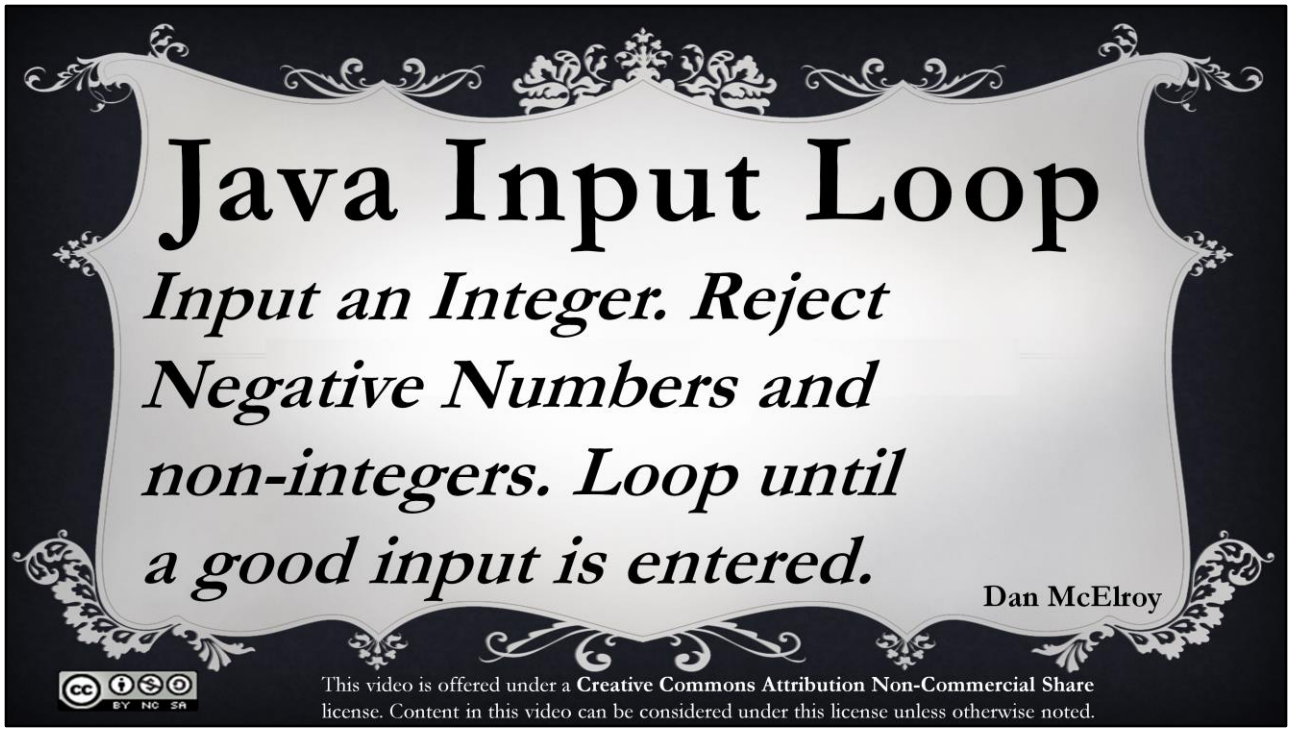

Hello programmers. This presentation discusses how to make a program input an integer and keep trying if a negative number or non-integers are entered.

Project Definition

Develop a program that inputs an integer, doubles it and displays the result. Negative values and data type mismatches are to be rejected. A type mismatch occurs if anything except an integer is entered.

Use a **do...while** loop to keep asking for an input until a positive integer is entered.

Develop a program that inputs an integer, doubles it and displays the result. Negative values and data type mismatches are to be rejected. A type mismatch occurs if anything except an integer is entered.

Use a **do...while** loop to keep asking for an input until a positive integer is entered.

Sample program execution

```
Enter a positive integer:  -5
Value must be positive. Try again:  3.6
Only integers are accepted. Try again:  asdf
Only integers are accepted. Try again:  42
The value 42 doubled is 84
```

Here is a sample program execution. The prompt says, "Enter a positive integer: ".
On my first try, I enter a negative 5. The program responds with, "Value must be positive. Try again: ".
Then I try entering 3.6 and the program responds with, "Only integers are accepted. Try again: ".
This time I'll see if I can really fake it out. I am entering the letters asdf.
The program responds again with, "Only integers are accepted. Try again: "I
I am being rejected by the program. It must not like me.
How rude. It had better accept the number 42.
Yeah. It worked and responded with, "The value 42 doubled is 84".

I could make a fortune selling this program as the latest app. I could advertise it as, "See what would happen if you doubled your income!"

Part 1 - Input an integer and double it

```
1  /*
2      This program inputs an integer, doubles it and displays the result.
3      Negative inputs and input mismatches are rejected. A type mismatch
4      occurs if anything except an integer is entered. A do...while loop is
5      used to keep asking for the input until a positive integer is entered.
6   */
7  package javainputloop;
8  import  java.util.Scanner; // used to read from keyboard
9
10 public class JavaInputLoop {
11     public static void main(String[] args) {
12         Scanner stdin = new Scanner (System.in);
13         int num1 = 0;    // the value input from the console
14         int sum;         // computed to be num*2 and then displayed
15
16         System.out.printf("Enter a positive integer: ");   // prompt
17         num1 = stdin.nextInt();    // input value from the keyboard
18
19         sum = num1 * 2;
20         System.out.printf ("The value %d doubled is %d\n", num1, sum);
21     } // end of main( )
22 } // end of class Java Input Loop
```

Let's start off with a simple program that only inputs an integer, doubles it and displays the result. I named the program, "JavaInputLoop".
he package name is javainputloop and does not use any capital letters.

The import java.util.Scanner; line gives the program the ability to read from the keyboard.
The class name is JavaInputLoop and the program will be in a file named "JavaInputLoop.java" .
Here is the start of the executable code defined by public static void main(String[ ] args) {
Create the scanner object. I named it stdin.
Declare two variables num1 and sum.
num1 will be used to hold the integer read from the

keyboard.
sum will hold the value from num1 multiplied by 2.
A System.out.printf command displays the prompt message.
num1 = stdin.nextInt( ); is used to read an integer from the keyboard.
Once we have an input for num1, it is doubled and assigned to the variable named sum.
Another System.out.printf statement displays the original value and the value that has been doubled.

Part 2 - Reject negative inputs

```
/*
    This program inputs an integer, doubles it and displays the result.
    Negative inputs and input mismatches are rejected. A type mismatch
    occurs if anything except an integer is entered. A do...while loop is
    used to keep asking for the input until a positive integer is entered.
 */
package javainputloop;
import  java.util.Scanner; // used to read from keyboard

public class JavaInputLoop {
    public static void main(String[] args) {
        Scanner stdin = new Scanner (System.in);
        int num1 = 0;   // the value input from the console
        int sum;        // computed to be num*2 and then displayed

        System.out.printf("Enter a positive integer: ");   // prompt
        num1 = stdin.nextInt();    // input value from the keyboard
        if (num1 < 0) {
            System.out.printf("Value must be positive. Try again: ");
        }
        else {
            // We get to this point when a positive integer has been entered
            sum = num1 * 2;
            System.out.printf ("The value %d doubled is %d\n", num1, sum);
        }
    } // end of main( )
} // end of class Java Input Loop
```

I can check for negative numbers right after doing the input from the keyboard.
I'm using an if / else construct.
The if tests for a negative input.
When the comparison expression in the if statement (num1 < 0) is true, the message "Value must be positive. Try again" is displayed.
Only when the if condition is false, the block of code attached to the else is executed.
Then, the sum is computed to be num1 * 2, and both numbers are displayed.

Part 3 - Reject non-integers

```
7   package javainputloop;
8   import  java.util.Scanner; // used to read from keyboard
9   import  java.util.InputMismatchException;
10
11  public class JavaInputLoop {
12      public static void main(String[] args) {
13          Scanner stdin = new Scanner (System.in);
            int num1 = 0;   // the value input from the console
15          int sum;        // computed to be num*2 and then displayed
16
17          System.out.printf("Enter a positive integer: ");   // prompt
18          try {
19              // Throw an exception if there is a data type mismatch
20              num1 = stdin.nextInt();    // input value from the keyboard
21
22              if (num1 < 0) {
23                  System.out.printf("Value must be positive. Try again: ");
24              }
25              else {   // SUCCESS - compute and display the result
26                  sum = num1 * 2;
27                  System.out.printf ("The value %d doubled is %d\n", num1, sum);
28              }
29          }
30          catch (InputMismatchException e) {
31              stdin.nextLine( );   // get rid of the rest of the line
32              System.out.printf("Only integers are accepted. Try again: ");
33          }
34      } // end of main( )
35  } // end of class Java Input Loop
```

Java implements an object oriented programming technique called exception handling. It does this with try / throw / catch blocks of code. Sections of code that can cause an error are placed in a 'try' block. When an error is detected, a 'throw' operation is used to jump out of the middle of the 'try' block directly to the 'catch' block. Some operations have an automatic 'throw' built in, such as the scanner's detection of a data type mismatch. Other times, the programmer is responsible for doing the 'throw' from inside the 'try' block.

In this example, the scanner's keyboard input and processing for negative input values are placed inside the try block.

If the user would type in a floating point number or even the letters t-e-n instead of an integer, the scanner would

detect an 'InputMismatchException' and immediately jump to the catch block without even testing for a negative value. The 'catch' statement identifies the type of exception it is willing to handle.
Additional information about the exception is provided in a character string that can be used within the catch block.
It is common to give this string the name 'e' or 'ex'. I am not using that extra information in this program.

The first line in the catch block is stdin.nextLine( );
This is why the line is needed. The scanner was previously attempting to read an integer, but when it finds something that is not an integer, it stops removing characters from the keyboard's input buffer. If the user typed in the characters t-e-n, they would still be in the keyboard's input buffer. So far, the program does not have a loop to keep reading data from the scanner. But when the program is updated to use a loop, the non-integer data would still be left in the buffer just in case a different call to scanner would try to process it.

Unfortunately, this program will be looping back again to try and read an integer, but the bad data is still in the buffer. The program would be left in an infinite loop with scanner looping to read an integer with the non-integer data still in the input buffer. The solution is to use stdin.nextLine( ) to read the rest of the input buffer to clear it. Since there is not an assignment statement for the stdin.nextLine( );, the remaining data from the input buffer is discarded. Now we don't need to worry about trying to process left over and

unwanted keyboard data. The next line in the 'catch' block displays, "Only integers are accepted. Try again: "

The program needs additional code to process the "InputMismatchException".
This code is brought into the program by the import Java.util.InputMismatchException; at the top of the program.

Part 4 - The **do…while** loop

```
int x = 0;
do {
     System.out.printf("x = %d\n", x);
     x++;
 } while (x < 5);
System.out.println("Done");
```

Console output

```
x = 0
x = 1
x = 2
x = 3
x = 4
Done
```

Java has several ways of implementing loops.
Code inside a loop is executed repeatedly until some condition occurs that ends the loop.
When the loop ends, the line of code after the loop is executed next.
One of the most important things that needs done is to make sure that there is a condition that will cause the loop to end.
Otherwise, the program can get caught in an infinite loop and continue to loop until someone kills the program.
Here is the do…while loop. Notice that there is a conditional expression that belongs to the while statement.
Since the while statement is at the bottom of the loop, the block of code for the do…while loop is guaranteed to execute at least one time.

In this code fragment, the integer x is declared and initialized to 0. The block of code in the do…while loop starts by displaying, "x = 0".
Then x is incremented to a 1. A test is made in the while statement. Since 1 is less than 5, the program jumps back to the top of the loop.
The program now displays, "x = 1" and increments x to a 2. The code continues to loop until finally, "x = 4" is displayed and x is incremented to a 5.
Since x is no longer less than 5, the conditional expression that belongs to the while statement evaluates to false and the loop is ended.
The printf statement after the loop is executed and it displays, "Done".

I want the program to display an error message and ask for the input again as long as the user is typing in negative numbers or non-integer values.
I am going to do this by using a do…while loop. Inside the body of the do…while loop, I am going to place all the code that does the console input, tests for invalid values, and associated error messages. I will make the program to continue to loop until good data is entered.

Part 5 - Loop if bad data is entered

```
11  public class JavaInputLoop {
12      public static void main(String[] args) {
13          Scanner stdin = new Scanner (System.in);
14          int num1 = 0;   // the value input from the console
15          int sum ;       // computed to be num*2 and then displayed
16          boolean keepTrying; // flag when true keeps the loop going
17
18          System.out.printf("Enter a positive integer: ");   // prompt
19          do {
20              keepTrying = false; // assume no error
21              try {
22                  // Throw an exception if there is a data type mismatch
23                  num1 = stdin.nextInt();    // input value from the keyboard
24
25                  if (num1 < 0) {
26                      System.out.printf("Value must be positive. Try again: ");
27                      keepTrying = true;
28                  }
29              }
30              catch (InputMismatchException e) {
31                  stdin.nextLine( );   // get rid of the rest of the line
32                  System.out.printf("Only integers are accepted. Try again: ");
33                  keepTrying = true;
34              }
35          } while (keepTrying);
36          // We get to this point when a positive integer has been entered
37          sum = num1 * 2;
38          System.out.printf ("The value %d doubled is %d\n", num1, sum);
39      } // end of main( )
40  } // end of class Java Input Loop
```

Here is the full implementation of the check for negative numbers, non-integer inputs, and a loop.

The do…while loop is has the blocks of code for 'try' and 'catch' to keep inputting data from the keyboard until a good value has been entered. One thing is a little different from the previous version, is that the computation of the sum and its display are now outside the loop instead of being part of an else block that was executed when a non-negative value was received. Now, the do…while is only being used for input.

It is simpler to code by using a flag when there are two different cases that are being tested: negative-values and non-integer values.

I am using a boolean variable named 'keepTrying' as a flag. to identify when to loop or not to loop. I could have chosen

all sorts of other meaningful names such as, tryAgain, repeatUntilGoodDataIsEntered, thatDidntLookGood. Think of a flag on a mailbox. It is placed UP to let the letter carrier know there is mail to be picked up, otherwise it is set DOWN.
The program uses a prompt to request a positive integer. The first thing that happens in the 'try' block is the flag is set to false. I am going to assume there is no error unless one is detected. An attempt is made to read an integer. I see that the flag is set to 'true' if a negative number is entered or there is a mismatchException. The while statement tests the flag and loops back to the top of the loop if the flag is 'true' because one of those conditions exists. The flag is set back to false again at the top of the loop. Eventually when the user enters good data, the flag remains 'false' and the loop ends. The sum is computed as num1 * 2, the output message is displayed and the program ends.

This is the version of the code that is being placed on the class Canvas website and in the comments section for the YouTube video.

I hope you found this information very useful. You can modify the code to process other data types such as double. The way it is currently written, you would need to duplicate a lot of code if you needed to input and validate many values. Instead of duplicating code over and over again, a function/subroutine could be used to hold the code that loops until good data is received. Then you could just call the function with a single line of code each time a new input was needed. Maybe I will show that in another presentation.

I'll see you around. Until then, Bye.