Now we will examine some other program statements.

Chapter 3 focuses on:

- program development stages
- the flow of control through a method
- decision-making statements
- expressions for making complex decisions
- repetition statements
- drawing with conditionals and loops

Let’s skip to here to work on your 1st assignment: Repetition Statements
A *conditional statement* lets us choose which statement will be executed next.

Therefore they are sometimes called *selection statements*.

Conditional statements give us the power to make basic decisions.

Java's conditional statements are:
- the *if statement*
- the *if-else statement*
- the *switch statement*
The if Statement

- The *if statement* has the following syntax:

```java
if ( condition )
statement;
```

- The *condition* must be a boolean expression. It must evaluate to either true or false.

- *if* is a Java reserved word.

- If the *condition* is true, the *statement* is executed. If it is false, the *statement* is skipped.
An example of an if statement:

```java
if (sum > MAX)
    delta = sum - MAX;
System.out.println ("The sum is " + sum);
```

First, the condition is evaluated. The value of `sum` is either greater than the value of `MAX`, or it is not.

If the condition is true, the assignment statement is executed. If it is not, the assignment statement is skipped.

Either way, the call to `println` is executed next.

See [Age.java](#) (page 135)
Logic of an if statement

condition evaluated

true
statement

false
A condition often uses one of Java's equality operators or relational operators, which all return boolean results:

- `==` equal to
- `!=` not equal to
- `<` less than
- `>` greater than
- `<=` less than or equal to
- `>=` greater than or equal to

Note the difference between the equality operator (==) and the assignment operator (=)
The if-else Statement

- An *else clause* can be added to an *if* statement to make an *if-else statement*

```java
if ( condition )
    statement1;
else
    statement2;
```

- If the *condition* is true, *statement1* is executed; if the condition is false, *statement2* is executed

- One or the other will be executed, but not both

- See *Wages.java* (page 139)
Logic of an if-else statement

- Condition evaluated
  - True
    - statement1
  - False
    - statement2
Block Statements

- Several statements can be grouped together into a block statement.
- A block is delimited by braces: `{ ... }`
- A block statement can be used wherever a statement is called for by the Java syntax.
- For example, in an if-else statement, the if portion, or the else portion, or both, could be block statements.
- See Guessing.java (page 141)
Nested if Statements

- The statement executed as a result of an if statement or else clause could be another if statement.
- These are called *nested if statements*.
- See MinOfThree.java (page 143).
- An else clause is matched to the last unmatched if (no matter what the indentation implies).
- Braces can be used to specify the if statement to which an else clause belongs.
Logical Operators

- Boolean expressions can use the following *logical operators*:
  - `!` Logical NOT
  - `&&` Logical AND
  - `||` Logical OR

- They all take boolean operands and produce boolean results

- Logical NOT is a unary operator (it operates on one operand)

- Logical AND and logical OR are binary operators (each operates on two operands)
Logical NOT

- The *logical NOT* operation is also called *logical negation* or *logical complement*.
- If some boolean condition \( a \) is true, then \( \neg a \) is false; if \( a \) is false, then \( \neg a \) is true.
- Logical expressions can be shown using *truth tables*.

<table>
<thead>
<tr>
<th>( a )</th>
<th>( \neg a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
The *logical AND* expression

```
a && b
```

is true if both \( a \) and \( b \) are true, and false otherwise

The *logical OR* expression

```
a || b
```

is true if \( a \) or \( b \) or both are true, and false otherwise
A truth table shows the possible true/false combinations of the terms since `&&` and `||` each have two operands, there are four possible combinations of conditions `a` and `b`.

|    |    | `a && b` | `a || b` |
|----|----|----------|----------|
| true | true | true     | true     |
| true | false | false    | true     |
| false | true | false    | true     |
| false | false | false    | false    |
### Truth Tables

|    |    | a && b | a || b | ? | ? |
|----|----|--------|--------|---|---|
| true | true | true   | true   | t | f |
| true | false| false  | true   | f | t |
| false| true | false  | true   | t | t |
| false| false| false  | false  | t | t |

\(a \parallel b\) (also \(a \rightarrow b\))

\(!a \parallel !b\)
Logical Operators

- Conditions can use logical operators to form complex expressions

```java
if (total < MAX+5 && !found)
    System.out.println("Processing..."n);
```

- Logical operators have precedence relationships among themselves and with other operators
  - all logical operators have lower precedence than the relational or arithmetic operators
  - logical NOT has higher precedence than logical AND and logical OR
Short Circuited Operators

- The processing of logical AND and logical OR is “short-circuited”
- If the left operand is sufficient to determine the result, the right operand is not evaluated

```java
if (count != 0 && total/count > MAX)
    System.out.println("Testing...");
```

- This type of processing must be used carefully
Specific expressions can be evaluated using truth tables.

<table>
<thead>
<tr>
<th>total &lt; MAX</th>
<th>found</th>
<th>!found</th>
<th>total &lt; MAX &amp;&amp; !found</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
We also have to be careful when comparing two floating point values (float or double) for equality.

You should rarely use the equality operator (==) when comparing two floats.

In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal.

Therefore, to determine the equality of two floats, you may want to use the following technique:

```java
if (Math.abs(f1 - f2) < 0.00001)
    System.out.println("Essentially equal.");
```
More Operators

- To round out our knowledge of Java operators, let's examine a few more.

- In particular, we will examine:
  - the increment and decrement operators
  - the assignment operators
  - the conditional operator
The increment and decrement operators are arithmetic and operate on one operand.

- The increment operator (`++`) adds one to its operand.
- The decrement operator (`--`) subtracts one from its operand.

The statement

```
count++;  
```

is functionally equivalent to

```
count = count + 1;  
```
Assignment Operators

- Often we perform an operation on a variable, and then store the result back into that variable.
- Java provides *assignment operators* to simplify that process.
- For example, the statement
  
  ```java
  num += count;
  ```

  is equivalent to
  
  ```java
  num = num + count;
  ```
There are many assignment operators, including the following:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent To</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>x += y</td>
<td>x = x + y</td>
</tr>
<tr>
<td>-=</td>
<td>x -= y</td>
<td>x = x - y</td>
</tr>
<tr>
<td>*=</td>
<td>x *= y</td>
<td>x = x * y</td>
</tr>
<tr>
<td>/=</td>
<td>x /= y</td>
<td>x = x / y</td>
</tr>
<tr>
<td>%=</td>
<td>x %= y</td>
<td>x = x % y</td>
</tr>
</tbody>
</table>
Repetition Statements

- *Repetition statements* allow us to execute a statement multiple times
- Often they are referred to as *loops*
- Like conditional statements, they are controlled by boolean expressions
- Java has three kinds of repetition statements:
  - the *while loop*
  - the *do loop*
  - the *for loop*
- The programmer should choose the right kind of loop for the situation
The while Statement

The while statement has the following syntax:

```plaintext
while (condition) statement;
```

- **while** is a reserved word.
- If the **condition** is true, the **statement** is executed.
- Then the **condition** is evaluated again.

The **statement** is executed repeatedly until the **condition** becomes false.
The while Statement

- Note that if the condition of a `while` statement is false initially, the statement is never executed.
- Therefore, the body of a `while` loop will execute zero or more times.
Infinite Loops

- The body of a `while` loop eventually must make the condition false.
- If not, it is an *infinite loop*, which will execute until the user interrupts the program.
- This is a common logical error.
- You should always double check to ensure that your loops will terminate normally.
The for statement has the following syntax:

```java
for (initialization; condition; increment) statement;
```

- **Reserved word**
- The **initialization** is executed once before the loop begins
- The **statement** is executed until the **condition** becomes false
- The **increment** portion is executed at the end of each iteration
- The **condition-statement-increment** cycle is executed repeatedly
A `for` loop is functionally equivalent to the following `while` loop structure:

```plaintext
initialization;
while (condition)
{
    statement;
    increment;
}
```

This is a key to your homework assignment.
Logic of a for loop

- **Initialization**: The loop starts with the initialization step.
- **Condition Evaluation**: The condition is evaluated.
  - If true, the statement is executed.
  - If false, the loop ends.
- **Increment**: After the statement is executed, the loop variable is incremented.

The flowchart illustrates the sequence of these steps in a for loop.
The for Statement

- Like a `while` loop, the condition of a `for` statement is tested prior to executing the loop body.
- Therefore, the body of a `for` loop will execute zero or more times.
- It is well suited for executing a loop a specific number of times that can be determined in advance.
The for Statement

Each expression in the header of a for loop is optional

- If the *initialization* is left out, no initialization is performed
- If the *condition* is left out, it is always considered to be true, and therefore creates an infinite loop
- If the *increment* is left out, no increment operation is performed

Both semi-colons are always required in the *for* loop header
Let’s look at your assignment

```java
for(int i=1; i<=100; i++){
    for(int j=1; j<=100; j++){
        ...
    } // for
} // for

Replace with two while loops.

Hmmmm. Let’s try a simpler example, Counter3.java from chapter 3.
```
public class Counter3 {
    public static void main (String[] args) {
        final int LIMIT = 5;
        for (int count=1; count <= LIMIT; count++)
            System.out.println (count);
        System.out.println ("Done");
    } //main
} //Counter3

Let’s open it up in JCreator and see if we can change this for-loop to a while-loop.
public class Summer {
    public static void main (String[] args) {
        final int LIMIT = 10;
        int sum = 0;
        int i = 1;
        while ( i <= LIMIT ){
            sum = sum + i;
            i++;
        } //while
        System.out.println (sum);
    } //main
} //Summer

Let’s open it up in JCreator and see if we can change this while-loop to a for-loop.