SD400: Problem Solving

Lesson 4: Flow of Control
Branching (if statements)

Objectives

- Use JavaScript branching statements
- Compare values of primitive types
- Compare objects such as strings
- Use the primitive type **`boolean`**

Flow of Control

- *Flow of control* is the order in which a program performs actions.
  - Up to this point, the order has been sequential.
- A *branching statement* chooses between two or more possible actions.
- A *loop statement* repeats an action until a stopping condition occurs.

The `if-else` Statement: Outline

- Basic `if-else` Statement
- Boolean Expressions
- Comparing Strings
- Nested `if-else` Statements
- Multibranch `if-else` Statements

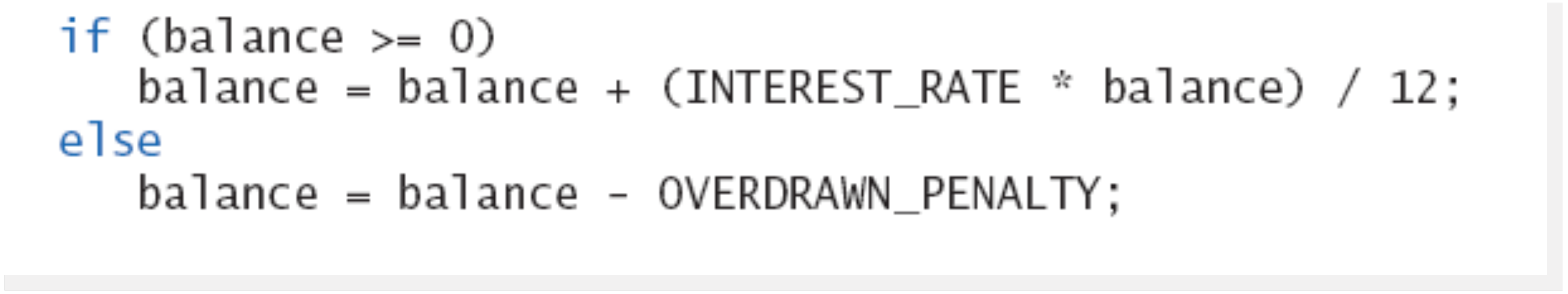
The `if-else` Statement

- A branching statement that chooses between two possible actions.
- Syntax

```
if (Boolean_Expression)
   Statement_1
else
   Statement_2
```

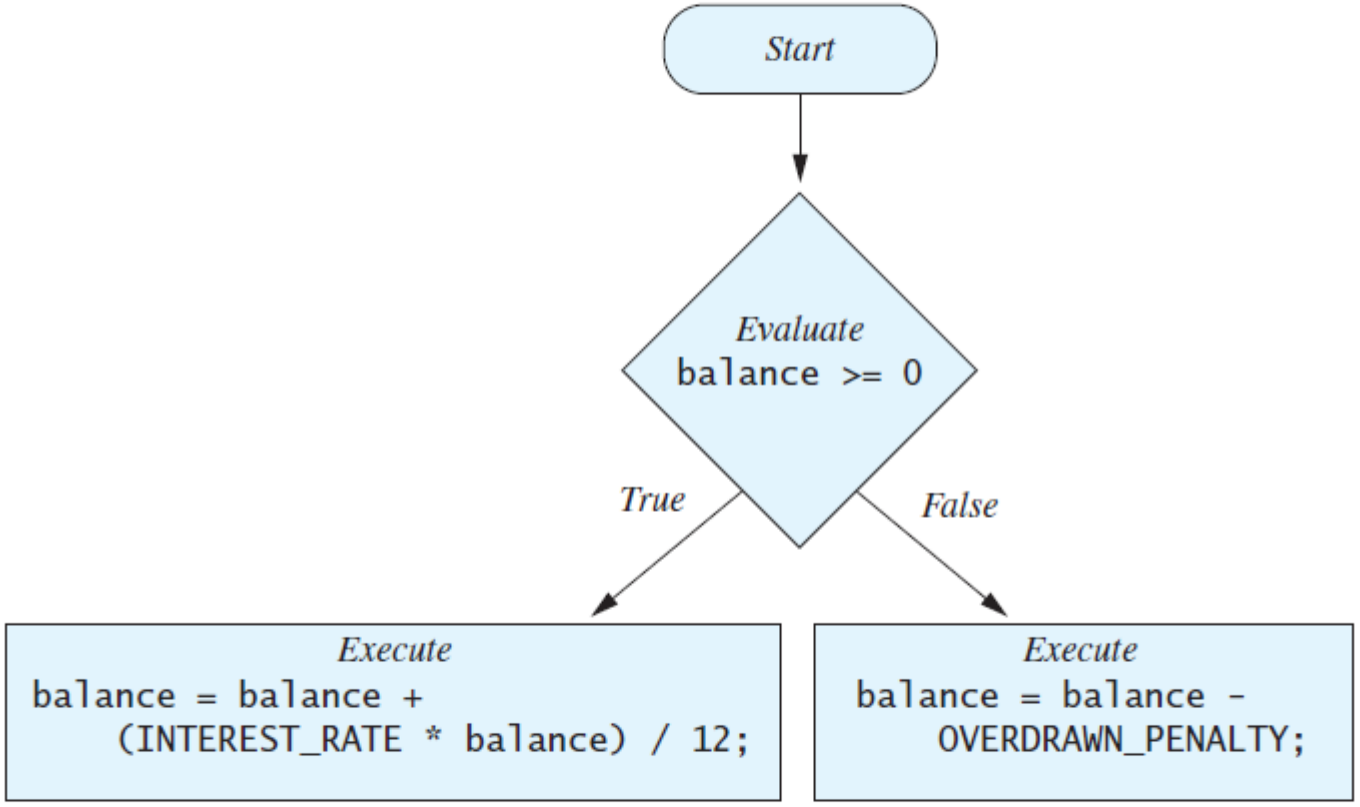
The if-else Statement

- Example



```
if (balance >= 0)
   balance = balance + (INTEREST_RATE * balance) / 12;
else
   balance = balance - OVERDRAWN_PENALTY;
```

The `if-else` Statement



Start
Evaluate
balance >= 0
True
False
Execute
balance = balance + (INTEREST_RATE * balance) / 12;
Execute
balance = balance - OVERDRAWN_PENALTY;

The `if-else` Statement

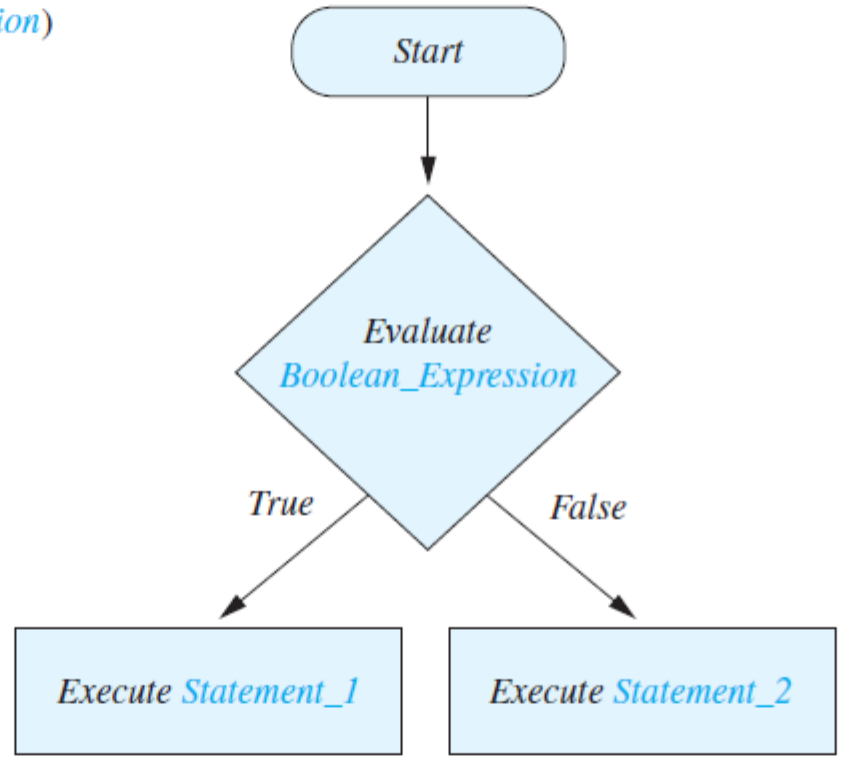
Sample screen output

```
Enter your checking account balance: $505.67
Original balance $505.67
After adjusting for one month of interest and penalties,
your new balance is $506.51278
```

```
Enter your checking account balance: $-15.53
Original balance $-15.53
After adjusting for one month of interest and penalties,
your new balance is $-23.53
```

Semantics of the `if-else` Statement

```
if (Boolean_Expression)
    Statement_1
else
    Statement_2
```



Compound Statements

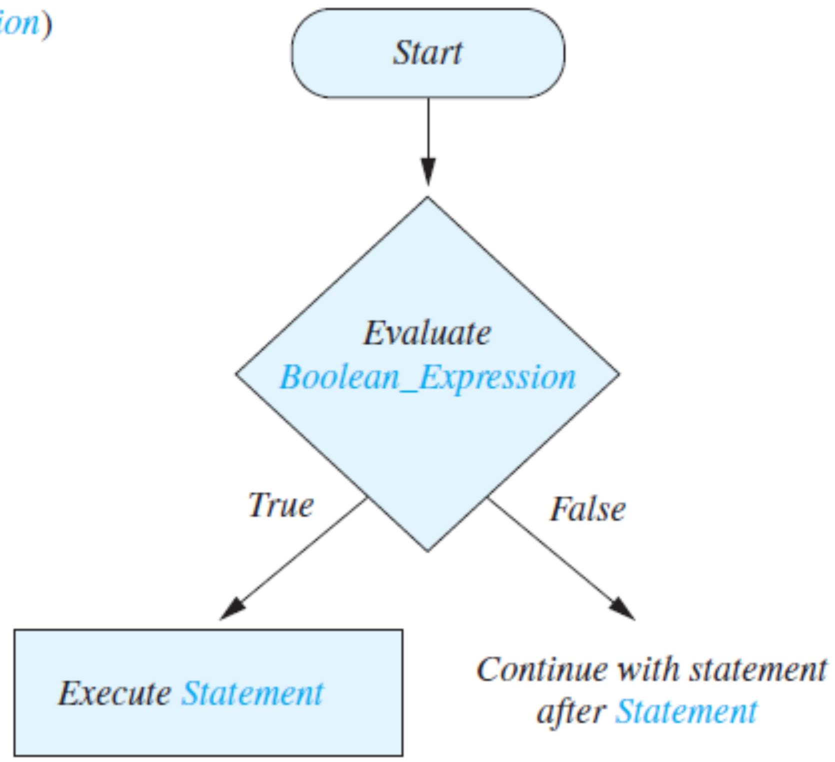
- To include multiple statements in a branch, enclose the statements in braces.

```
if (count < 3)
    {
        total = 0;
        count = 0;
    }
```

Omitting the `else` Part

- The Semantics of an `if` Statement without an `else`

```
if (Boolean_Expression)
    Statement
```

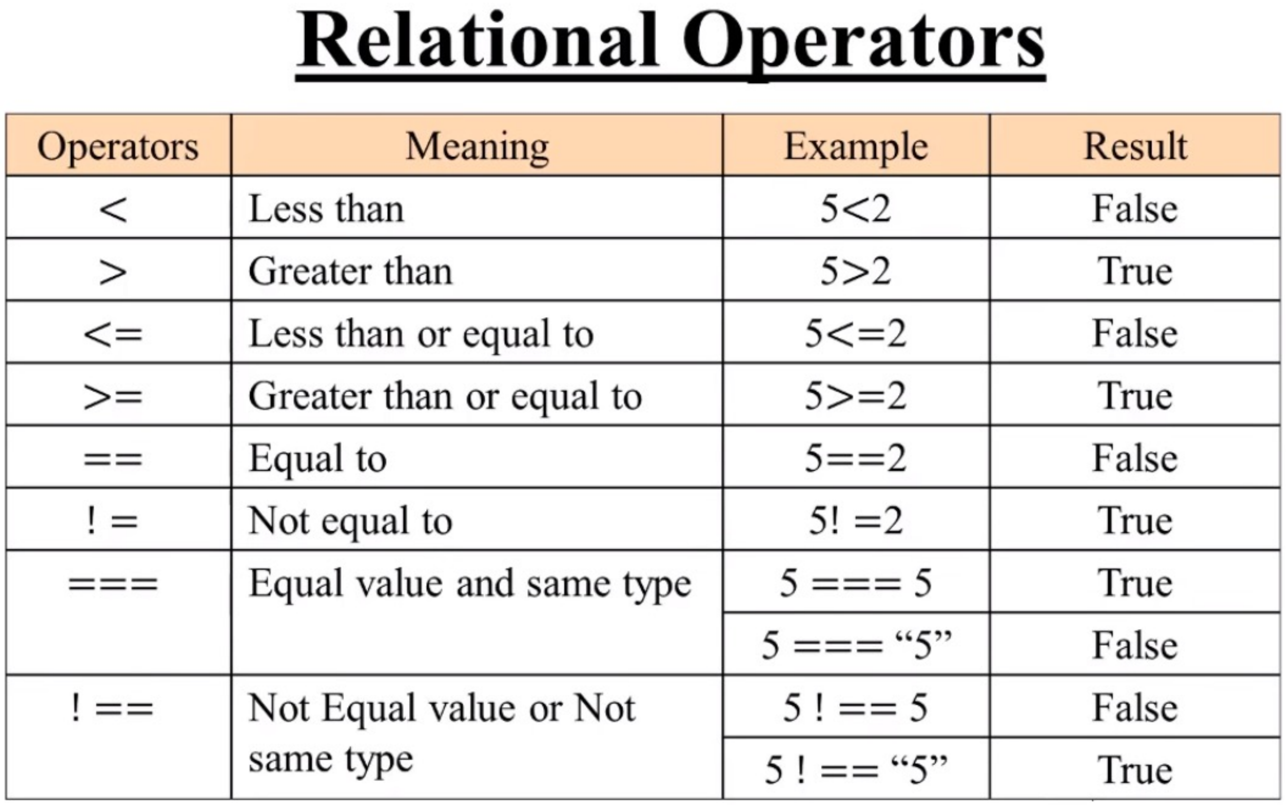

Start
Evaluate Boolean_Expression
True
False
Execute Statement
Continue with statement after Statement

Introduction to Boolean Expressions

- The value of a *boolean expression* is either **`true`** or **`false`**.
- Examples

```
time < limit
balance <= 0
```

JavaScript Comparison Operators



Relational Operators

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
! =	Not equal to	5! =2	True
===	Equal value and same type	5 === 5	True
		5 === "5"	False
! ==	Not Equal value or Not same type	5 ! == 5	False
		5 ! == "5"	True

Compound Boolean Expressions

- Boolean expressions can be combined using the "and" `(&&)` operator.
- Example

```
if ((score > 0) && (score <= 100))
...
```

- Not allowed

```
if (0 < score <= 100)
...
```

Compound Boolean Expressions

- Syntax
```
(Sub_Expression_1) && (Sub_Expression_2)
```
- Parentheses often are used to enhance readability.
- The larger expression is true only when both of the smaller expressions are true.

Compound Boolean Expressions

- Boolean expressions can be combined using the "or" **(||)** operator.
- Example

```
if ((quantity > 5) || (cost < 10))
...
```

- Syntax

```
(Sub_Expression_1) || (Sub_Expression_2)
```

Negating a Boolean Expression

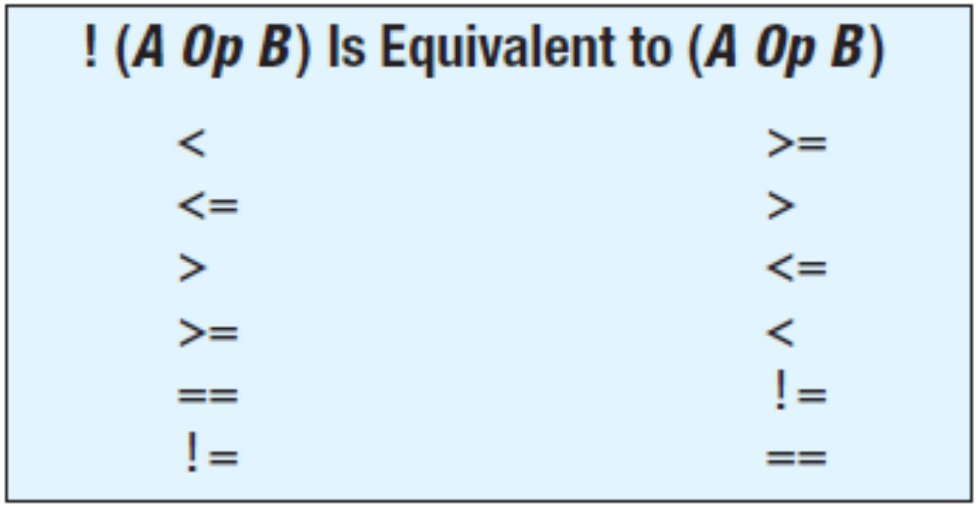
- A boolean expression can be negated using the "not" (**!**) operator.
- Syntax

  !(Boolean_Expression)
- Example

  `(a || b) && !(a && b)`

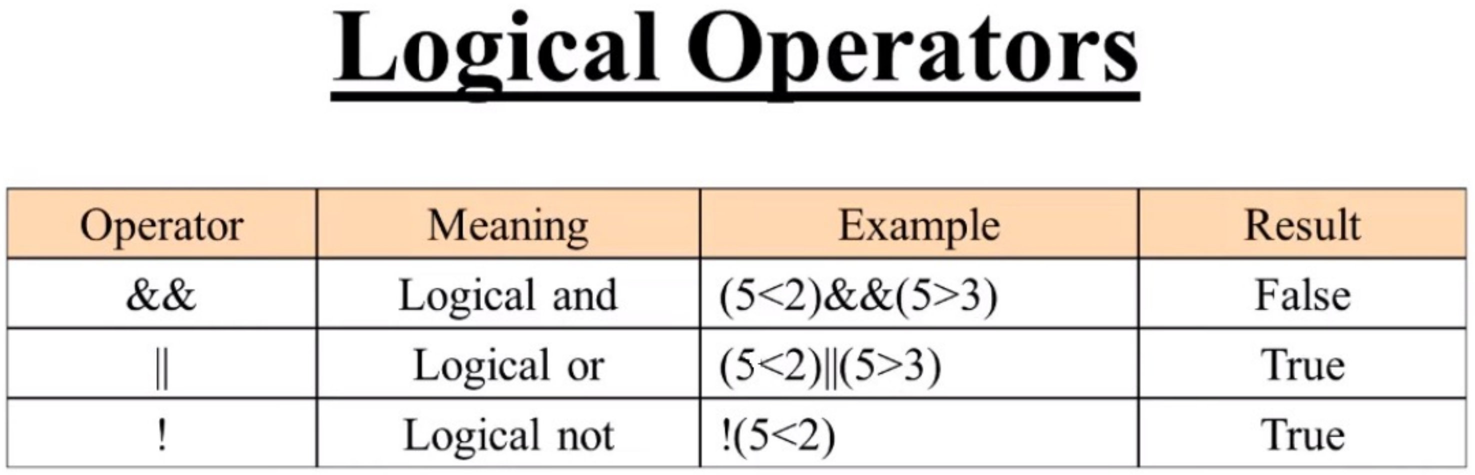
  which is the *exclusive or*

Negating a Boolean Expression



! (*A Op B*)	Is Equivalent to (*A Op B*)
<	>=
<=	>
>	<=
>=	<
==	!=
!=	==

JavaScript Logical Operators

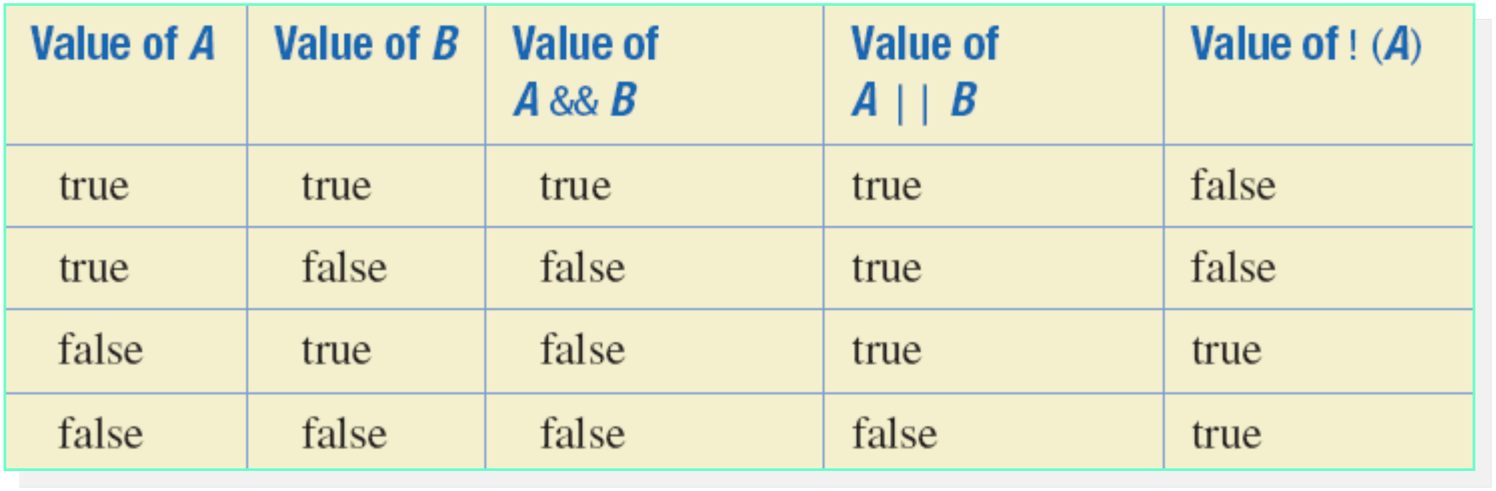


Logical Operators

Operator	Meaning	Example	Result
&&	Logical and	(5<2)&&(5>3)	False
||	Logical or	(5<2)||(5>3)	True
!	Logical not	!(5<2)	True

Boolean Operators

- The Effect of the Boolean Operators **&&** (and), **||** (or), and **!** (not) on Boolean values



Value of *A*	Value of *B*	Value of *A* && *B*	Value of *A* || *B*	Value of ! (*A*)
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Nested `if-else` Statements

- An `if-else` statement can contain any sort of statement within it.
- In particular, it can contain another `if-else` statement.
  - An `if-else` may be nested within the "if" part.
  - An `if-else` may be nested within the "else" part.
  - An `if-else` may be nested within both parts.

Nested Statements

- Syntax

```
if (Boolean_Expression_1)
      if (Boolean_Expression_2)
            Statement_1)
      else
            Statement_2)
else
    if (Boolean_Expression_3)
            Statement_3)
      else
            Statement_4);
```

Nested Statements

- Each `else` is paired with the nearest unmatched `if`.
- **If used properly**, indentation communicates which `if` goes with which `else`.
- Braces can be used like parentheses to group statements.

Nested Statements

- Subtly different forms

First Form

```
if (a > b)
{
    if (c > d)
        e = f
}
    else
        g = h;
```

Second Form

```
if (a > b)
    if (c > d)
        e = f
     else
        g = h;

// oops
```

Compound Statements

- When a list of statements is enclosed in braces (**{ }**), they form a single *compound statement.*
- Syntax

```
{
    Statement_1;
    Statement_2;
  …
}
```

Compound Statements

- A compound statement can be used wherever a statement can be used.
- Example

```
if (total > 10)
{
   sum = sum + total;
   total = 0;
}
```

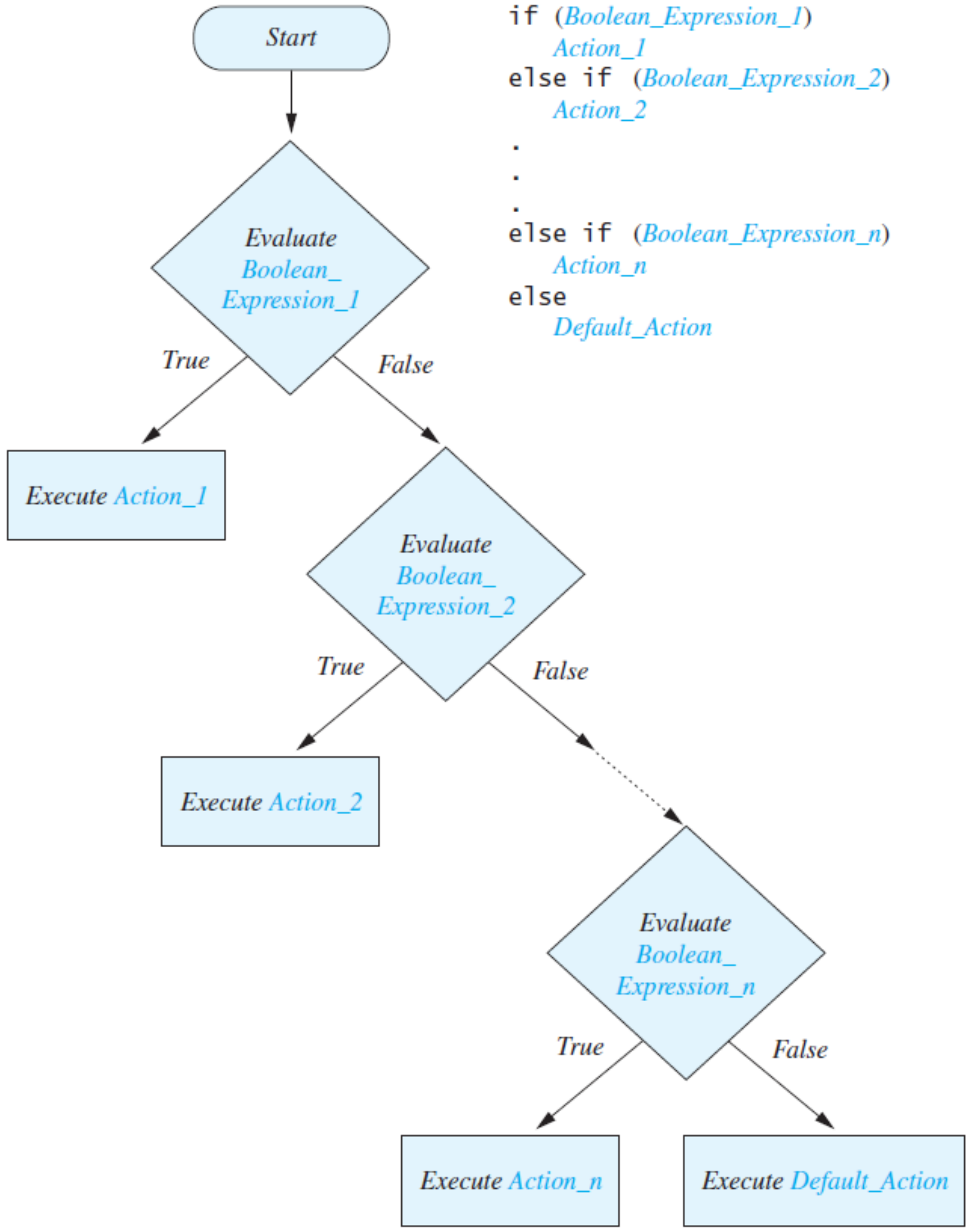
Multibranch `if-else` Statements

- Syntax

```
if (Boolean_Expression_1)
    Statement_1
else if (Boolean_Expression_2)
    Statement_2
else if (Boolean_Expression_3)
    Statement_3
else if …
else
    Default_Statement
```

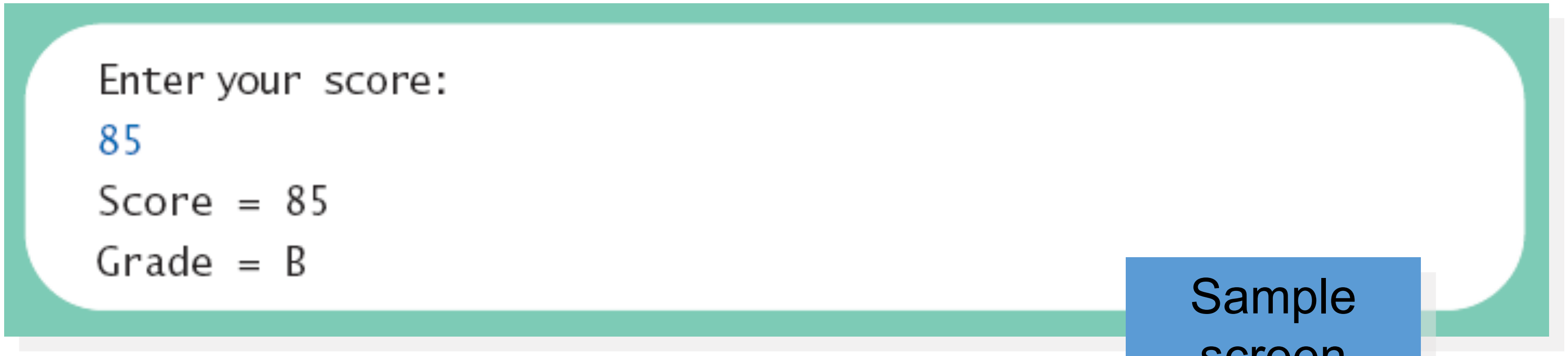
Multibranch `if-else` Statements

```
if (Boolean_Expression_1)
    Action_1
else if (Boolean_Expression_2)
    Action_2
.
.
.
else if (Boolean_Expression_n)
    Action_n
else
    Default_Action
```



Multibranch `if-else` Statements

- View sample program Listing 3.3
  `class Grader`



```
Enter your score:
85
Score = 85
Grade = B
```

Sample screen output

Multibranch `if-else` Statements

- Equivalent code

```
if (score >= 90)
      grade = 'A';
else if ((score >= 80) && (score < 90))
      grade = 'B';
else if ((score >= 70) && (score < 80))
      grade = 'C';
else if ((score >= 60) && (score < 70))
      grade = 'D';
else
      grade = 'F';
```

Case Study – Body Mass Index

- Body Mass Index (BMI) is used to estimate the risk of weight-related problems
- BMI = mass / height^2
  - Mass in kilograms, height in meters
- Health assessment if:

BMI < 18.5	Underweight
18.5 ≤ BMI < 25	Normal weight
25 ≤ BMI < 30	Overweight
30 ≤ BMI	Obese

Case Study – Body Mass Index

- Algorithm
  - Input height in feet & inches, weight in pounds
  - Convert to meters and kilograms
    - 1 lb = 2.2 kg
    - 1 inch = 0.254 meters
  - Compute BMI
  - Output health risk using if statements

View sample program Listing 3.4

```
class BMI
```

The Conditional Operator

```
if (n1 > n2)
    max = n1;
else
    max = n2;
```

can be written as

```
max = (n1 > n2) ? n1 : n2;
```

- The `?` and `:` together are call the *conditional operator* or *ternary operator.*

The Conditional Operator

- The conditional operator is useful with print and println statements.

```
System.out.print("You worked " + hours +
   ((hours > 1) ? "hours" : "hour"));
```

The Type `boolean`

- The type `boolean` is a primitive type with only two values: `true` and `false`.
- Boolean variables can make programs more readable.

```
if (systemsAreOK)
```

instead of

```
if((temperature <= 100) && (thrust >= 12000) &&
  (cabinPressure > 30) && …)
```

Boolean Expressions and Variables

- Variables, constants, and expressions of type **`boolean`** all evaluate to either **`true`** or **`false`**.
- A boolean variable can be given the value of a boolean expression by using an assignment operator.

```
boolean isPositive = (number > 0);
...
if (isPositive) ...
```

Naming Boolean Variables

- Choose names such as `isPositive` or `systemsAreOk`.
- Avoid names such as `numberSign` or `systemStatus`.

Precedence Rules

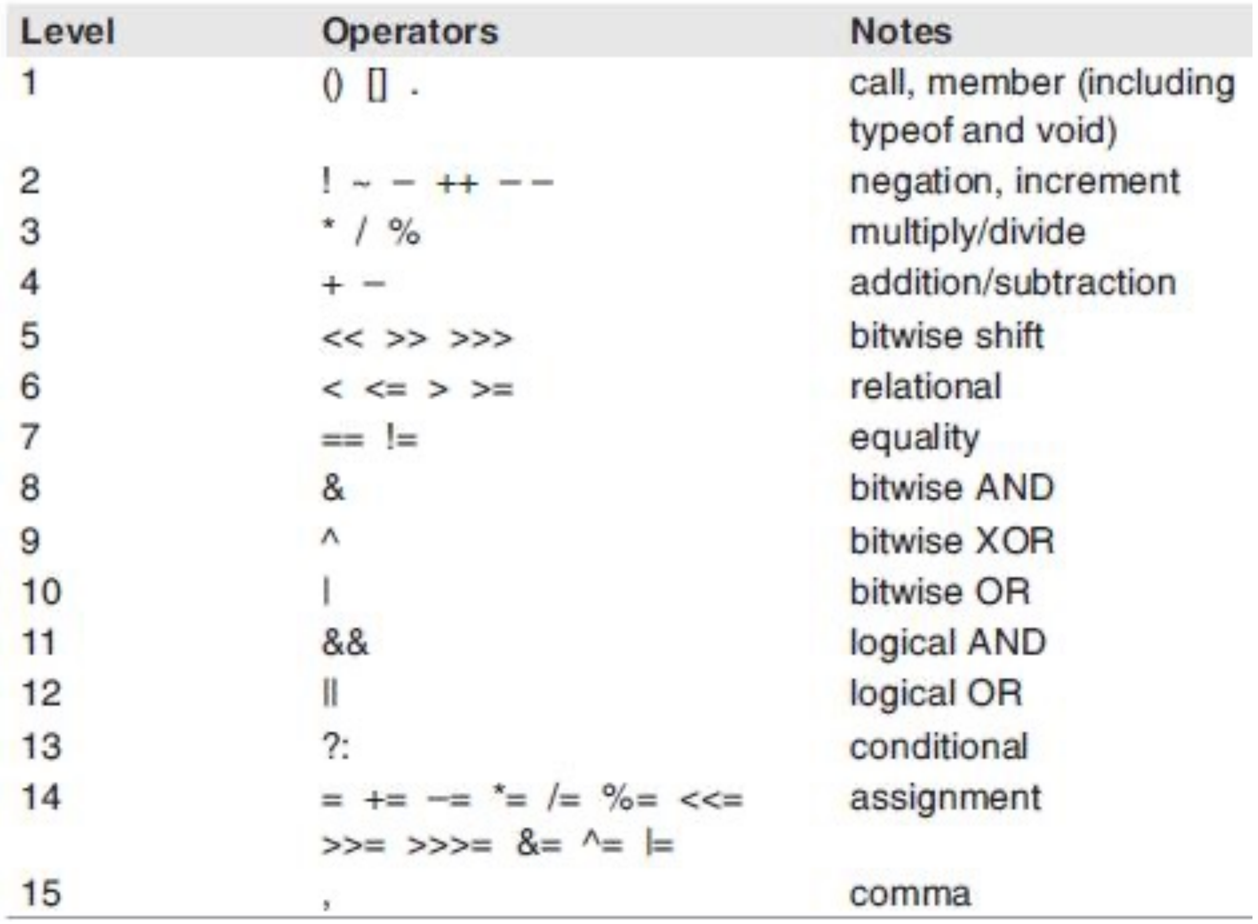
- Parentheses should be used to indicate the order of operations.
- When parentheses are omitted, the order of operation is determined by *precedence rules.*

Precedence Rules

- Operations with *higher precedence* are performed before operations with *lower precedence.*
- Operations with *equal precedence* are done left-to-right (except for unary operations which are done right-to-left).

Precedence Rules

Operator precedence



Level	Operators	Notes
1	() [] .	call, member (including typeof and void)
2	! ~ − ++ − −	negation, increment
3	* / %	multiply/divide
4	+ −	addition/subtraction
5	<< >> >>>	bitwise shift
6	< <= > >=	relational
7	== !=	equality
8	&	bitwise AND
9	^	bitwise XOR
10	|	bitwise OR
11	&&	logical AND
12	||	logical OR
13	?:	conditional
14	= += −= *= /= %= <<= >>= >>>= &= ^= |=	assignment
15	,	comma

Precedence Rules

- In what order are the operations performed?

```
score < min/2 - 10 || score > 90
score < (min/2) - 10 || score > 90
score < ((min/2) - 10) || score > 90
(score < ((min/2) - 10)) || score > 90
(score < ((min/2) - 10)) || (score > 90)
```

Short-circuit Evaluation

- Sometimes only part of a boolean expression needs to be evaluated to determine the value of the entire expression.
  - If the first operand associated with an `||` is `true`, the expression is `true`.
  - If the first operand associated with an `&&` is `false`, the expression is `false`.
- This is called *short-circuit* or *lazy* evaluation.

Short-circuit Evaluation

- Short-circuit evaluation is not only efficient, sometimes it is essential!
- A run-time error can result, for example, from an attempt to divide by zero.

```
if ((number != 0) && (sum/number > 5))
```

Input and Output of Boolean Values

- Example

```
boolean booleanVar = false;
System.out.println(booleanVar);
System.out.println("Enter a boolean value:");
Scanner keyboard = new Scanner(System.in);
booleanVar = keyboard.nextBoolean();
System.out.println("You entered " + booleanVar);
```

Input Validation

- You should check your input to ensure that it is within a valid or reasonable range. For example, consider a program that converts feet to inches. You might write the following:

```
int feet = keyboard.nextInt();
int inches = feet * 12;
```

- What if:
  - The user types a negative number for feet?
  - The user enters an unreasonable value like 100? Or a number larger than can be stored in an int? (2,147,483,647)

Input Validation

- Address these problems by ensuring that the entered values are reasonable:

```
int feet = keyboard.nextInt();
if ((feet >= 0) && (feet < 10))
{
  int inches = feet * 12;
  ...
}
```

Summary

- You have learned about JavaScripting branching statements.
- You have learned about the type **`boolean`**.