Programming 1

Week 04 – Decision Structures

Control structures

- So far, we’ve viewed programs as sequences of instructions that are followed one after the other.
- While this is a fundamental programming concept, it is not sufficient in itself to solve every problem.
- We need to be able to alter the sequential flow of a program to suit a particular situation.
- *Control structures* allow us to alter this sequential program flow.

Selection Statements

Selection Statements allow a computer to make choices based on a **condition**

- Three types of selection statements.
- `if` statement:
  - Single-selection statement—selects or ignores a single action (or group of actions).
  - Performs an action, if a condition is *true*; skips it, if *false*.
- `if…else` statement:
  - Double-selection statement—selects between two different actions (or groups of actions).
  - Performs an action if a condition is *true* and performs a different action if the condition is *false*.
- `switch` statement
  - Multiple-selection statement—selects among *many different actions* (or *groups of actions*).
  - Performs one of several actions, based on the value of an expression.

Single-selection statement: The **`if`** Statement

- The `if` statement decides whether a section of code executes or not.
- The `if` statement uses a `boolean` to decide whether the next statement or block of statements executes.
  - The simplest statement to make a decision
  - A Boolean expression appears within parentheses
  - No space between the keyword `if` and the opening parenthesis

if (boolean expression is true)

execute next statement.

Relational Operators

- In most cases, the `boolean` expression, used by the `if` statement, uses **relational operators**.

Relational Operator	Meaning
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to
==	is equal to
!=	is not equal to

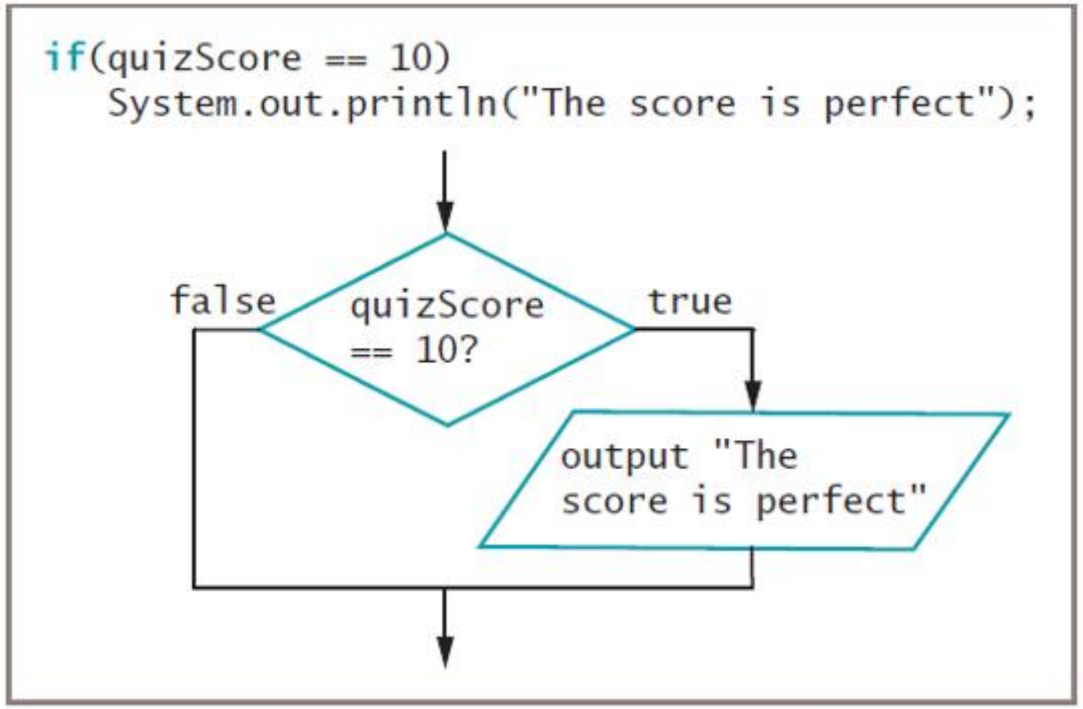
Using a single equal sign (=) rather than a double equal sign (==) is illegal

Boolean Expressions

- A **boolean expression** is any variable or calculation that results in a **true** or **false** condition.

Expression	**Meaning**
`x > y`	Is x greater than y?
`x < y`	Is x less than y?
`x >= y`	Is x greater than or equal to y?
`x <= y`	Is x less than or equal to y.
`x == y`	Is x equal to y?
`x != y`	Is x not equal to y?

The `if` Statement



```
if(quizScore == 10)
   System.out.println("The score is perfect");
```

The `if` Statement

- There should be no semicolon at the end of the first line of the `if` statement
  - `if(someVariable == 10)`
  - The statement does not end there
- When a semicolon follows `if` directly:
  - An **empty statement** contains only a semicolon
  - Execution continues with the next independent statement

`if` Single-Selection Statement

- Pseudocode

```
If student’s grade is greater than or equal to 60
   Print “Passed”
```

- If the condition is false, the Print statement is ignored, and the next pseudocode statement in order is performed.
- Indentation
  - Optional, but recommended
  - Emphasizes the inherent structure of structured programs
- The preceding pseudocode *If* in Java:

```java
if (studentGrade >= 60) {
   System.out.println("Passed");
}
```

- Corresponds closely to the pseudocode.

`if` Statements and Boolean Expressions

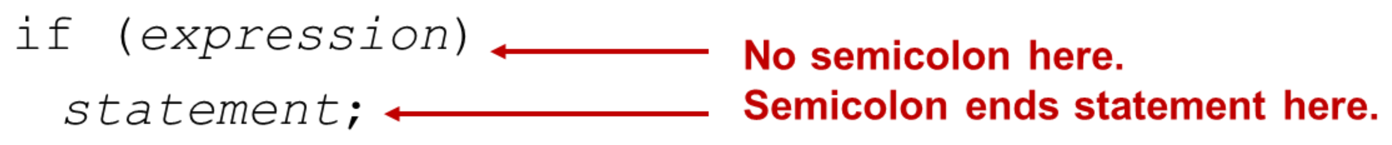
```
if (x > y)
   System.out.println("X is greater than Y");

if (x == y)
   System.out.println("X is equal to Y");

if (x != y)
{
   System.out.println("X is not equal to Y");
   x = y;
   System.out.println("However, now it is.");
}
```

Programming Style and `if` Statements

- Rules of thumb:
  - The conditionally executed statement should be on the line after the `if` condition.
  - The conditionally executed statement should be indented one level from the `if` condition.
  - If an `if` statement does not have the block curly braces, it is ended by the first semicolon encountered after the `if` condition.



```
if (expression)
  statement;
```

Block `if` Statements

- Conditionally executed statements can be grouped into a block by using curly braces { } to enclose them.
- If curly braces are used to group conditionally executed statements, the `if` statement is ended by the closing curly brace.

```
if (expression)
{
  statement1;
  statement2;

 }  ←———————  Curly brace ends the statement.
```

Block `if` Statements

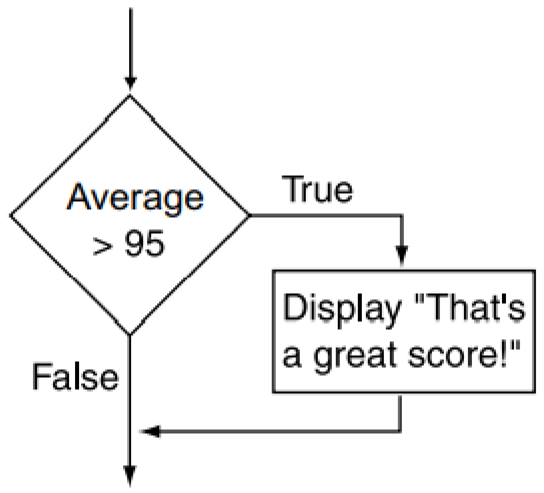
- Remember that when the curly braces are not used, then only the next statement after the `if` condition will be executed conditionally.

```
if (expression)
  statement1;
  statement2;
  statement3;
```

← **Only this statement is conditionally executed.**

Demo

- Write a program where the user enters three test scores and the program calculates their average. If the average is greater than 95, the program congratulates the user on obtaining a high score



Demo

1. Write an if statement that multiplies payRate by 1.5 if hours is greater than 40.
2. Write an if statement that assigns 0.2 to commission if sales is greater than or equal to 10000.
3. Write an if statement that assigns 0 to the variable b and assigns 1 to the variable c if the variable a is less than 10.

Flags

- You can store a Boolean expression’s value in a Boolean variable before using it in an `if` statement
- A flag is a `boolean` variable that monitors some condition in a program.
- When a condition is true, the flag is set to `true`.
- The flag can be tested to see if the condition has changed.

```
if (average > 95)
  highScore = true;
```

- Later, this condition can be tested:

```
if (highScore)
  System.out.println("That's a high score!");
```

Comparing Characters

- Characters can be tested with relational operators.
- Characters are stored in memory using the Unicode character format.
- Unicode is stored as a sixteen (16) bit number.
- Characters are **ordinal**, meaning they have an order in the Unicode character set.
- Since characters are ordinal, they can be compared to each other.

```
char c = 'A';
if(c < 'Z')
  System.out.println("A is less than Z");
```

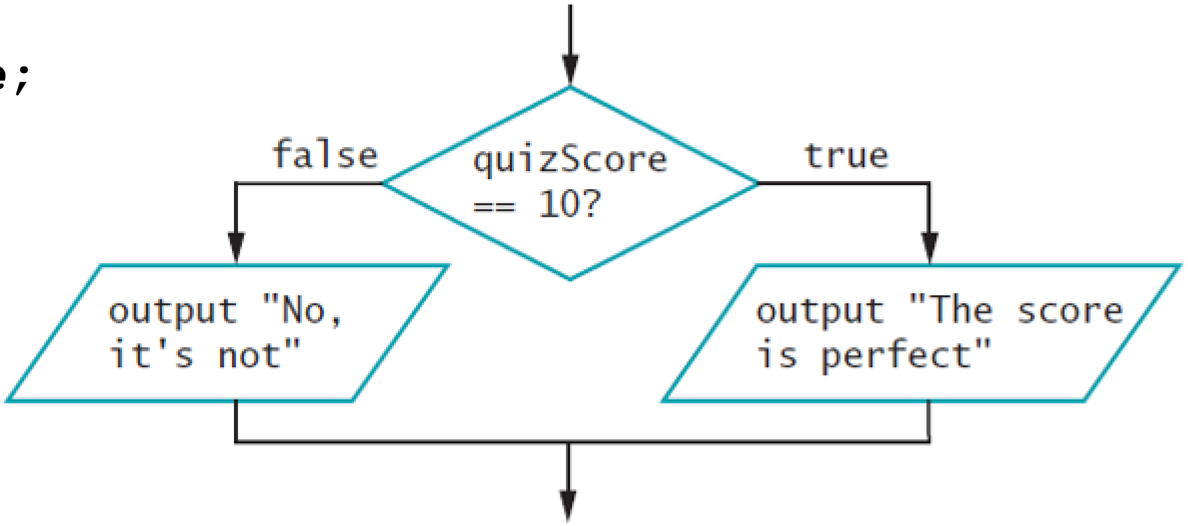
The if…else Statement

- **`if`…`else` statement**
  - A statement that executes when `if` is `true` or `false` and ends with a semicolon
  - Vertically align the keyword `if` with the keyword `else`
  - It’s illegal to code `else` without `if`
  - Depending on the evaluation of the Boolean expression following `if`, only one resulting action takes place

The if…else Statement

```
if (expression)
  statementOrBlockIfTrue;
else
  statementOrBlockIfFalse;
```

```
if(quizScore == 10)
   System.out.println("The score is perfect");
else
   System.out.println("No, it's not");
```



The if…else Statement

- To execute more than one statement, use a pair of curly braces
    - Place dependent statements within a block
    - It’s crucial to place the curly braces correctly
- Any variable declared within a block is local to that block

Demo

- Write a program to allow the user to input his/her age. Then the program will show if the person is eligible to vote. A person who is eligible to vote must be older than or equal to 18 years old.

Nested `if` Statements (1 of 2)

- If an `if` statement appears inside another `if` statement (single or block) it is called a **nested `if`** statement.
- The nested `if` is executed only if the outer `if` statement results in a true condition.

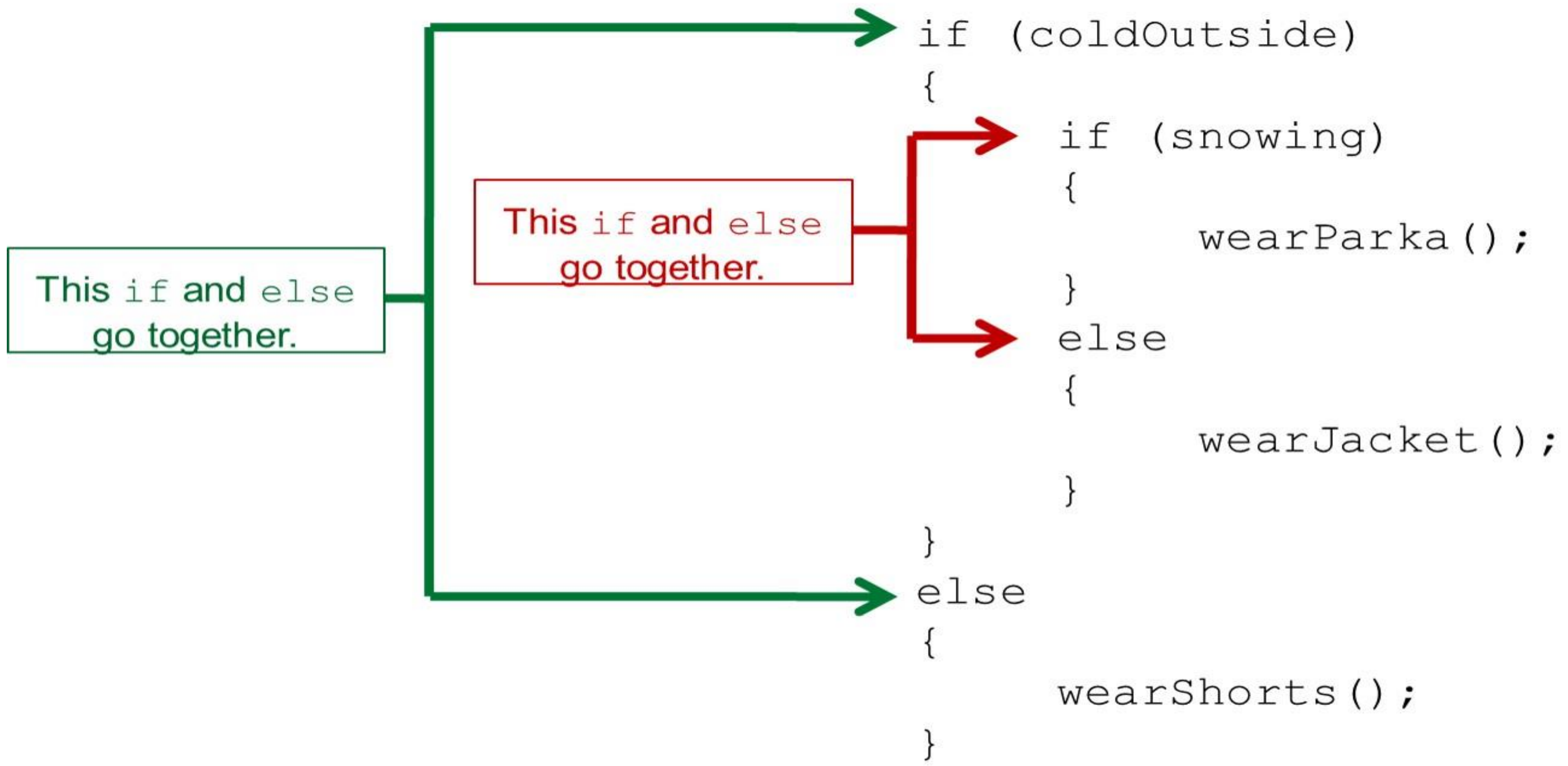
Nested `if` Statements (2 of 2)

```
if (coldOutside)
{
      if (snowing)
      {
            wearParka();
      }
      else
      {
            wearJacket();
      }
}
else
{
      wearShorts();
}
```

`if-else` Matching

- Curly brace use is not required if there is only one statement to be conditionally executed.
- However, sometimes curly braces can help make the program more readable.
- Additionally, proper indentation makes it much easier to match up else statements with their corresponding `if` statement.

Alignment and Nested **`if`** Statements



```
if (coldOutside)
{
     if (snowing)
     {
          wearParka();
     }
     else
     {
          wearJacket();
     }
}
else
{
     wearShorts();
}
```

Demo

- Write code that tests the variable x to determine whether it is greater than 0. If x is greater than 0, the code should test the variable y to determine whether it is less than 20. If y is less than 20, the code should assign 1 to the variable z. If y is not less than 20, the code should assign 0 to the variable z.

`if-else-if` Statements (1 of 3)

```
if (expression_1)
{
   statement;
   statement;
   etc.
}

else if (expression_2)
{
   statement;
   statement;
   etc.
}
```

If *expression_1* **is true these statements are executed, and the rest of the structure is ignored.**

Otherwise, if *expression_2* **is true these statements are executed, and the rest of the structure is ignored.**

`if-else-if` Statements (2 of 3)

Insert as many *`else if`* clauses as necessary

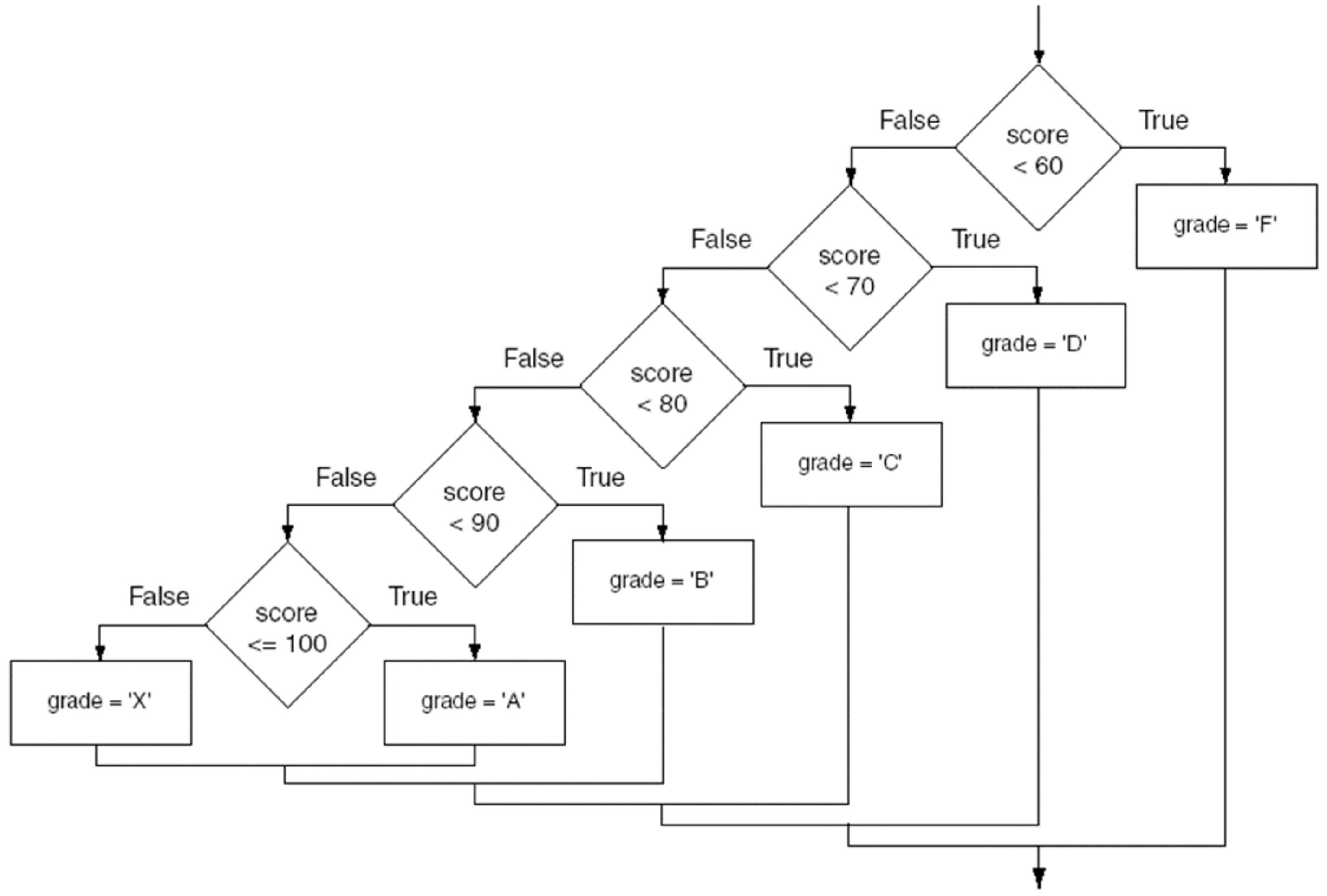
```
else
{
    statement;
    statement;
    etc.
}
```

$\left.\vphantom{\begin{matrix}a\\b\\c\end{matrix}}\right\}$ **These statements are executed if none of the expressions above are true.**

if-else-if Statements (3 of 3)

- Nested if statements can become very complex.
- The if-else-if statement makes certain types of nested decision logic simpler to write.
- Care must be used since else statements match up with the immediately preceding unmatched if statement.

`if-else-if` Flowchart



Logical Operators (1 of 2)

- Java provides two binary **logical operators** (&& and ||)

  that are used to combine `boolean` expressions.

- Java also provides one **unary** (!) logical operator to reverse the truth of a `boolean` expression.
- Note that we can implement any logic in a program using only a simple Boolean expression.
- However, we can make shorter and more expressive code by combining simple Boolean expressions using logical operators(and, or, not) to create compound Boolean expressions.

Logical Operators (2 of 2)

Operator	Meaning	Effect
&&	**AND**	Connects two Boolean expressions into one. Both expressions must be true for the overall expression to be true.
||	**OR**	Connects two Boolean expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which one.
!	**NOT**	The ! operator reverses the truth of a Boolean expression. If it is applied to an expression that is true, the operator returns false. If it is applied to an expression that is false, the operator returns true.

The `&&` Operator

- The logical AND operator (`&&`) takes two operands that must both be `boolean` expressions.
- The resulting combined expression is true if (and **only** if) both operands are true.

Expression 1	**Expression 2**	**Expression1 && Expression2**
true	false	false
false	true	false
false	false	false
true	true	true

The || Operator

- The logical OR operator (`||`) takes two operands that must both be `boolean` expressions.
- The resulting combined expression is false if (and **only** if) both operands are false.

Expression 1	**Expression 2**	**Expressi||ı1 double vertical bars Expression2**
true	false	true
false	true	true
false	false	false
true	true	true

The ! Operator

- The ! operator performs a logical NOT operation.
- If an *expression* is true, !*expression* will be false.

```
if (!(temperature > 100))
  System.out.println("Below the maximum temperature.");
```

- If **`temperature > 100`** evaluates to false, then the output statement will be run.

Expression 1	!Expression1
true	false
false	true

Order of Precedence (1 of 2)

- The `!` operator has a higher order of precedence than the `&&` and `||` operators.
- The `&&` and `||` operators have a lower precedence than relational operators like `<` and `>`.
- Parenthesis can be used to force the precedence to be changed.

Consider the following two approaches:

Approache#1:

```
System.out.print ("Enter the numeric grade please: ");
Int number = input.nextInt();
if (number > 100)
      System.out.print("Error: grade must be between 100 and 0")
else
If (number <0)
      System.out.print("Error: grade must be between 100 and 0")
```

Approache#2:

```
System.out.print ("Enter the numeric grade please: ");
Int number = input.nextInt();
if (number > 100 or number <0)
      System.out.print("Error: grade must be between 100 and 0")
```

Variable Scope

- In Java, a local variable does not have to be declared at the beginning of the method.
- The scope of a local variable begins at the point it is declared and terminates at the end of the method.
- When a program enters a section of code where a variable has scope, that variable has **come into scope**, which means the variable is visible to the program.

The Conditional Operator (1 of 4)

- The **conditional operator** is a ternary (three operand) operator.
- You can use the conditional operator to write a simple statement that works like an `if-else` statement.

The Conditional Operator (2 of 4)

- The format of the operators is:

```
BooleanExpression ? Value1 : Value2
```

- This forms a conditional expression.
  - If *`BooleanExpression`* is true, the value of the conditional expression is *`Value1`*.
  - If *`BooleanExpression`* is false, the value of the conditional expression is *`Value2`*.

The Conditional Operator (3 of 4)

- Example:

```
z = x > y ? 10 : 5;
```

- This line is functionally equivalent to:

```
if(x > y)
  z = 10;
else
  z = 5;
```

The Conditional Operator (4 of 4)

- The conditional operator is used also to supply a value.

```
number = x > y ? 10 : 5;
```

- This is functionally equivalent to:

```
if(x > y)
  number = 10;
else
  number = 5;
```

Demo

letter score: Convert Score to Letter Score

Score	Rule	Letter Score
[90, 100]	if (90 <= score <= 100)	A
[80, 90)	if (80 <= score < 90)	B
[70, 80)	if (70 <= score < 80)	C
[60, 70)	if (60 <= score < 70)	D
[0, 60)	if (0 <= score < 60)	F

A student can get only one Letter Score based on their score. If they get 'A', they cannot get another letter score.

Demo

- A parking garage charges a $3.00 minimum fee to park for up to three hours. The garage charges an additional $0.50 per hour for each hour or part thereof more than three hours. The maximum charge for any given 24-hour period is $10.00. Assume that no car parks for longer than 24 hours at a time. If the parking time is greater than 24 hours, then print out an error message.
- Write a java program that determine the parking charge for each customer. You should enter the hours parked for each customer. The program should display the total charge for the customer

```
public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the number of hours please: ");
        double hours = input.nextDouble();
        double charge = 0;
        if (hours > 24) {
            System.out.println("NO PARKING");
        }
        else { if (hours == 24) {
            charge = 10;
        } else if (hours <= 3) {
            charge = 3;
        } else {
            charge = 3 + (hours - 3) * 0.5;
        }
            System.out.println("Customer Charge is: " + charge);
        }
    }
}
```