighteenth century. Repeat Gauss's remarkable calculation by writing a loop that will compute and print the above sum. After you have the program working, rewrite it so you can compute `1 + 2 + … + n` where `n` is any positive integer.

2.2) Java provides three types of loops: *while*, *for*, and *do* (also called *do-while*). Theoretically, they are interchangeable – any program you write with one kind of loop could be rewritten using any of the other types of loops. As a practical matter, though, it is often the case that choosing the right kind of loop will make your code easier to produce, debug, and read. It takes time and experience to learn to make the best loop choice, so this is an exercise to give you some of that experience.

Rewrite Lab 4.2.1 using a *for* loop. Repeat the exercise again but this time use a *do while* loop. Which form of loop seems to work best? Why?

3) Write a program that uses a *while* loop. In each iteration of the loop, prompt the user to enter a number – positive, negative, or zero. Keep a running total of the numbers the user enters and also keep a count of the number of entries the user makes. The program should stop whenever the user enters "q" to quit. When the user has finished, print the grand total and the number of entries the user typed.

4) You can test to see if an integer, x, is even or odd using the Boolean expression `(x / 2) * 2 == x`. Integers that are even make this expression true, and odd integers make the expression false.

Use a *for* loop to iterate five times. In each iteration, request an integer from the user. Print each integer the user types, and whether it is even or odd. Keep up with the number of even and odd

integers the user types, and print "Done" when finished, so the user won't try to type another integer. Finally, print out the number of even and odd integers that were entered.

5) One of the oldest numerical algorithms was described by the Greek mathematician, Euclid, in 300 B.C. That algorithm is described in Book VII of Euclid's multi-volume work *Elements*. It is a simple but very effective algorithm that computes the greatest common divisor of two given integers.

For instance, given integers 24 and 18, the greatest common divisor is 6, because 6 is the largest integer that divides evenly into both 24 and 18. We will denote the greatest common divisor of x and y as gcd(x, y). The algorithm is based on the clever idea that the gcd(x, y) = gcd($x - y$, y) if $x >= y$. The algorithm consists of a series of steps (loop iterations) where the "larger" integer is replaced by the difference of the larger and smaller integer.

In the example below, we compute gcd(72, 54) and list each loop iteration computation on a separate line. The whole process stops when one of the integers becomes zero. When this happens, the greatest common divisor is the non-zero integer.

```
gcd(72, 54) = gcd(72 - 54, 54) = gcd(18, 54)
gcd(18, 54) = gcd(18, 54 - 18) = gcd(18, 36)
gcd(18, 36) = gcd(18, 36 - 18) = gcd(18, 18)
gcd(18, 18) = gcd(18 - 18, 18) = gcd(0, 18) = 18
```

To summarize:

Create a loop, and subtract the smaller integer from the larger one (if the integers are equal you may choose either one as the "larger") during each iteration.
Replace the larger integer with the computed difference.
Continue looping until one of the integers becomes zero.
Print out the non-zero integer.

Use the code below to prompt the user for the two integers.

```
public class GCD
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);
      System.out.println("Enter the first integer: ");
      int x = in.nextInt();
      System.out.println("x = " + x);
      System.out.println("Enter the second integer: ");
      int y = in.nextInt();
      System.out.println("y = " + y);
      // Your gcd computation code goes here
   }
}
```

6.1) Use nested *for* loops to produce the following output

```
1  1
1  2
1  3
1  4
2  1
2  2
2  3
2  4
```

Let the outer loop print the numbers in the left column, and the inner loop print the numbers in the right column. In each iteration, print the loop control variables to produce the output.

6.2) Repeat Lab 6.1 using nested *while* loops.

6.3) Repeat Lab 6.1 using nested *do* loops.

7.1) Use nested *for* loops to produce the following output:

```
X
XX
XXX
XXXX
XXXXX
```

The outer loop can control the number of rows that will be printed. The inner loop can control the number of X's that print on a single line. The trick is to notice that there is a relationship between the row number and the number of X's in the row. This relationship allows you to use the outer loop control variable to control the inner loop.

7.2) Use nested loops and the code from Lab 2.1 to produce the output below. The outer loop control variable can be used to help write the inner loop.

```
The sum of positive integers from 1 to 1 is 1
The sum of positive integers from 1 to 2 is 3
The sum of positive integers from 1 to 3 is 6
The sum of positive integers from 1 to 4 is 10
The sum of positive integers from 1 to 5 is 15…
The sum of positive integers from 1 to 100 is 5050
```

Because we already have a program that computes `1 + 2 + ... + n` for any positive integer `n`, we can embed that code in a second loop that loops 100 times. The outside loop control variable can be used to control the number of times we loop and more importantly, it can be used to control the upper limit of the inner computation.

Write the code that will produce the output described above.

8) When you have correctly completed the program from Lab 6.3.1 for computing the greatest common divisor for a single pair of integers, add a surrounding outer loop that allows the user to keep computing gcds for different pairs of integers. Use a sentinel value on the outer loop to allow the user to terminate the entire process.

9.1) Which values of `year` cause the following loop to terminate with a correct answer?

```
/**
   Counts the number of years from a year input by the user
   until the year 3000.
*/
public class CountYears
{
   public static void main(String[] args)
   {
      int millennium = 3000;
      Scanner in = new Scanner(System.in);
      System.out.print("Please enter the current year: ");

      int year = in.nextInt();
      int nyear = year;

      while (nyear != millennium)
      {
         nyear++;
      }

      System.out.println("Another " + (nyear - year) + " years to the millennium.");
   }
}
```

9.2) Re-write the *while* loop so that it will not overflow `nyear`.

```
/**
   Counts the number of years from a year input by the user
   until the year 3000.
*/
public class CountYears
{
   public static void main(String[] args)
   {
      int millennium = 3000;
      Scanner in = new Scanner(System.in);
      System.out.print("Please enter the current year: ");

      int year = in.nextInt();
      int nyear = year;
```

```
      while (nyear != millennium)
      {
         nyear++;
      }

      System.out.println("Another " + (nyear - year) + " years to the millennium.");
   }
}
```

10.1) A variable that counts the iterations of a loop is called a *loop index* or *loop control variable*. In the preceding examples `nyear` served as an index, counting the number of years to the next millennium. This type of loop is frequently written using a `for` loop.

```
for (initialization; condition; update)
{
   statement
}
```

Write a program controlled by two (non-nested) *for* loops that produces the following listing of inclusive dates, from the fifth century B.C. through the fifth century A.D.

```
Century 5 BC  400-499
Century 4 BC  300-399
Century 3 BC  200-299
Century 2 BC  100-199
Century 1 BC  1-99
Century 1 AD  1-99
Century 2 AD  100-199
Century 3 AD  200-299
Century 4 AD  300-399
Century 5 AD  400-499
```

10.2) Write the `ListCenturies` program using a single loop `for (i = -5; i <= 5; i++)` with an `if` statement in the body of the loop.

11.1) One loop type might be better suited than another to a particular purpose. The following usages are idiomatic:

`for`	Known number of iterations
`while`	Unknown number of iterations
`do`	At least one iteration

Convert the following `while` loop to a `do` loop.

```
public class PrintSum
{
   public static void main(String[] args)
   {
```

```
      Scanner in = new Scanner(System.in);
      int sum = 0;
      int n = 1;

      while (n != 0)
      {
         System.out.print("Please enter a number, 0 to quit: ");
         n = in.nextInt();
         if (n != 0)
         {
            sum = sum + n;
            System.out.println("Sum = " + sum);
         }
      }
   }
}
```

11.2) Is the do loop in Lab 4.11.1 an improvement over the while loop? Why or why not?

12) Convert this `while` loop to a `for` loop.

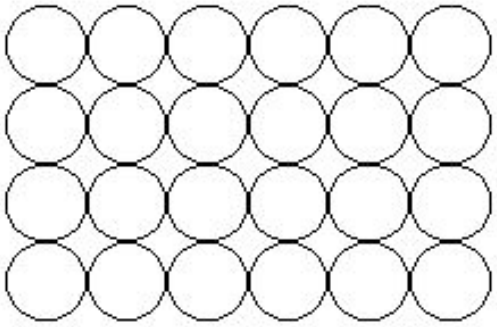
```
/**
   Program to compute the first integral power to which 2 can be
   raised that is greater than that multiple of a given integer.
*/
public class CountPowerOf2
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);
      System.out.print("Please enter a number, 0 to quit: ");
      int n = in.nextInt();

      int i = 1;
      while (n * n > Math.pow(2, i))
      {
         i++;
      }

      System.out.println("2 raised to " + i
            + " is the first power of two greater than " + n + " squared");
      }
   }
}
```

13) Convert this `for` loop to a `while` loop:

```
public static void main(String[] args)
{
   for (int i = 1; i <= 10; i++)
   {
```

```
      System.out.println(i + " squared equals " + i * i);
   }
}
```

14) Write a program to draw a top view of 24 soda cans—that is, draw 24 circles, arranged in a 4 x 6 grid like this:



Here is a partially completed program that you can use as a framework for this graphics lab. Add your drawing code to the `draw` method below.

```
import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JComponent;

public class SodaCanRunner
{
   public static void draw(Graphics g)
   {
      // Your code goes here

   }

   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

      final int FRAME_WIDTH = 400;
      final int FRAME_HEIGHT = 400;

      frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      JComponent component = new JComponent()
      {
         public void paintComponent(Graphics graph)
         {
            draw(graph);
         }
      };
```

```
      frame.add(component);
      frame.setVisible(true);
   }
}
```

15) To generate random numbers, you construct an object of the `Random` class, and then apply one of the following methods:

`nextInt(n)`: A random integer between the integers 0 (inclusive) and n (exclusive)
`nextDouble()`: A random floating-point number between 0 (inclusive) and 1 (exclusive)

Write a program that simulates the drawing of one card (for example, an ace of spades).