Motivating problem

- Suppose you wanted to print a multiplication table of the sort you had to memorize as a child

- Each line and column of the table has a number as its heading; the entries at each row/column intersection are the results when the row heading is multiplied by the column heading

Multiplication table program output

```
1    2    3    4    5    6    7    8    9    10
2    4    6    8    10   12   14   16   18   20
3    6    9    12   15   18   21   24   27   30
4    8    12   16   20   24   28   32   36   40
5    10   15   20   25   30   35   40   45   50
6    12   18   24   30   36   42   48   54   60
7    14   21   28   35   42   49   56   63   70
8    16   24   32   40   48   56   64   72   80
9    18   27   36   45   54   63   72   81   90
10   20   30   40   50   60   70   80   90   100
```

Nested `for` loops

```
for (int i = 1; i <= 10; i++) {



}
```

A **`for`** loop can contain any kind of statement in its body, including another **`for`** loop.

Nested `for` loops

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 10; j++) {
        out.print(i*j + "\t");
    }
    out.println();
}
```

The inner loop must have a different name for its loop counter variable so that it will not conflict with the outer loop.

Nested `for` loops

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 10; j++) {
        out.print(i*j + "\t");
    }
    out.println();
}
```

Loops, when placed inside one another creating a loop of loops, are called ***nested loops***.

Nested `for` loops

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 10; j++) {
        out.print(i*j + "\t");
    }
    out.println();
}
```

The outer loop is executed as expected.
This i loop executes ten times.

Nested `for` loops

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 10; j++) {
        out.print(i*j + "\t");
    }
    out.println();
}
```

All of the statements in the outer loop's body are executed on each iteration of the outer loop.

Nested `for` loops

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 10; j++) {
        out.print(i*j + "\t");
    }
    out.println();
}
```

Each time the outer i loop repeats, the inner j loop starts over again from 1.

Nested `for` loops

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 10; j++) {
        out.print(i*j + "\t");
    }
    out.println();
}
```

The outer i loop runs 10 times.
For **each** of those times the inner j loop runs 10 times, for a total of 100 numbers printed.

Nested `for` loops

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= i; j++) {
        out.print(i*j + "\t");
    }
    out.println();
}
```

Consider this very slight change to the inner j loop…

Nested `for` loops

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= i; j++) {
        out.print(i*j + "\t");
    }
    out.println();
}
```

Instead of executing 10 times, the j loop now executes i times, where i is the outer loop’s counter variable.

Modified program output

```
1
2       4
3       6       9
4       8       12      16
5       10      15      20      25
6       12      18      24      30      36
7       14      21      28      35      42      49
8       16      24      32      40      48      56      64
9       18      27      36      45      54      63      72      81
10      20      30      40      50      60      70      80      90    100
```

Triangular `for` loops

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= i; j++) {
        out.print(i*j + "\t");
    }
    out.println();
}
```

Such a loop nest is called a "*triangular loop*" due to the pattern of the iteration space as seen on the prior slide.

How would we generate this?

```
1       2       3       4       5       6       7       8       9       10
2       4       6       8       10      12      14      16      18
3       6       9       12      15      18      21      24
4       8       12      16      20      24      28
5       10      15      20      25      30
6       12      18      24      30
7       14      21      28
8       16      24
9       18
10
```

How would we generate this?

```
1      2      3      4      5      6      7      8      9      10
2      4      6      8      10     12     14     16     18
3      6      9      12     15     18     21     24
4      8      12     16     20     24     28
5      10     15     20     25     30
6      12     18     24     30
7      14     21     28
8      16     24
9      18
10
```

In this case we want 10 items on the first line, 9 on the next line, 8 on the line after that, etc.

Triangular `for` loops

```
for (int i = 1; i <= 10; i++) {
    for (int j = 10; j >= i; j--) {
        out.print(i*j + "\t");
    }
    out.println();
}
```

It appears that we simply need to make the j loop count down.

Triangular `for` loops

```
for (int i = 1; i <= 10; i++) {
    for (int j = 10; j >= i; j--) {
        out.print(i*j + "\t");
    }
    out.println();
}
```

It appears that we simply need to make the j loop count down. But that also changes the values printed. Here is the wrong output:

This is not what we wanted:

```
10    9     8     7     6     5     4     3     2     1
20    18    16    14    12    10    8     6     4
30    27    24    21    18    15    12    9
40    36    32    28    24    20    16
50    45    40    35    30    25
60    54    48    42    36
70    63    56    49
80    72    64
90    81
100
```

Triangular `for` loops

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 11-i; j++){
        out.print(i*j + "\t");
    }
    out.println();
}
```

So we make the j loop still count up, but change the limit to be smaller each time by subtracting i.

Drawing ASCII art

- Drawing figures with ASCII characters can illustrate how nested loops work

- Keep in mind the principle: the outer loop controls the number of lines, while the inner loops control the content of lines

Drawing ASCII art

Let’s draw an equilateral triangle whose size is determined by the user.

Here is a size 4 triangle:

```
   *
  * *
 * * *
* * * *
```

Here is a size 6 triangle:

```
     *
    * *
   * * *
  * * * *
 * * * * *
* * * * * *
```

Drawing ASCII art

To draw an equilateral triangle we need:

Drawing ASCII art

To draw an equilateral triangle we need:

- An outer loop to control the number of lines printed.

Drawing ASCII art

To draw an equilateral triangle we need:

- An outer loop to control the number of lines printed.
- The body of the outer loop will then consist of:
  - A triangular loop to printed some spaces, with the number of spaces decreasing on each line.

Drawing ASCII art

To draw an equilateral triangle we need:

- An outer loop to control the number of lines printed.
- The body of the outer loop will then consist of:
  - A triangular loop to printed some spaces, with the number of spaces decreasing on each line.
  - Followed by another triangular loop to print the `*`'s, with the number of `*`'s increasing on each line.

Drawing ASCII art

```
int height = out.getHeight();
// draw "height" number of lines
for (int i=1; i<=height; i++) {









}
```

Drawing ASCII art

```
int height = out.getHeight();
// draw "height" number of lines
for (int i=1; i<=height; i++) {
    // draw decreasing number of spaces
    for (int j=height; j>i; j--) {
        out.print(" ");
    }





}
```

Drawing ASCII art

```
int height = out.getHeight();
// draw "height" number of lines
for (int i=1; i<=height; i++) {
    // draw decreasing number of spaces
    for (int j=height; j>i; j--) {
        out.print(" ");
    }
    // draw increasing number of stars
    for (int j=1; j<=i; j++) {
        out.print("* ");
    }


}
```

Drawing ASCII art

```
int height = out.getHeight();
// draw "height" number of lines
for (int i=1; i<=height; i++) {
    // draw decreasing number of spaces
    for (int j=height; j>i; j--) {
        out.print(" ");
    }
    // draw increasing number of stars
    for (int j=1; j<=i; j++) {
        out.print("* ");
    }
    // end the current line
    out.print("\n");
}
```

Common errors

- Both of the following sets of code produce *infinite loops*:

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; i <= 10; j++) {//test wrong variable
        out.print(i*j + "\t");
    }
    out.println();
}

for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 10; i++) {//update wrong variable
        out.print(i*j + "\t");
    }
    out.println();
}
```