What we will cover:
Boolean data type
Boolean operations
Comparison operators
Conditional structures (if, if-else, if-else if, nested if)
Common Errors

Boolean data type
A boolean data type can represent a true or false (true and false are reserved words). Boolean expressions are commonly used for comparisons and are usually used with control statements (if-else, for, while).

A boolean variable declaration:
```java
boolean pass = true; // could also initialize to false
```

Boolean Operations
Boolean or logical operators operate on boolean values. Java has four Boolean operators: logical not (!), logical and (&&), logical or (||), and logical exclusive or (^). The results of logical operators are given via truth tables.

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p &amp;&amp; q</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

| p  | q  | p || q |
|----|----|--------|
| false | false | false |
| false | true  | true  |
| true  | false | true  |
| true  | true  | true  |

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p ^ q</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

The general format for using a logical operator is
 `<Boolean value> <logical operator> <Boolean value>`

For example
```java
boolean p = true, q = false, r;
r = (p && q); // false
r = (p || q); // true
r = !p; // false
r = (p ^ q); // true
```

Comparison Operators
Boolean or logical operators

Java has six comparison (relational) operators. The result of a comparison operator is a boolean (true or false). The comparison operators in Java are: less than (<), greater than (>), equal to (==), less than or equal to (<=), greater than or equal to (>=), and not equal to (!=).

The general format for using a comparison operator is
 `<expression> <comparison operator> <expression>`
For example

```java
boolean r;
int number = 7;
r = ((number >= 6) && (number <= 10)); // true
r = (number == 8); // false
```

Remember that in Java, = is the assignment operator and == is the equal comparison operator.

Consider writing a program to determine if a student passes or fails a quiz. Quiz scores fall in the range of 0–10 and 7 or higher is considered passing.

We can replace our usual calculation step in this case with a comparison. We can use our comparison operators to determine if the score is great than or equal to seven and display the value of the `boolean` variable.

```java
import javax.swing.JOptionPane;
public class QuizPassFail {
    public static void main(String[] args) {
        String temp;
        int quizscore = 0;
        boolean pass = false;

        // read in quiz score
        temp = JOptionPane.showInputDialog(null, "Enter quiz score (0 - 10): ");
        quizscore = Integer.parseInt(temp);

        // determine if quiz score is passing
        pass = (quizscore >= 7);

        // display result
        temp = "Is quiz score of " + quizscore + " a passing grade? " + pass;
        JOptionPane.showMessageDialog(null, temp, "Grade", JOptionPane.PLAIN_MESSAGE);

        // terminate the program
        System.exit(0);
    }
}
```

**Operator Precedence**

When evaluation expressions, operator precedence is as follows:

<table>
<thead>
<tr>
<th>highest</th>
<th>highest</th>
</tr>
</thead>
<tbody>
<tr>
<td>!=,</td>
<td>()</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=</td>
<td>!, -</td>
</tr>
<tr>
<td>-, *</td>
<td>,</td>
</tr>
<tr>
<td>+, %=</td>
<td>=</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td></td>
</tr>
<tr>
<td>lowest</td>
<td></td>
</tr>
</tbody>
</table>

Operations are evaluated from left to right. The operation is stopped (short-circuited) when the computer determines the result of the expression (even if the entire operation has not been completed).

For example

```java
boolean r;
int a = 7, b = 3;
r = (a > b) || (a == b); // r is true, the (a == b) part is never evaluated
```
Control Statements
Control structures allow us to control the flow of the program. Selection statements (if-else, switch) allow the program to execute different instructions based on the result of a test of some condition. Iteration (for, while) allows the program to repeat instructions some number of times.

Selection Statements
Selection allows a program to execute a block of code depending upon the result of a test. If the test is true, the block of code is executed and if the test is false, the block of code is skipped.

if and if-else statements
The if statement has the general format

\[
\text{if (expression) statement}
\]

The expression must evaluate to true or false and is usually a combination of relational and/or logical and/or mathematical operators.

The statement can be a compound statement; a block of code enclosed by braces {}.

\[
\text{if (expression) }
\begin{align*}
\text{statement1;} \\
\text{statement2;} \\
\text{etc}
\end{align*}
\]

Consider our previous program that determined if a quiz score was passing or failing. We could determine pass or fail and display part of the program to selectively generate a text message to be displayed using if statements rather than getting a true/false result from the Boolean operations.

```
// determine if quiz score is passing
if (quizscore >=7) {
    temp = "Student passed";
}
if (quizscore <7) {
    temp = "Student failed";
}
```

For conditional operations with only two outcomes, we can use if-else selection statement. The if-else statement has the general format

\[
\text{if (expression)}
\begin{align*}
\text{statements to execute if true;} \\
\text{else } \text{ statements to execute if false;}
\end{align*}
\]

The else block is optional; you would have a simple if statement if it was left off.

Rewriting the previous quiz pass fail code segment using an if-else:
```
// determine if quiz score is passing
if (quizscore >=7) {
    temp = "Student passed";
} else {
    temp = "Student failed";
}
```
Nested if Statements

If statement blocks can contain any legal, valid Java code, including more if statements. This can be used in cases where there are multiple alternatives based on a conditional statement.

```java
if (i > k) {
    if (j > k) {
        System.out.println("i and j are greater than k");
    }
} else {
    System.out.println("i is less than or equal to k");
}
```

Nested if statements like this can be confusing however, as shown here:

```java
if (score >= 90.0)
    grade = 'A';
else
    if (score >= 80.0)
        grade = 'B';
    else
        if (score >= 70.0)
            grade = 'C';
        else
            if (score >= 60.0)
                grade = 'D';
            else
                grade = 'F';
```

This type of statement can (and should) be rewritten as follows:

```java
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```

Common Errors

Selection statements are slightly different from most of the things we have been doing before now. Beware these common errors:

1. Forgetting Braces – if and else blocks can hold multiple statements to be executed, but they must be contained in curly braces. It is usually a good idea to get in the habit of using braces for most conditional statements.
2. Ending the if line with a semicolon – A semicolon indicates the end of a statement. Putting a semicolon at the end of the if line is equivalent to having an empty block. Since any code can be surrounded with curly braces, the compiler will not recognize that you only intended the braced code to be executed for the condition specified. This usually occurs with the next-line brace style.

```java
boolean wasUpdated = true;
if (wasUpdated);
```

3. Redundant testing of booleans – If your conditional statement is a boolean variable, like above, you do not need to test if it is == to true or false. You can use just the variable, or the ! (not) operator to test for true or false.
```
4. Dangling else – Else clauses should clearly indicate the if clause to which they belong. Poorly indented or formatted code can be hard to read, but also obfuscate, or hide, the true meaning of a conditional clause. In Java, the else
clause is always matched to the most recent unmatched clause in the same block. This may not be the behavior you expect or want, and is another reason to use braces to clearly denote your conditional blocks.

```java
int i = -2, j = 1, k = 0;

if (i > j)
    if (i > k)
        System.out.println("i is greater than j and k");
else
    System.out.println("i is less than or equal to j");
```

This code segment prints nothing because the last else is associated with the inner-if block. Note that proper bracing clears this up quickly.

```java
int i = -2, j = 1, k = 0;

if (i > j) {
    if (i > k) {
        System.out.println("i is greater than j and k");
    }
} else {
    System.out.println("i is less than or equal to j");
}
```

Class Exercise 10 – Circle.java and SafeSquareRoot.java