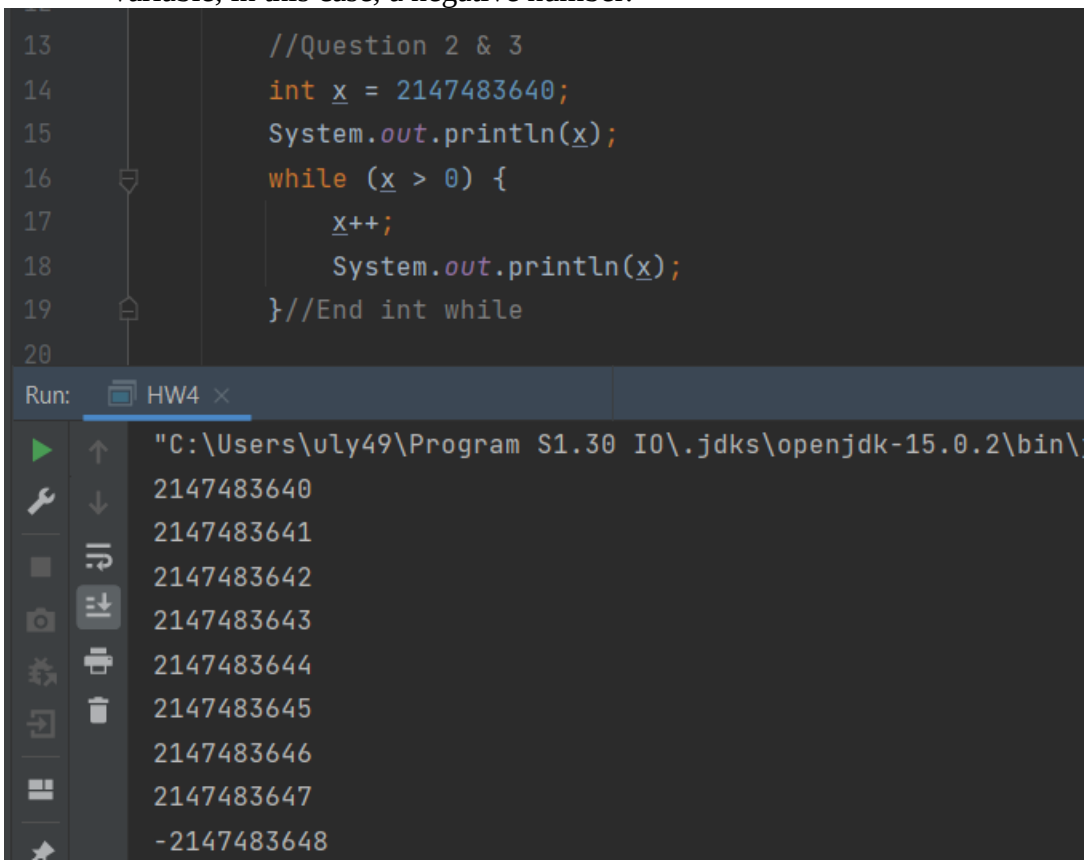
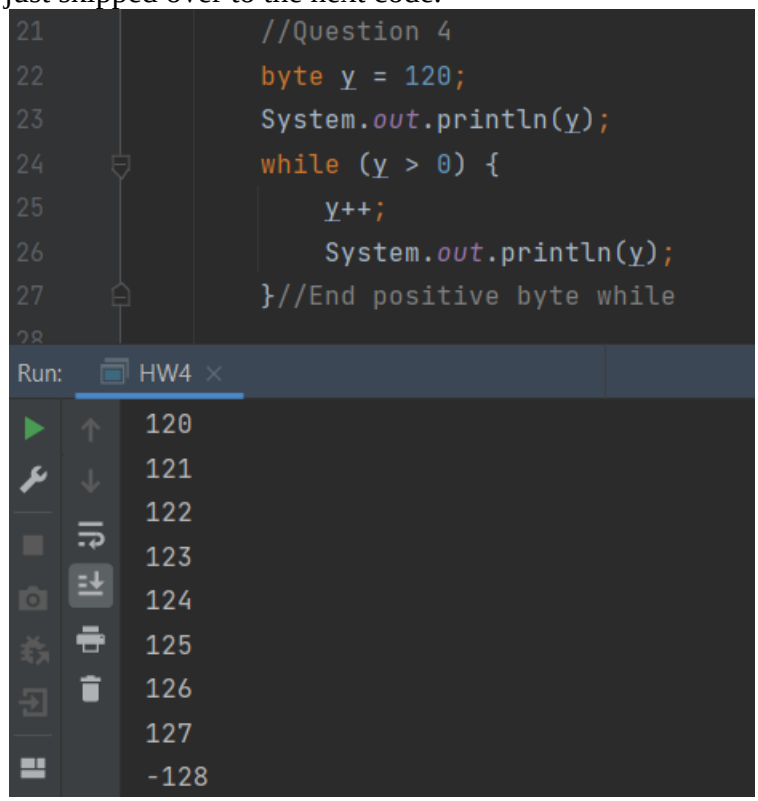
Integers

1) What programming language and machine are you doing this with?
    a. I am using Java through IntelliJ on my laptop. I don't know if you want to know specifically what machine, so my laptop is a MSI GF63 Thin 85C.
2) If you allocate an int in your programming language, initialize it to a large number, and increment it to overflow, how large is the int before it overflows? How many bits are allocated to the int? (It is 2 to what power?)
    a. I created a while loop to increment *x*, set to 2,147,483,640, by 1 while x was greater than 0. The program printed the sum inside the loop, showing the values as it incremented. The last value to print before the program finished running was -2,147,483,648. Before this overflow, the value printed was 2,147,483,999. In normalized binary form, this number is $1.11111111111110101110010011111 1 \times 2^{30}$. So, the total number bits allocated for this number was 31.
3) If you continually increment the int, do positive numbers go negative? Is there ever a trap to throw the program out? Do large negative numbers go positive? Be specific as to what happens.
    a. My program incremented 2,147,483,640 by 1indefinitely. The second to last value printed was 2,147,483,647. After this, -2,147,367,999 was printed. Then the program stopped. Int variables in Java hold 4 bytes (32 bits). It seems that the program can only hold decimal numbers that are no bigger than 1 minus the maximum number of bits able to be held in the variable. Once something triggers this "trap," it will print the lowest possible number able to be held in an int variable, in this case, a negative number.



4) If you create a byte and increment it, when and how does it overflow? What is number before and after overflow?
    a. It overflows after the decimal number 127. My program was set to increment a byte holding 120 by 1 and printing each sum. After it printed 127, it printed -128 and just skipped over to the next code.



5) If you create a byte and decrement it, how does it underflow? Do large negative numbers go positive?
    a. For this question, I essentially did the same thing. I set a byte variable to -120 and had it decrement by 1 and print each result. After it printed -127, it printed 128 and jumped out of the loop.

```
29          //Question 5
30          byte z = -120;
31          System.out.println(z);
32          while (z < 0) {
33              z--;
34              System.out.println(z);
35          }//End negative byte while
```

```
Run:  HW4
-120
-121
-122
-123
-124
-125
-126
-127
-128
127
```

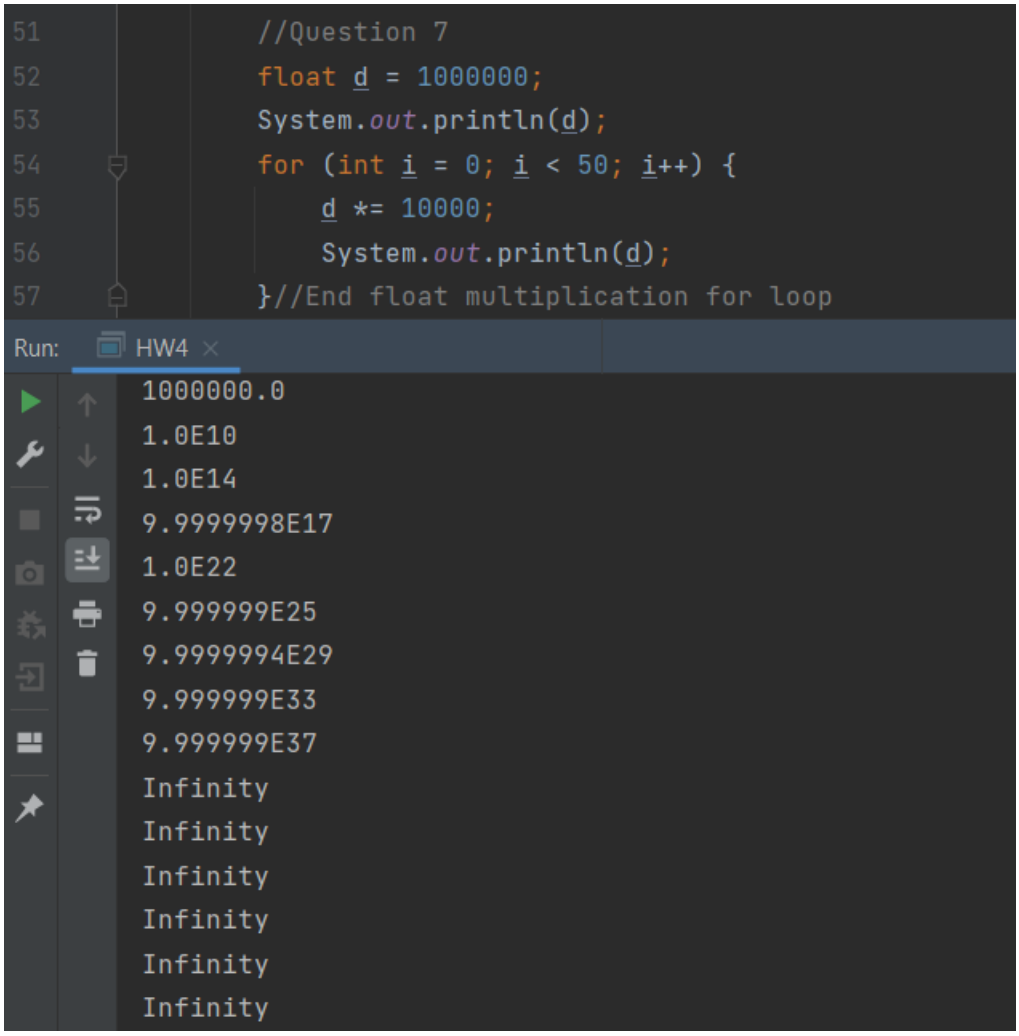
Floats & Doubles

6) Now work with floats and doubles. Enter 3 long fractional numbers (hardcoded) into a variable defined as a double. Print it out. Then save the double value to a float (type cast) and print that out. Show a *table* demonstrating the accuracy of these floats and doubles against the number you actually entered. In other words, demonstrate the differences between the accuracy of floats versus doubles.

Value entered to double variable	Value after typecasting to float variable
12.4567898765434567	12.45679
.732674868	0.7326749
4.74983783288331	4.749838

a.

7) Now work with overflows and underflows with floats. Contrast What happens if you continually multiply a large number by ten thousand? What happens if you continually divide a fraction by ten thousand? What is the smallest and largest possible actual numbers and what prints when you go beyond these numbers?
   a. I first set variable *d* to 1,000,000 and continually multiplied it by 10,000. I did this in a loop set to do this 50 times. I also had it print each product. Immediately after printing the first value, 1,000,000, it switched to scientific notation. The scientific notation only ever showed 6 to 7 digits after the decimal. After it printed 9.999999E37, the code still continued multiplying, but it only kept printing "infinity" until the loop ran out.



b. For the division, I set the variable *e* to 5.75 and had it divide by 10,000. I did this in a loop set to do this 50 times and had it print each result. The lowest value it printed was -5.7E44, after that, it kept printing “0.0” until the loop ran out.

```
58          float e = 5.75f;
59          System.out.println(e);
60          for (int i = 0; i < 50; i++) {
61              e /= 10000;
62              System.out.println(e);
63          }//End float division for loop
64
```

```
Run:  HW4
5.75
5.75E-4
5.7500003E-8
5.7500003E-12
5.75E-16
5.75E-20
5.75E-24
5.75E-28
5.75E-32
5.75E-36
5.75E-40
5.7E-44
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
```

8) Write a java (or other language) program that uses doubles to:
   - Print the number 0.333,333,33
   - Adds 0.333,333,33 to a total 100,000 times.
   - Multiplies 0.333,333,33 x 100,000

   Compare the sum and the multiplication result. Consider what is happening in the exponent and fraction in both operations.
   a. The sum, after adding .333,333,33 to .333,333,33 100,000 times, was 33333.666333313646. The product, after multiplying .333,333,33 by 100,000, was 33333.333.
9) Retest 8) by using 10,000,000 (instead of 100,000) times in the addition and multiplication. Compare the sum and the multiplication result. Why do they not match? Consider what is happening in the exponent and faction in both operations.

   a. The sum was 3333333.634158811. The product was 3333333.3. I'm not sure what exactly is happening. My best guess would be that when a number is multiplied by a multiple of 10, the program will just carry over the decimal by the corresponding number of zeros. So, the decimal doesn't change. But, when doing addition instead of multiplication, the program has to decide to round which, after rounding so many times, will result in a different decimal number.

10) With the above information, how should a web programmer take and process numeric input? Consider verifying that input is within a range or that sums or results are always larger than the input. What might be the advantages of these checks?
   a. It seems like the best option would be to test the input to see under which data type it would fall. For example, you could first save it to a double and then test to see how big or small the input is and then typecast it to that type of data type. This could help with keeping the exact value or result if you plan on doing anything to that value.