Recap: Assignment as an Operator

- You can consider assignment as an operator, with a lower precedence than the arithmetic operators.
  
  \[
  \text{answer} = \frac{\text{sum}}{4} + \text{MAX} \times \text{lowest};
  \]

  Then the result is stored in the variable on the left hand side, overwritten previous value.

Repetition with `for` Loops

- Java’s `for` loop statement performs a task many times.
  
  ```java
  preheatOven();
mixBatter();
  for (int i = 1; i <= 4; i++) {
    // repeat 4 times
    bake();
  }
decorate();
  ```

  ```java
  preheatOven();
mixBatter();
bake();
bake();
bake();
bake();
decorate();
  ```

  `for` loop syntax

  ```java
  for (initialization; test; update) {
    statement;
    statement;
    ... statement;
  }
  ```

  - Perform initialization once.
  - Repeat the following:
    - Check if the test is true. If not, stop.
    - Execute the statements.
    - Perform the update.
The for Statement: Syntax

Reserved word

The initialization portion is executed once before the loop begins
The statement is executed until the condition becomes false
for ( initialization ; condition ; increment )
statement;

Both semi-colons are always required
The increment portion is executed at the end of each iteration

Flowchart of a for loop

for ( initialization ; condition ; increment )
statement;

 Initialization

for (int i = 1; i <= 6; i++) {
    System.out.println("I am so smart");
}

- Tells Java what variable to use in the loop
  - The variable is called a loop counter
    - can use any name, not just i
    - can start at any value, not just 1
    - only valid in the loop
  - Performed once as the loop begins

Test

for (int i = 1; i <= 6; i++) {
    System.out.println("I am so smart");
}

- Tests the loop counter variable against a limit
  - Uses comparison operators:
    - < less than
    - <= less than or equal to
    - > greater than
    - >= greater than or equal to
Update

```java
for (int i = 1; i <= 6; i++) {
    System.out.println("I am so smart");
}
```

- Typically changes the loop counter variable

Increment and decrement

**shortcuts to increase or decrease a variable’s value by 1**

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent longer version</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable++</td>
<td>variable = variable + 1;</td>
</tr>
<tr>
<td>variable--</td>
<td>variable = variable - 1;</td>
</tr>
</tbody>
</table>

```java
int x = 2;
x++; // x = x + 1;
// x now stores 3
```

```java
double gpa = 2.5;
gpa--; // gpa = gpa - 1;
// gpa now stores 1.5
```

Modify-and-assign operators

**shortcuts to modify a variable’s value**

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent longer version</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable += value;</td>
<td>variable = variable + value;</td>
</tr>
<tr>
<td>variable -= value;</td>
<td>variable = variable - value;</td>
</tr>
<tr>
<td>variable *= value;</td>
<td>variable = variable * value;</td>
</tr>
<tr>
<td>variable /= value;</td>
<td>variable = variable / value;</td>
</tr>
<tr>
<td>variable %= value;</td>
<td>variable = variable % value;</td>
</tr>
</tbody>
</table>

```java
x += 3; // x = x + 3;
gpa -= 0.5; // gpa = gpa - 0.5;
number *= 2; // number = number * 2;
```

Simplification

- Since such update expressions are so common:

  ```java
  count = count + increment;
  count += increment;
  ```

  - when increment is 1

    ```java
    count ++;
    ```
More Assignment-related Operators

- Increment and decrement operators: `++`, `--`
- Assignment operators: `+=`, `-=`, `*=`, `/=`

```
these three expressions have the same effect

```
count = count + 1;
count += 1;
count ++;
```

```
these two expressions have the same effect

```
count = count - 10;
count -= 10;
```

Example: Repetition over a range

System.out.println("1 squared = " + 1 * 1);
System.out.println("2 squared = " + 2 * 2);
System.out.println("3 squared = " + 3 * 3);
System.out.println("4 squared = " + 4 * 4);
System.out.println("5 squared = " + 5 * 5);
System.out.println("6 squared = " + 6 * 6);

Intuition: "I want to print a line for each number from 1 to 6"

The for loop does exactly that!

```
for (int i = 1; i <= 6; i++) {
    System.out.println(i + " squared = " + (i * i));
}
```

Intuition: "For each integer i from 1 through 6, print ..."

Loop Walkthrough

```
for (int i = 1; i <= 4; i++) {
    System.out.println(i + " squared = " + (i * i));
}
System.out.println("Whoo!");
```

Output:
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
Whoo!
### Multi-line Loop Body

```java
System.out.println("++++++");
for (int i = 1; i <= 3; i++) {
    System.out.println("\    /");
    System.out.println("/    \");
}
System.out.println("++++++");
```

**Output:**
```
+----+
\    /
/\   \ /
\  / \\
\ /   \
+/     +----+
```

### Counting Down

- There is much flexibility in initialization and update: instead of increasing the loop counter, you can decrease it.
- Exercise: write a program generating output
  
  T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!

```java
for (initialization; test; update) {
    statement;
    statement;
    ...
    statement;
}
```

- The `update` uses `--` to make the loop count down.
- The `test` must say `>` instead of `<.

```java
System.out.print("T-minus ");
for (int i = 10; i >= 1; i--) {
    System.out.print(i + ", ");
}
System.out.println("blastoff!");
```

### Counting Down: Exercise

- Revise the program so that
  - you have to increase the counter `i` from 1 to 10 but still produce the same output:
    - T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!

```java
System.out.print("T-minus ");
for (int i = 1; i <= 10; i++) {
    // ???
}
System.out.println("blastoff!");
```
Counting Down: Revision

- The code has the “magic” number 11, whose relationship with 10 is not represented:

```java
System.out.print("T-minus ");
for (int i = 1; i <= 10; i++) {
    System.out.print(11-i + ", ");
}
System.out.println("blastoff!");
```

Counting Down: Revision

- If I want to count down from 12, what should I change?

```java
T-minus 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!
```

Counting Down: Revision

- Redundant specification:

```java
System.out.print("T-minus ");
for (int i = 1; i <= 10; i++) {
    System.out.print(11-i + ", ");
}
System.out.println("blastoff!");
```

Counting Down: Revision

```java
int N = 10;
System.out.print("T-minus ");
for (int i = 1; i <= N; i++) {
    System.out.print(N+1-i + ", ");
}
System.out.println("blastoff!");
```
Constants

- A “constant variable” is an identifier that is similar to a variable except that it holds one value for its entire existence
  - why constants?

- In Java, we use the final modifier to declare a variable constant, and the convention is to use all capital words to name a constant, e.g.,
  ```java
  final int MONTHES_PER_YEAR = 12;
  ```

- The compiler will issue an error if you try to change the value of a constant variable after it is set

Constant

- Use keywords to tell computer your intention
- If there is a final before a variable declaration, it is your promise to the computer that you will not modify it after declaration
- If you break your promise, the compiler will catch you

```java
final int N = 10;
System.out.print("T-minus ");
for (int i = 1; i <= N; i++) {
    System.out.print(N+1-i + ", ");
}
System.out.println("blastoff!");
```

Variable Scope

- scope: The part of a program where a variable exists.
  - From its declaration to the end of the {} braces
    - A variable declared in a for loop exists only in that loop.
    - A variable declared in a method exists only in that method.

```java
public static void example() {
    int x = 3;
    for (int i = 1; i <= 10; i++) {
        System.out.println(x);
    }
    // ...
} // end of example
```
Why Scope?

- Encapsulation
  - e.g., different methods can use the same variable name without the need for coordination
  - many analogies: folders allow same file name so long in different folders

```
public static void aMethod()
{
    int x = 1;
    ...
}

public static void bMethod()
{
    int x = 2;
    ...
}
```

Scope Implications

- Variables without overlapping scope can have same name:

```
public static void Countdown2() {
    for (int i = 1; i <= 10; i++) {
        System.out.print( 11-i );
    }
    System.out.println("Count down again");
    for (int i = 1; i <= 10; i++) {
        System.out.print(11 - i);
    }
    int i = 5;
}
```

Scope Implications

- In a method, a variable can't be declared twice or used out of its scope.

```
for (int i = 1; i <= 100 * line; i++) {
    int i = 2;
    // ERROR: overlapping scope
    System.out.print("/");
} i = 4;
// ERROR: outside scope
```

Class Scope

```
public class CountDown {
    static int i
    public static void main(String[] args) {
        System.out.print("T-minus ");
        for (i = 1; i <= 10; i++) {
            System.out.print( 11-i );
        }
        System.out.println("blastoff!");
        System.out.println("Final count: " + i);
    } // end of main
} // end of class
```
```java
public class CountDown {
    static int N = 10;
    public static void main(String[] args) {
        System.out.print("T-minus ");
        for (int i = 1; i <= N; i++) {
            System.out.print(N+1-i + ", ");
        }
        System.out.println("blastoff!");
    }
}
```

```java
public static void main(String[] args) {
    System.out.print("T-minus ");
    for (int i = 1; i <= N; i++) {
        System.out.println(N+1-i + ", ");
    }
    System.out.println("blastoff!");
}
```

---

**Code Beauty Contest**: Which One’s Better?

**Nested Loops**

- **Nested loop**: A loop placed inside another loop.
  ```java
  for (int i = 1; i <= 5; i++) {
      for (int j = 1; j <= 10; j++) {
          System.out.print("*");
      }
      System.out.println(); // to end the line
  }
  ```
  - **Output**: 
    ```
    **********
    **********
    **********
    **********
    **********
    ```
  - **The outer loop repeats 5 times; the inner one 10 times.**
    - "sets and reps" exercise analogy

---

**Nested for loop exercise**

- **What is the output of the following nested for loops?**
  ```java
  for (int i = 1; i <= 5; i++) {
      for (int j = 1; j <= i; j++) {
          System.out.print("*");
      }
      System.out.println();
  }
  ```
  - **Output**: 
    ```
    *
    *
    *
    *
    ```

---

**Nested Loops**

- **Nested loop**: A loop placed inside another loop.
  ```java
  for (int i = 1; i <= 5; i++) {
      for (int j = 1; j <= 10; j++) {
          System.out.print("*");
      }
      System.out.println(); // to end the line
  }
  ```
  - **Output**: 
    ```
    **********
    **********
    **********
    **********
    **********
    ```
  - **The outer loop repeats 5 times; the inner one 10 times.**
    - "sets and reps" exercise analogy
What is the output of the following nested for loops?

```java
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

Output:
*  
**  
***  
****  
*****

What is the output of the following nested for loops?

```java
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(1);
    }
    System.out.println();
}
```

Output:
1  
22  
333  
4444  
55555

What is the output of the following nested for loops?

```java
for (int i = 1; i <= 9; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(i*j + 1000);
    }
    System.out.println();
}
```

Output:

```
1
2
3
4
5
6
7
8
9
```
Common Errors

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

```java
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.print("*");
    }
    System.out.println();
}
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 10; i++) {
        System.out.print("*");
    }
    System.out.println();
}
```

Complex lines

- What nested for loops produce the following output?

```
outer loop (loops 5 times because there are 5 lines)

inner loop (repeated characters on each line)

1
..2
...3
...4
....5
```

- We must build multiple complex lines of output using:
  - an outer "vertical" loop for each of the lines
  - inner "horizontal" loop(s) for the patterns within each line

Outer and inner loop

- First write the outer loop, from 1 to the number of lines.

```java
for (int line = 1; line <= 5; line++) {
   ...
}
```

- Now look at the line contents. Each line has a pattern:
  - some dots (0 dots on the last line), then a number

```
....1
...2
...3
...4
....5
```

- Observation: the number of dots is related to the line number.

Nested for loop exercise

- Make a table to represent any patterns on each line.

<table>
<thead>
<tr>
<th>line</th>
<th># of dots</th>
<th>-1 * line + 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Nested for loop solution

Answer:
```java
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (-1 * line + 5); j++) {
        System.out.print(".");
    }
    System.out.println(line);
}
```

Output:
```
....1
...2
..3
.4
5
```

Fixing for Consistency

```java
int N = 5;
for (int line = 1; line <= N; line++) {
    for (int j = 1; j <= (-1 * line + N); j++) {
        System.out.print(".");
    }
    System.out.println(line);
}
```

Nested for loop exercise

What is the output of the following nested for loops?
```java
int N = 5;
for (int line = 1; line <= N; line++) {
    for (int j = 1; j <= (-1 * line + N); j++) {
        System.out.print(".");
    }
    for (int k = 1; k <= line; k++) {
        System.out.println(line);
    }
}
```

Answer:
```
....1
...2
..3
.4444
55555
```
Nested for loop exercise

Modify the code below to produce this output:

....1
....2.
..3..
.4...
5....

int N = 5;
for (int line = 1; line <= N; line++) {
    for (int j = 1; j <= (-1 * line + N); j++) {
        System.out.print(".");
    }
    for (int k = 1; k <= line; k++) {
        System.out.print(line);
    }
    System.out.println();
}

Nested for loop exercise

Modify the previous code to produce this output:

....1
....2.
..3..
.4...
5....

Answer:

int N = 5;
for (int line = 1; line <= N; line++) {
    for (int j = 1; j <= (-1 * line + N); j++) {
        System.out.print(".");
    }
    System.out.print(line);
    for (int j = 1; j <= (line - 1); j++) {
        System.out.print(".");
    }
    System.out.println();
}

Reuse the Code

Printing a diagonal matrix is a useful task:

....1
....2.
..3..
.4...
5....

How do we reuse the code for different N?

int N = 5;
for (int line = 1; line <= N; line++) {
    for (int j = 1; j <= (-1 * line + N); j++) {
        System.out.print(".");
    }
    System.out.print(line);
    for (int j = 1; j <= (line - 1); j++) {
        System.out.print(".");
    }
    System.out.println();
}

Recap: Variable Scope

Two types of variables

- **Class variables**: declared inside a class but not any method
  - class variable’s scope is the whole class
  - cover more later in the course
- **Local variables**: declared in a method of a class
  - a local variable starts to exist at the point it is declared and disappears at the end of enclosing block
  - we say that the variable is in scope in this range
  - if a local variable x is in scope, you can use x directly in any statement, e.g., if integer x is in scope, you can write
    System.out.println(x);
Local Variable Scoping Rule

thisMethodName()
{ // block 1
    int x = 2; // x is in scope until end of block 1
    // ...
    { // block 2
        // x is also in scope in the nested block
        int y = 3; // now y in scope until end of block 2
        // int x = 4; // is this OK?
    } // end of block 2
} // end of block 1