Chapter 3
Selection - Conditional Statement

COSC1046

Ping Luo

ComputeArea

- ComputeArea.java

The `boolean` Type and Operators

- Often in a program you need to compare two values, such as whether a is greater than b.
- Java provides six comparison operators (also known as relational operators) that can be used to compare two values.
- The result of the comparison is a Boolean value: true or false.

Relational Operators

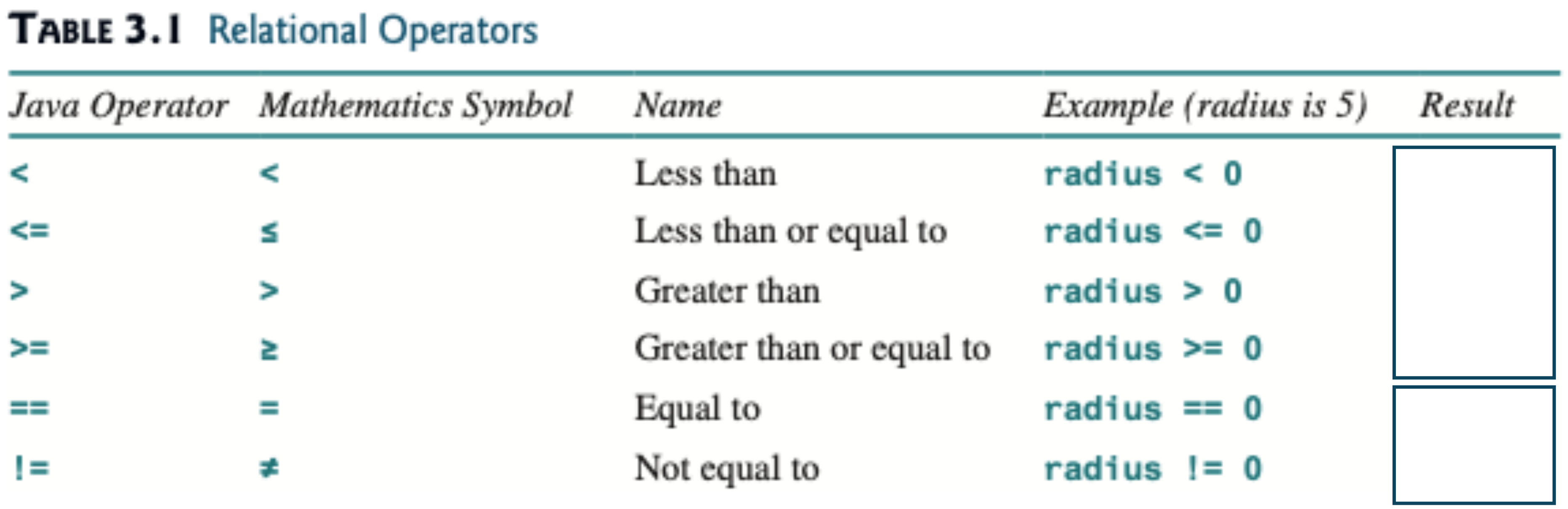


TABLE 3.1 Relational Operators

Java Operator	*Mathematics Symbol*	*Name*	*Example (radius is 5)*	*Result*
`<`	$<$	Less than	`radius < 0`	
`<=`	$\leq$	Less than or equal to	`radius <= 0`	
`>`	$>$	Greater than	`radius > 0`	
`>=`	$\geq$	Greater than or equal to	`radius >= 0`	
`==`	$=$	Equal to	`radius == 0`	
`!=`	$\neq$	Not equal to	`radius != 0`	

Problem: A Simple Math Learning Tool

- This example creates a program to let a first grader practice additions. The program generates two single-digit integers number1 and number2 and displays a question such as “What is 7 + 9?” to the student. After the student types the answer, the program displays a message to indicate whether the answer is true or false.

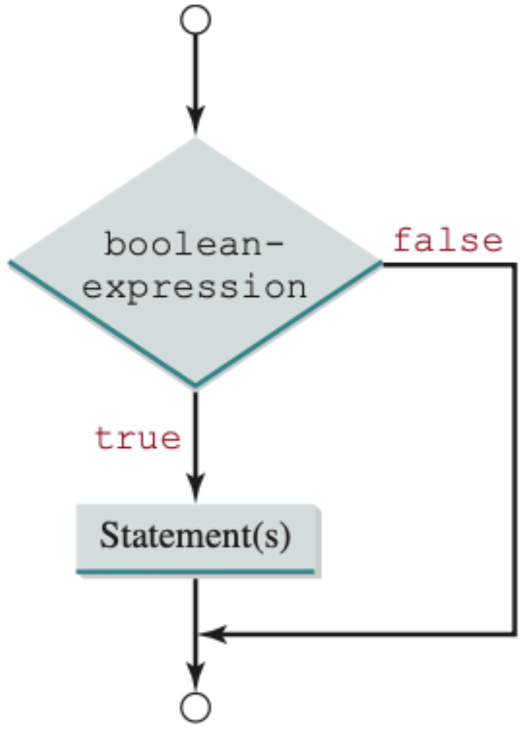
Relational Operators

- boolean b = true;
- int i = (int)b;

- int i = 1;
  boolean b = (boolean)i;

if Statement

```
if (boolean-expression) {
  statement(s);
}
```



Note

The **boolean-expression** needs to be enclosed in parentheses

```
if i > 0 {
  System.out.println("i is positive");
}
```

(a) Wrong

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(b) Correct

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(a)

Equivalent

```
if (i > 0)
  System.out.println("i is positive");
```

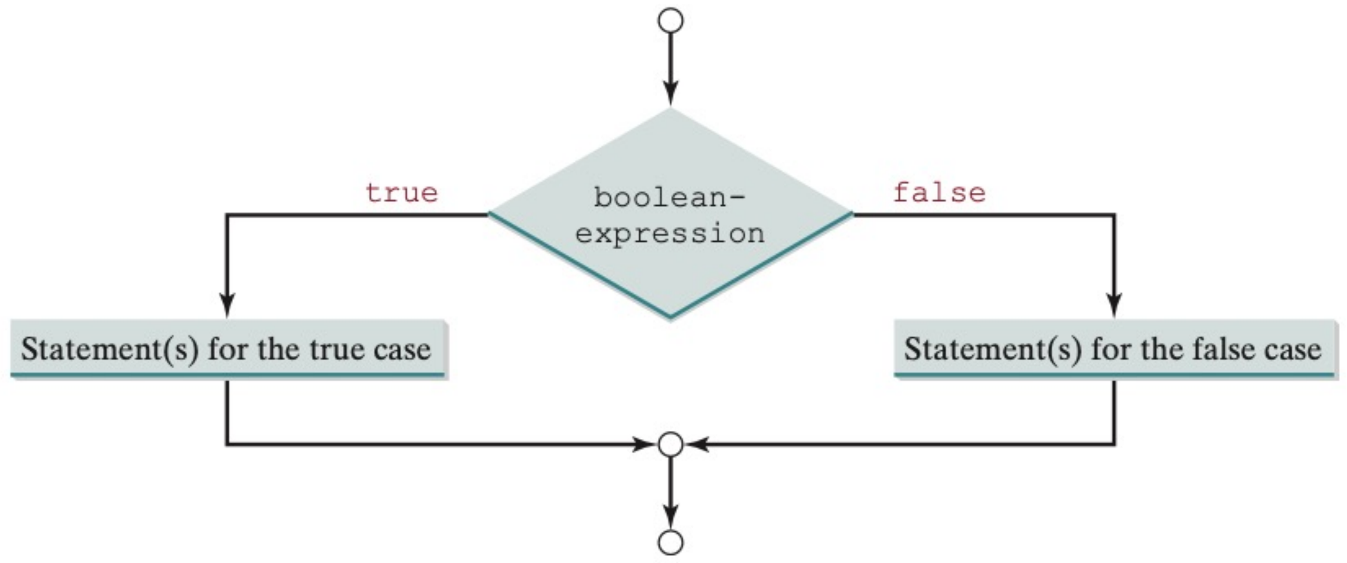
(b)

IF Demo

- Print ”yes” if a number is larger than 0

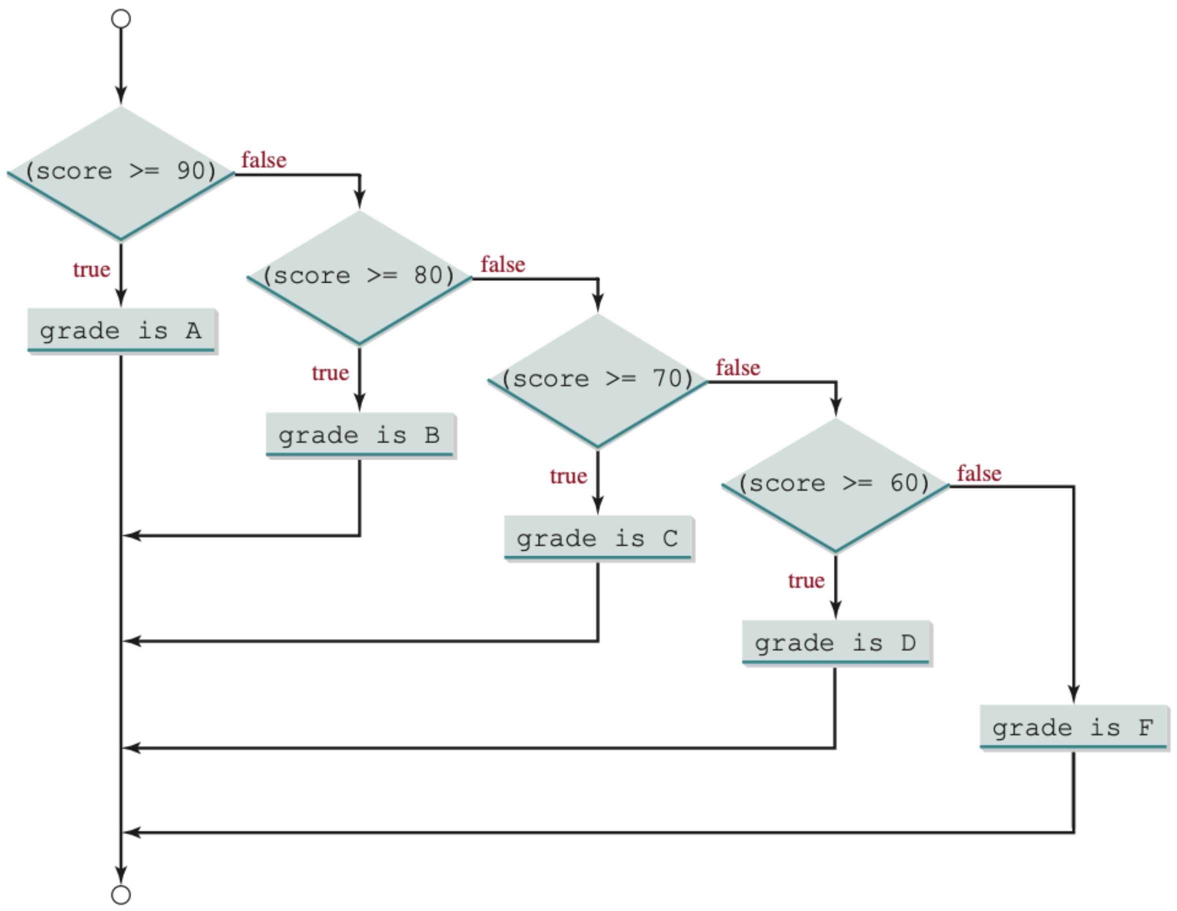
if-else Statement

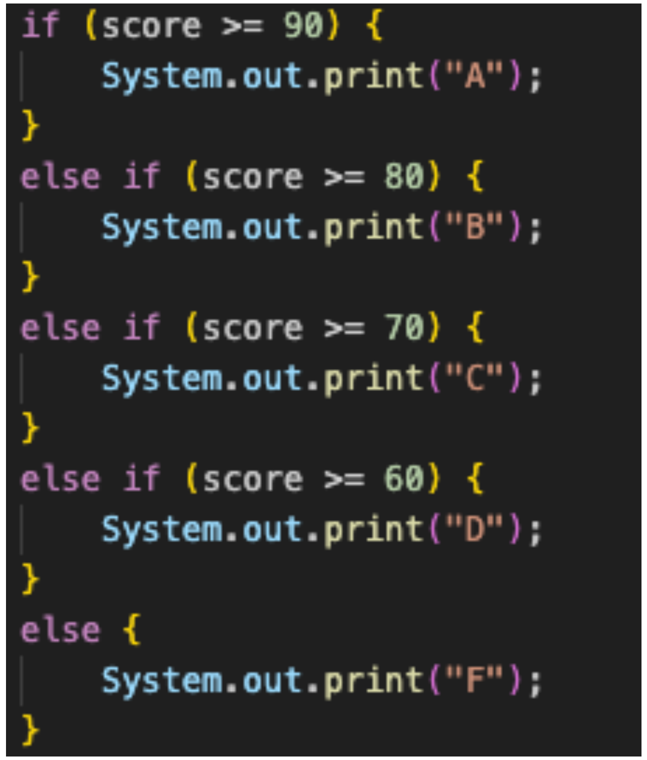
```
if (boolean-expression) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```


boolean-expression
true
false
Statement(s) for the true case
Statement(s) for the false case

ComputeArea (if-else)

Multiple if-else Statement

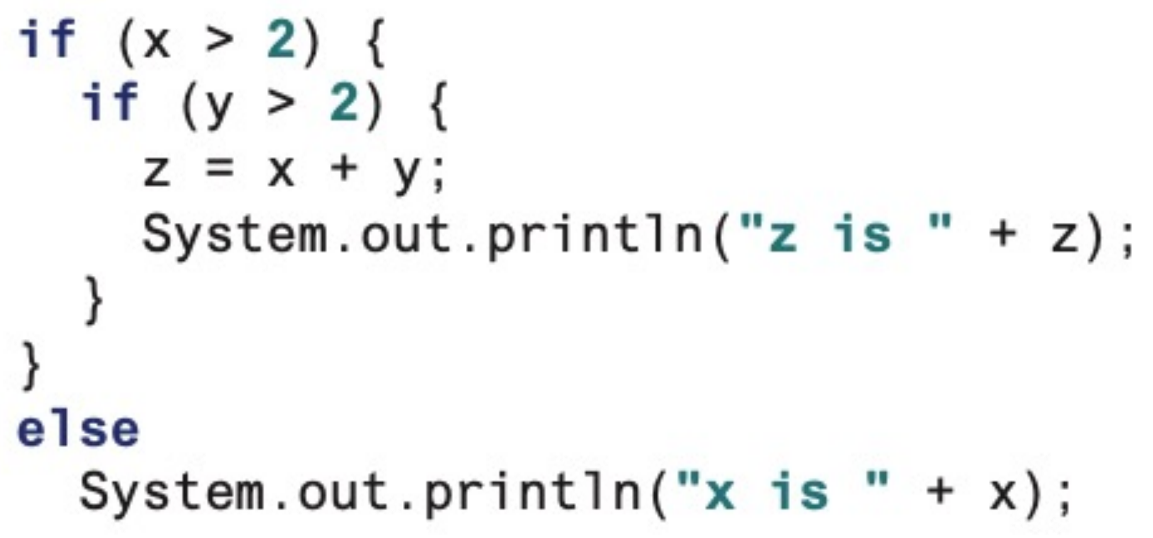




```java
if (score >= 90) {
    System.out.print("A");
}
else if (score >= 80) {
    System.out.print("B");
}
else if (score >= 70) {
    System.out.print("C");
}
else if (score >= 60) {
    System.out.print("D");
}
else {
    System.out.print("F");
}
```

Example

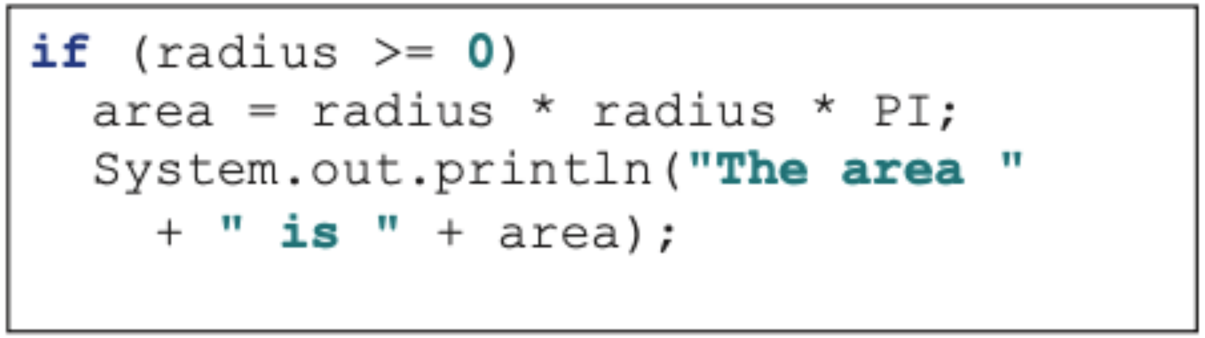
- x=3, y=2; answer?
- x=3, y=4; answer?



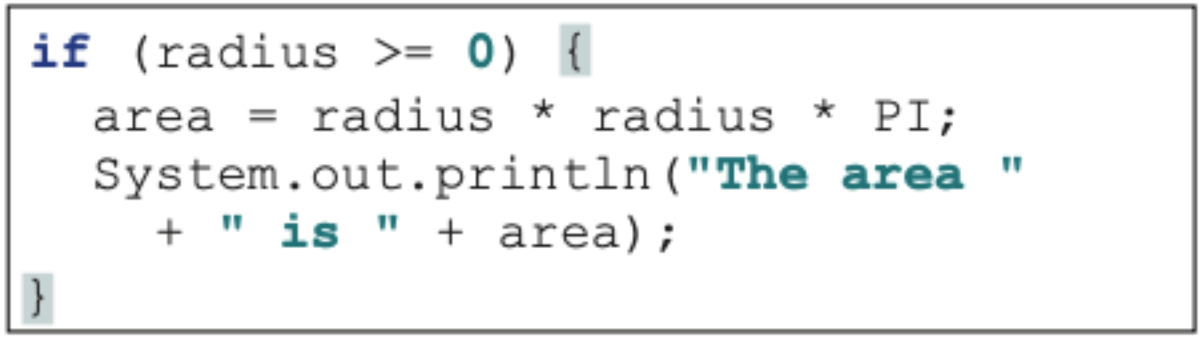
```
if (x > 2) {
  if (y > 2) {
    z = x + y;
    System.out.println("z is " + z);
  }
}
else
  System.out.println("x is " + x);
```

Common Errors

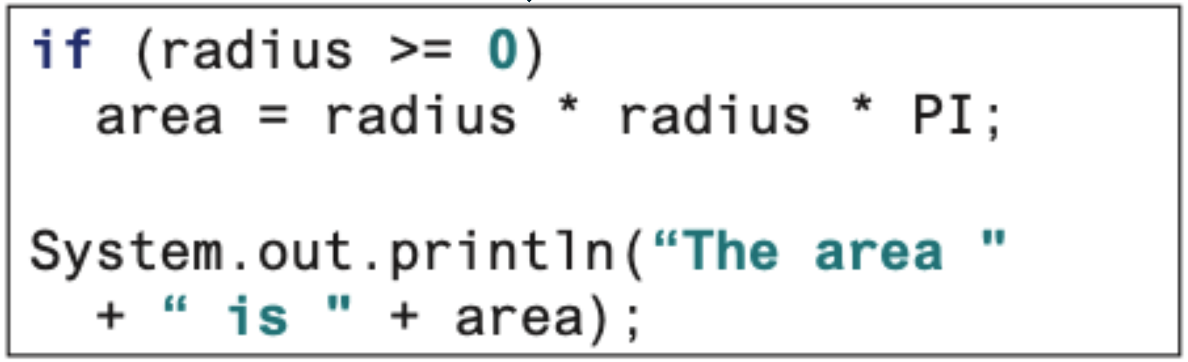
- The braces can be omitted if the block contains a single statement.



```
if (radius >= 0)
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
```



```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

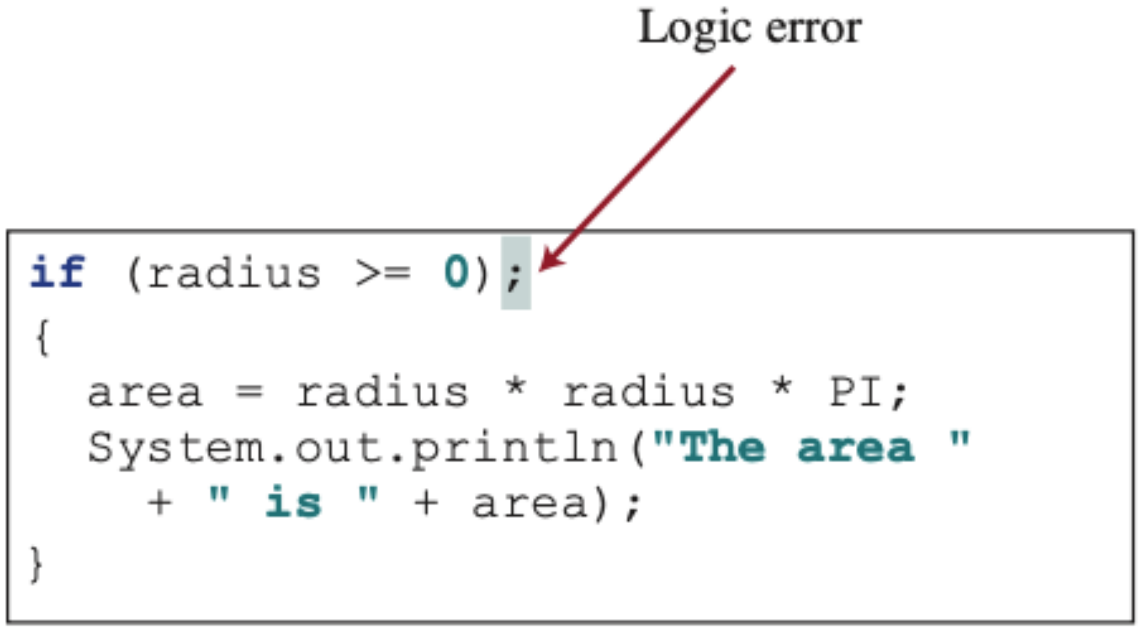


```
if (radius >= 0)
  area = radius * radius * PI;

System.out.println(“The area "
  + “ is " + area);
```

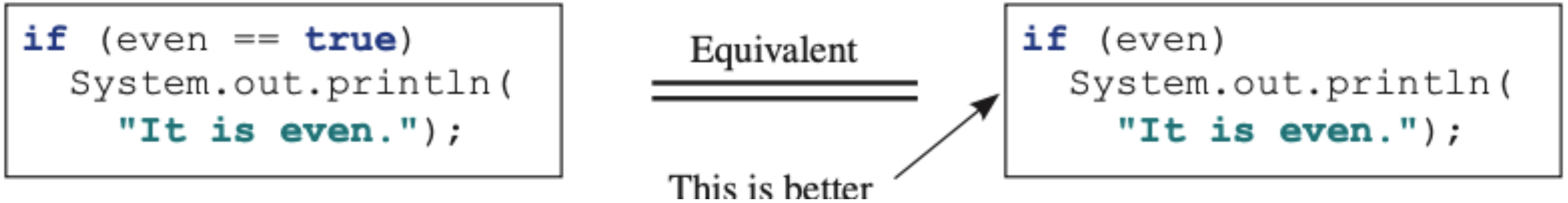
Common Errors

- Adding a semicolon at the end of an if clause is a common mistake.



```
if (radius >= 0);
{
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

Common Errors

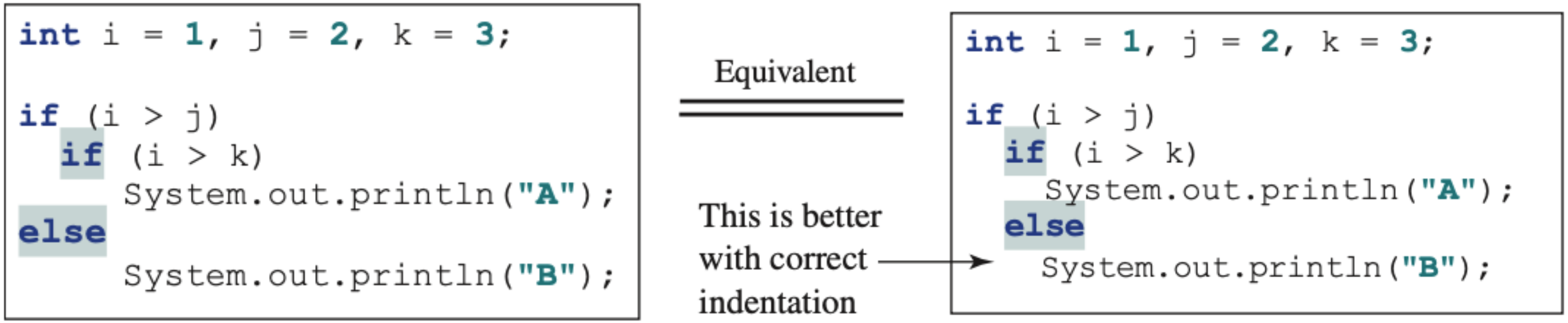


```
if (even == true)
  System.out.println(
   "It is even.");
```

```
if (even)
  System.out.println(
   "It is even.");
```

```
if (even = true)
  System.out.println("It is even.");
```

Common Errors



```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
     System.out.println("A");
else
     System.out.println("B");
```

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

The **else** clause always matches the most recent unmatched **if** clause in the same block.

Common Errors

- Equality test of two floating-point values is not reliable.
- **double** x = **1.0** − **0.1** − **0.1** − **0.1** − **0.1** − **0.1**;
- System.out.println(x == **0.5**);

- if (Math.abs(x − 0.5) < 1E-14)
  - System.out.println(x + " is approximately 0.5");

Avoid Duplicate Code

```java
if (inState) {
  tuition = 5000;
  System.out.println("The tuition is " + tuition);
}
else {
  tuition = 15000;
  System.out.println("The tuition is " + tuition);
}
```

```java
if (inState) {
  tuition = 5000;
}
else {
  tuition = 15000;
}
System.out.println("The tuition is " + tuition);
```

Problem: An Improved Math Learning Tool

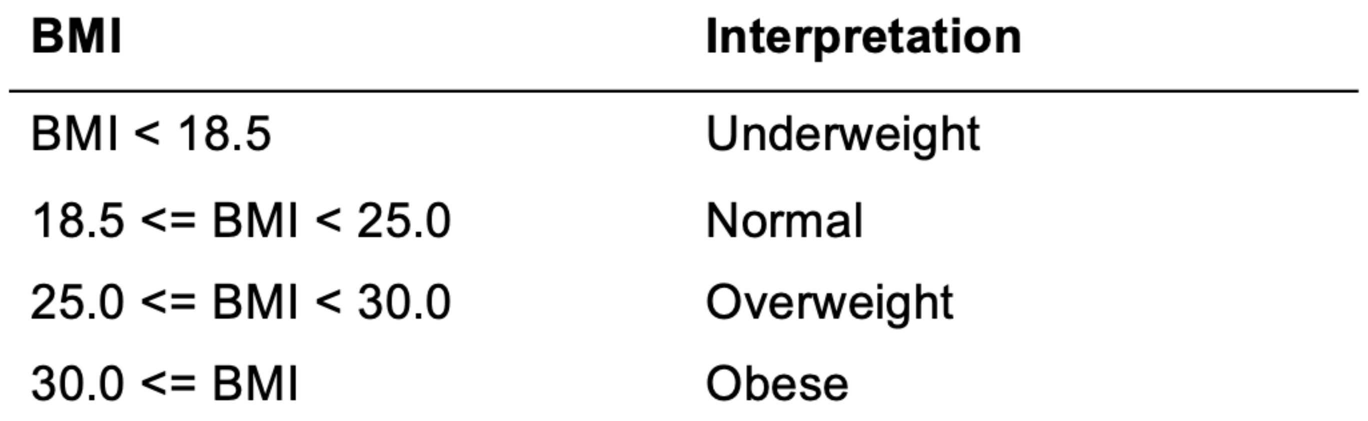
- Write a program to randomly generates two single-digit integers, number1 and number2, with number1 >= number2, and it displays to the student a question such as “What is 9 - 2?” After the student enters the answer, the program displays a message indicating whether it is correct.

- Math.random() generate a random double value between 0.0 and 1.0, excluding 1.0.

10 Minutes Break

Problem: Body Mass Index

- Body mass index (BMI) is a measure of health based on height and weight. It can be calculated by taking your weight in kilograms and dividing it by the square of your height in meters.



BMI	Interpretation
BMI < 18.5	Underweight
18.5 <= BMI < 25.0	Normal
25.0 <= BMI < 30.0	Overweight
30.0 <= BMI	Obese

Logical Operators

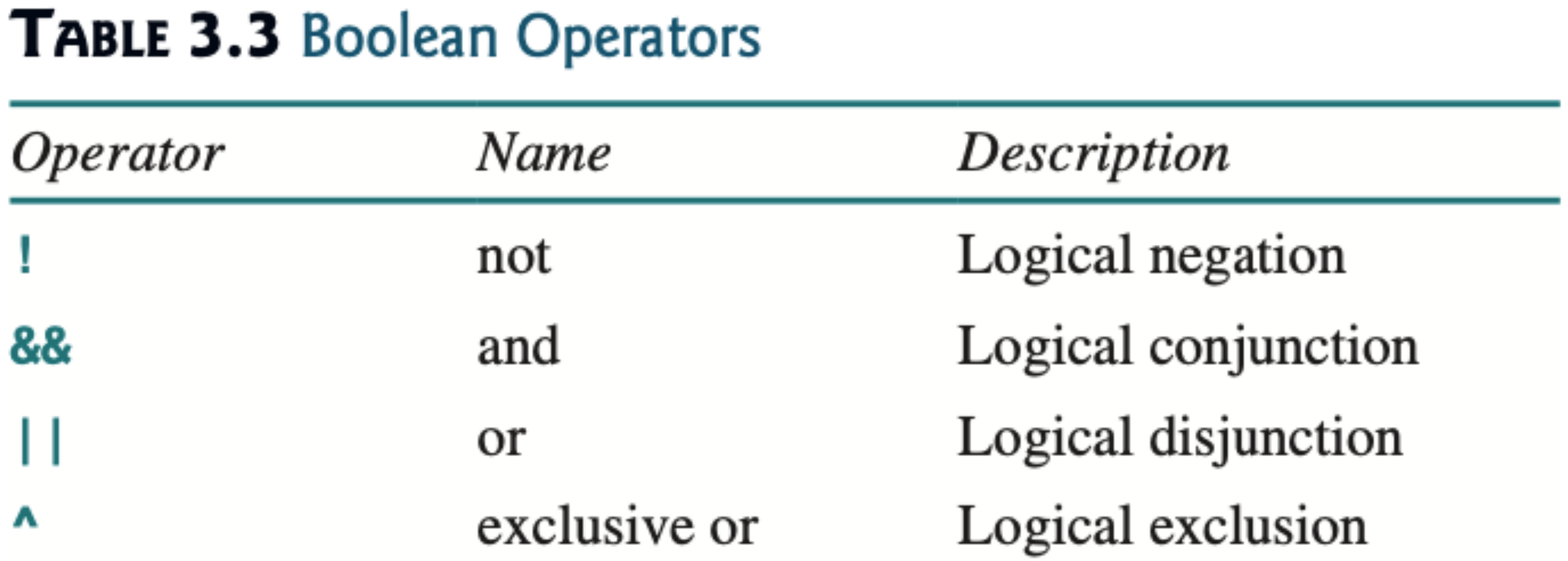
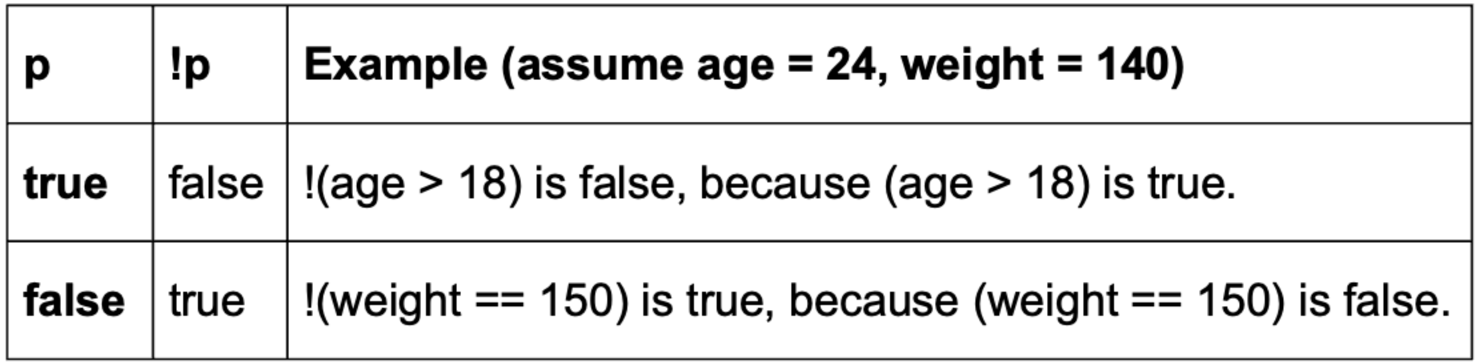


TABLE 3.3 Boolean Operators

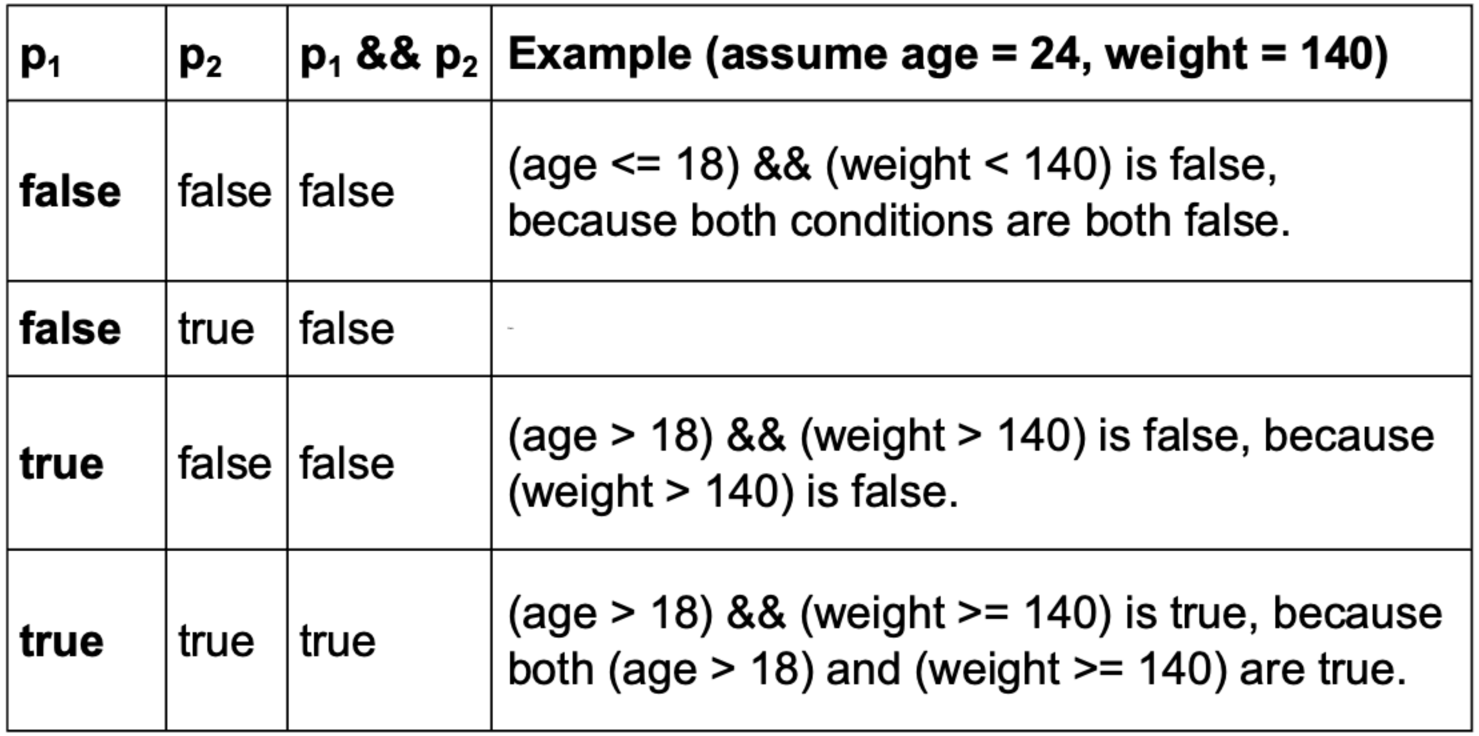
Operator	*Name*	*Description*
`!`	not	Logical negation
`&&`	and	Logical conjunction
`||`	or	Logical disjunction
`^`	exclusive or	Logical exclusion

Operator !



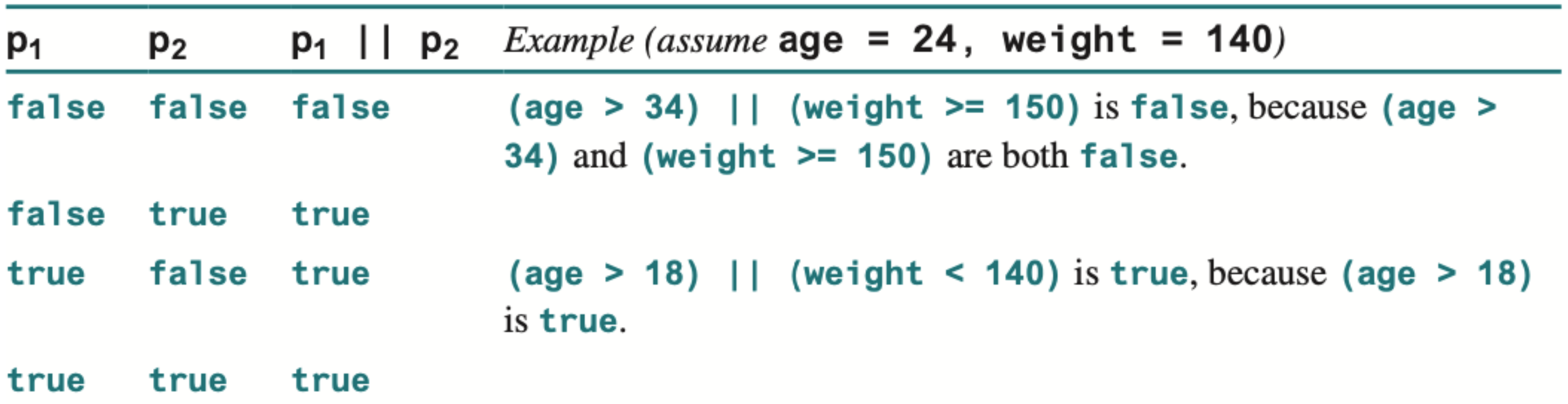
p	!p	Example (assume age = 24, weight = 140)
true	false	!(age > 18) is false, because (age > 18) is true.
false	true	!(weight == 150) is true, because (weight == 150) is false.

Operator &&



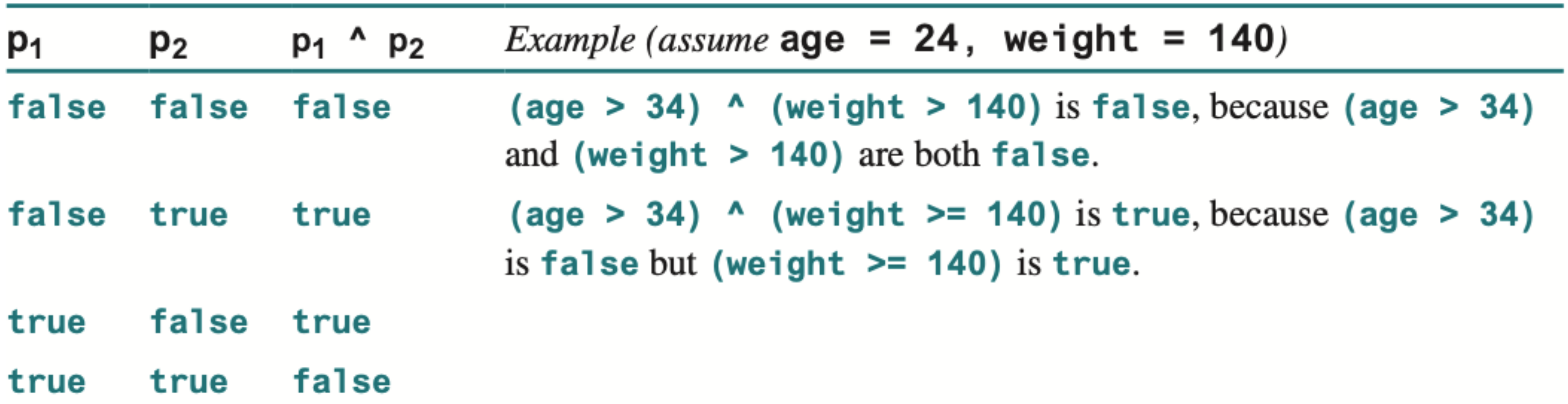
p_1	**p_2**	**p_1 && p_2**	**Example (assume age = 24, weight = 140)**
false	false	false	(age <= 18) && (weight < 140) is false, because both conditions are both false.
false	true	false	
true	false	false	(age > 18) && (weight > 140) is false, because (weight > 140) is false.
true	true	true	(age > 18) && (weight >= 140) is true, because both (age > 18) and (weight >= 140) are true.

Operator ||



p_1	p_2	p_1 `||` p_2	*Example (assume* `age = 24, weight = 140`*)*
`false`	`false`	`false`	`(age > 34) || (weight >= 150)` is `false`, because `(age > 34)` and `(weight >= 150)` are both `false`.
`false`	`true`	`true`	
`true`	`false`	`true`	`(age > 18) || (weight < 140)` is `true`, because `(age > 18)` is `true`.
`true`	`true`	`true`	

Operator ^



p_1	p_2	p_1 ^ p_2	*Example (assume* `age = 24, weight = 140`*)*
`false`	`false`	`false`	`(age > 34) ^ (weight > 140)` is `false`, because `(age > 34)` and `(weight > 140)` are both `false`.
`false`	`true`	`true`	`(age > 34) ^ (weight >= 140)` is `true`, because `(age > 34)` is `false` but `(weight >= 140)` is `true`.
`true`	`false`	`true`	
`true`	`true`	`false`	

Example

- Write a program that checks whether a number is divisible by 2 and 3, whether a number is divisible by 2 or 3, and whether a number is divisible by 2 or 3 but not both:

Note

- **28** <= numberOfDaysInAMonth <= **31**

Problem: Determining Leap Year

- A leap year has 366 days. The February of a leap year has 29 days.
- Hint:

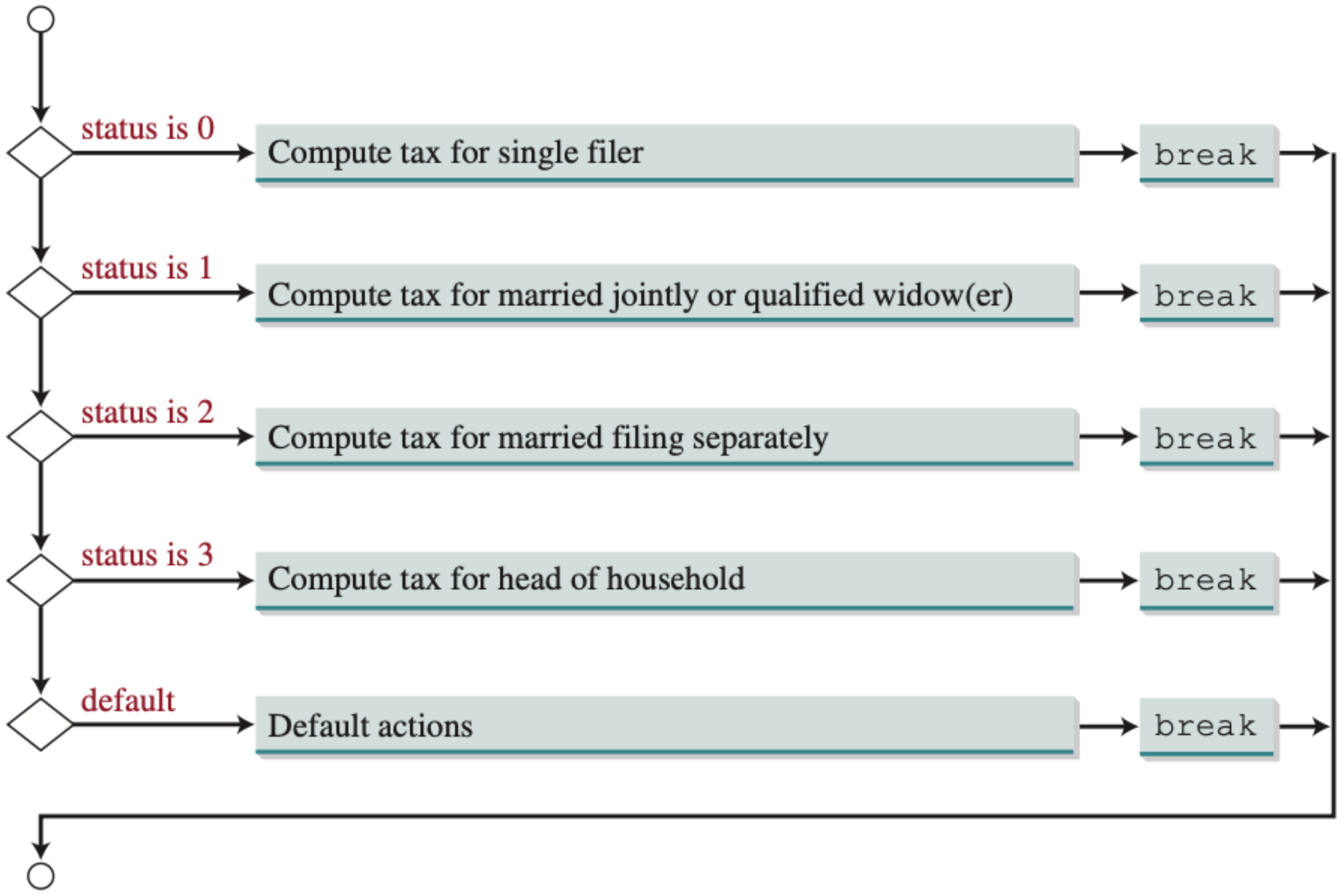
1. A leap year is divisible by 4
2. A leap year is divisible by 4 but not by 100
3. A leap year is divisible by 4 but not by 100 or divisible by 400

switch Statements

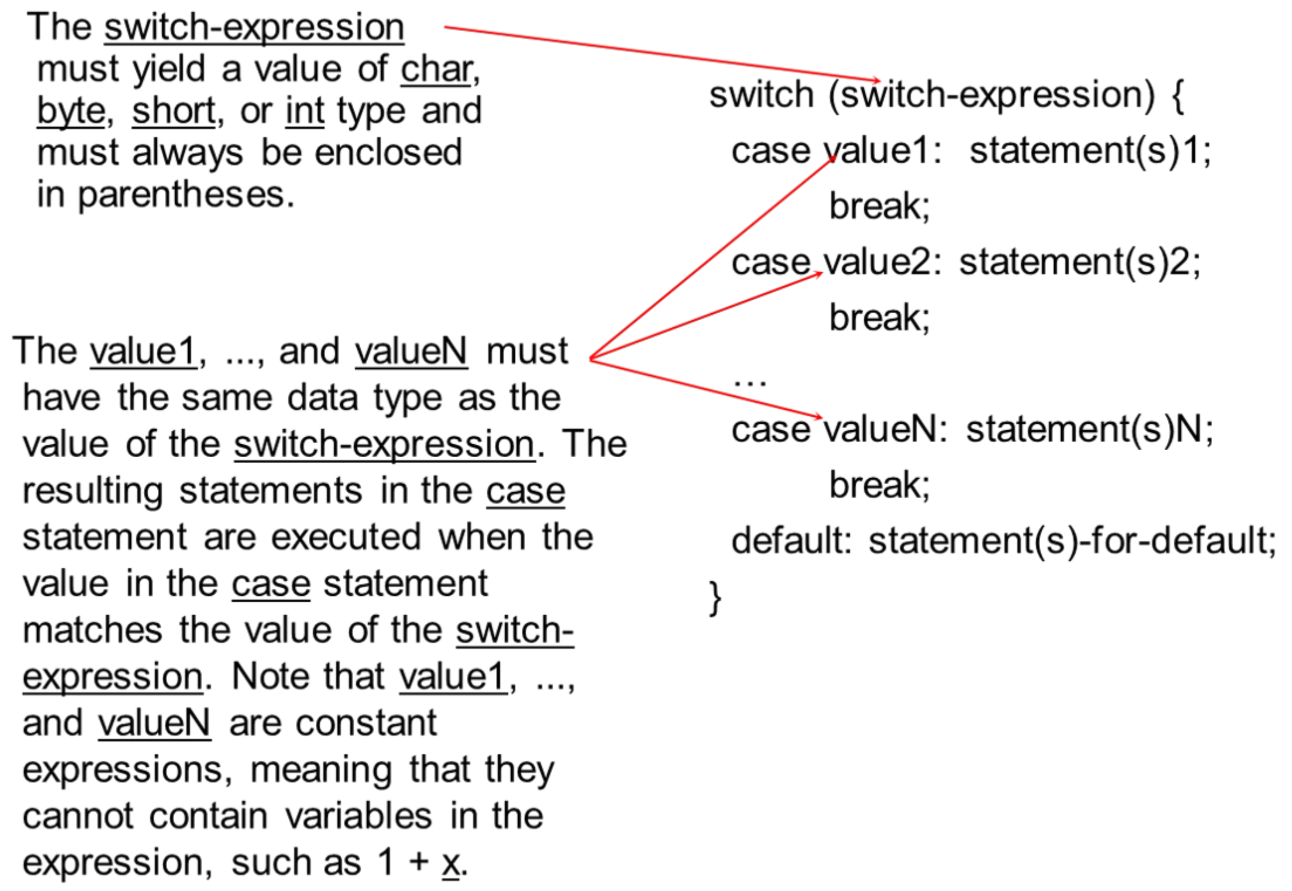
- A switch statement executes statements based on the value of a variable or an expression.

```
switch (status) {
  case 0:  compute tax for single filers;
           break;
  case 1:  compute tax for married jointly or qualifying widow(er);
           break;
  case 2:  compute tax for married filing separately;
           break;
  case 3:  compute tax for head of household;
           break;
  default: System.out.println("Error: invalid status");
           System.exit(1);
}
```

switch Statements


status is 0
Compute tax for single filer
break
status is 1
Compute tax for married jointly or qualified widow(er)
break
status is 2
Compute tax for married filing separately
break
status is 3
Compute tax for head of household
break
default
Default actions
break

switch Statements



The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.

The value1, ..., and valueN must have the same data type as the value of the switch-expression. The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression. Note that value1, ..., and valueN are constant expressions, meaning that they cannot contain variables in the expression, such as 1 + x.

```
switch (switch-expression) {
  case value1:  statement(s)1;
           break;
  case value2: statement(s)2;
           break;
  …
  case valueN: statement(s)N;
           break;
  default: statement(s)-for-default;
}
```

The keyword <u>break</u> is optional, but it should be used at the end of each case in order to terminate the remainder of the <u>switch</u> statement. If the <u>break</u> statement is not present, the next <u>case</u> statement will be executed.

The <u>default</u> case, which is optional, can be used to perform actions when none of the specified cases matches the <u>switch-expression</u>.

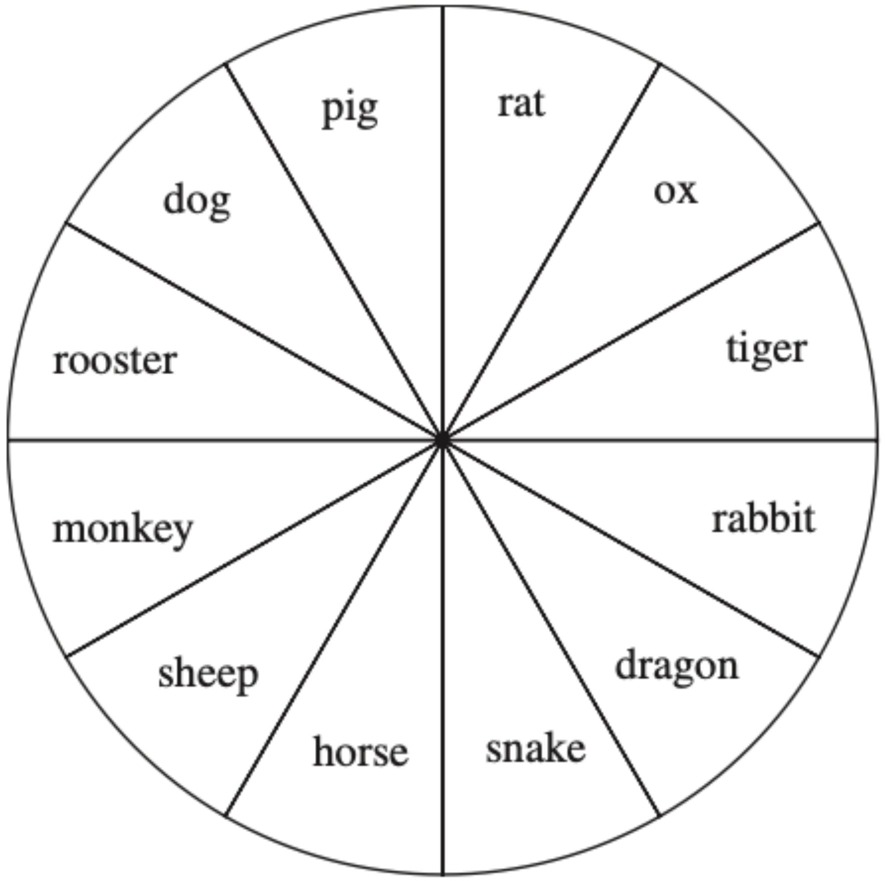
```
switch (switch-expression) {
  case value1:  statement(s)1;
           break;
  case value2: statement(s)2;
           break;
  …
  case valueN: statement(s)N;
           break;
  default: statement(s)-for-
default;
}
```

When the value in a **case** statement matches the value of the **switch-expression**, the statements *starting from this case* are executed until either a **break** statement or the end of the **switch** statement is reached.

Note

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println(“Weekday"); break;
  case 0:
  case 6: System.out.println(“Weekend");
}
```

Problem: Zodiac



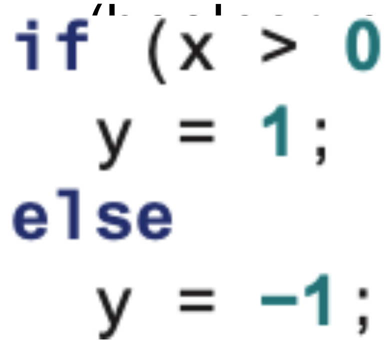
`year % 12 =`

0: monkey
1: rooster
2: dog
3: pig
4: rat
5: ox
6: tiger
7: rabbit
8: dragon
9: snake
10: horse
11: sheep

Conditional Operators

- A conditional operator evaluates an expression based on a condition.

  (boolean-expression) ? expression1 : expression2



```
if (x > 0)
   y = 1;
else
   y = -1;
```

- y = (x > 0)? 1: −1;

Operator Precedence

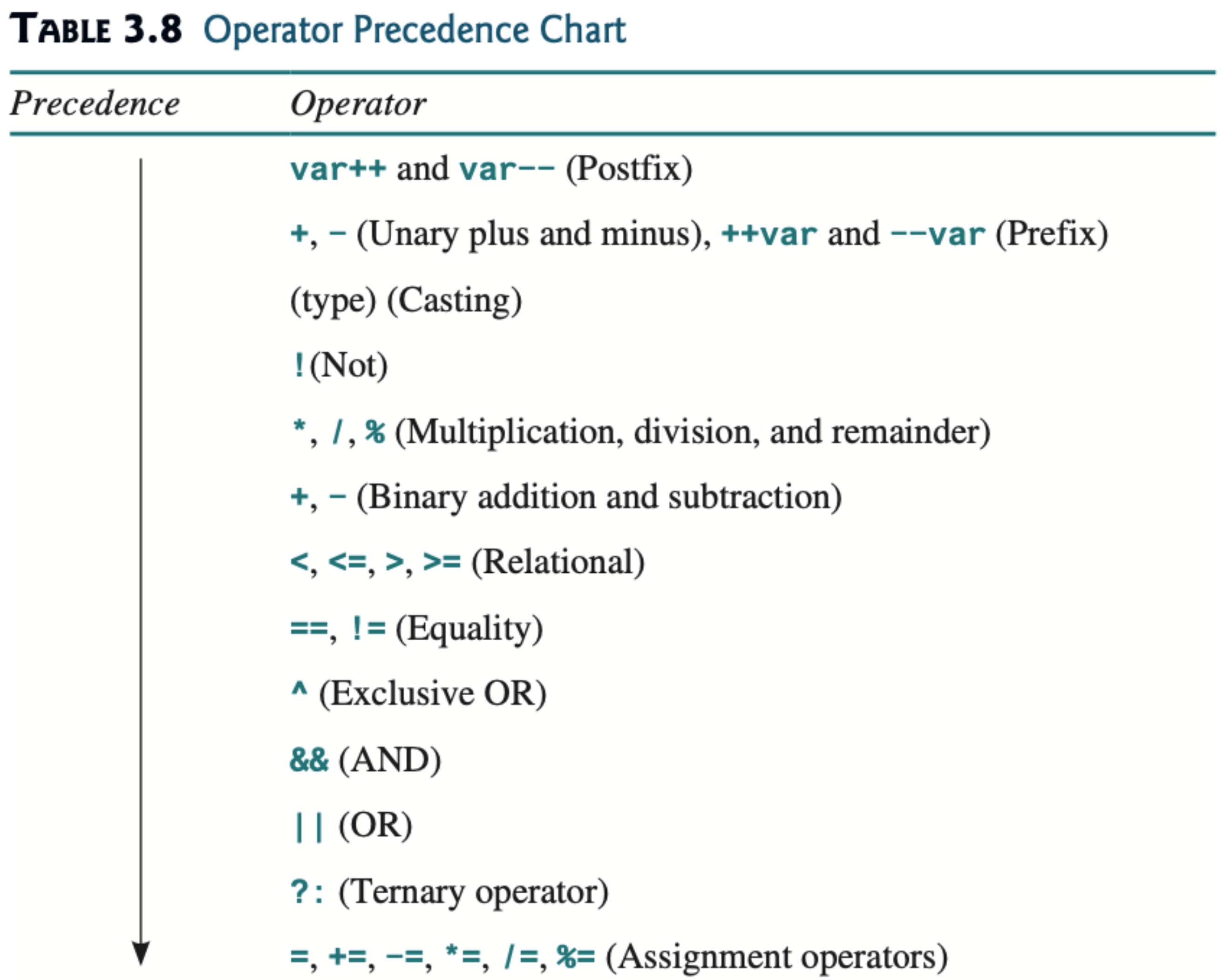


Table 3.8 Operator Precedence Chart

Precedence	*Operator*
↓	`var++` and `var--` (Postfix)
	`+`, `-` (Unary plus and minus), `++var` and `--var` (Prefix)
	(type) (Casting)
	`!`(Not)
	`*`, `/`, `%` (Multiplication, division, and remainder)
	`+`, `-` (Binary addition and subtraction)
	`<`, `<=`, `>`, `>=` (Relational)
	`==`, `!=` (Equality)
	`^` (Exclusive OR)
	`&&` (AND)
	`||` (OR)
	`?:` (Ternary operator)
	`=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operators)

Debugging

- Logic errors are called **bugs**. The process of finding and correcting errors is called debugging.
- A common approach to debugging is to use a combination of methods to narrow down to the part of the program where the bug is located.
- You can hand-trace the program (i.e., catch errors by reading the program), or you can insert print statements in order to show the values of the variables or the execution flow of the program. This approach might work for a short, simple program. But for a large, complex program, the most effective approach for debugging is to use a debugger utility.

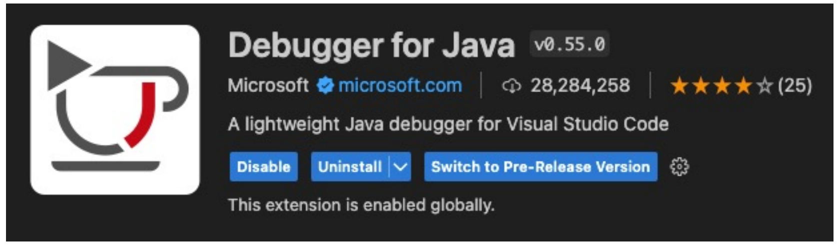
Debugger

Debugger is a program that facilitates debugging. You can use a debugger to

- Executing a single statement at a time: The debugger allows you to execute one statement at a time so that you can see the effect of each statement.
- Tracing into or stepping over a method: If a method is being executed, you can ask the debugger to enter the method and execute one statement at a time in the method, or you can ask it to step over the entire method. You should step over the entire method if you know that the method works. For example, always step over system-supplied methods, such as System.out.println().
- Setting breakpoints: You can also set a breakpoint at a specific statement. Your program pauses when it reaches a breakpoint. You can set as many breakpoints as you want. Breakpoints are particularly useful when you know where your programming error starts. You can set a breakpoint at that statement, and have the program execute until it reaches the breakpoint.
- Displaying variables: The debugger lets you select several variables and display their values. As you trace through a program, the content of a variable is continuously updated.
- Displaying call stacks: The debugger lets you trace all of the method calls. This feature is helpful when you need to see a large picture of the program-execution flow.
- Modifying variables: Some debuggers enable you to modify the value of a variable when debugging. This is convenient when you want to test a program with different samples, but do not want to leave the debugger.

Debugger in VScode

Tutorial: https://code.visualstudio.com/docs/java/java-debugging



Debugger for Java v0.55.0

Microsoft microsoft.com | 28,284,258 | ★★★★☆ (25)

A lightweight Java debugger for Visual Studio Code

This extension is enabled globally.

Breakpoint example

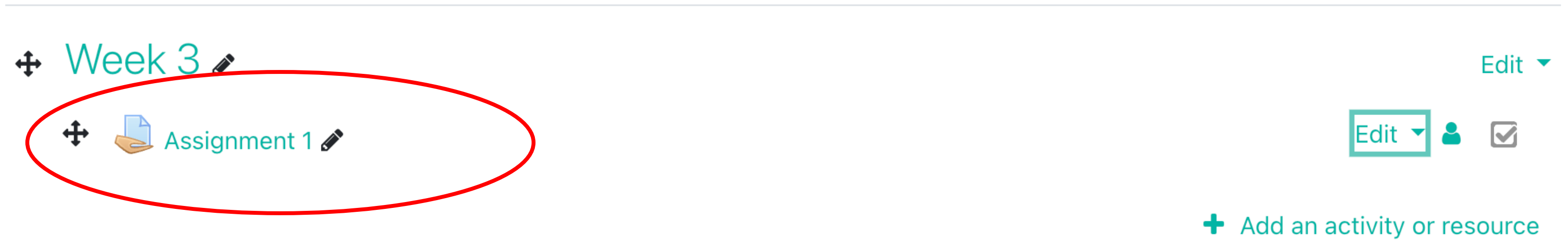
Summary

- Boolean type
- if-else, switch statement
- logic operator
- Parentheses can be used to force the order of evaluation to occur in any sequence.

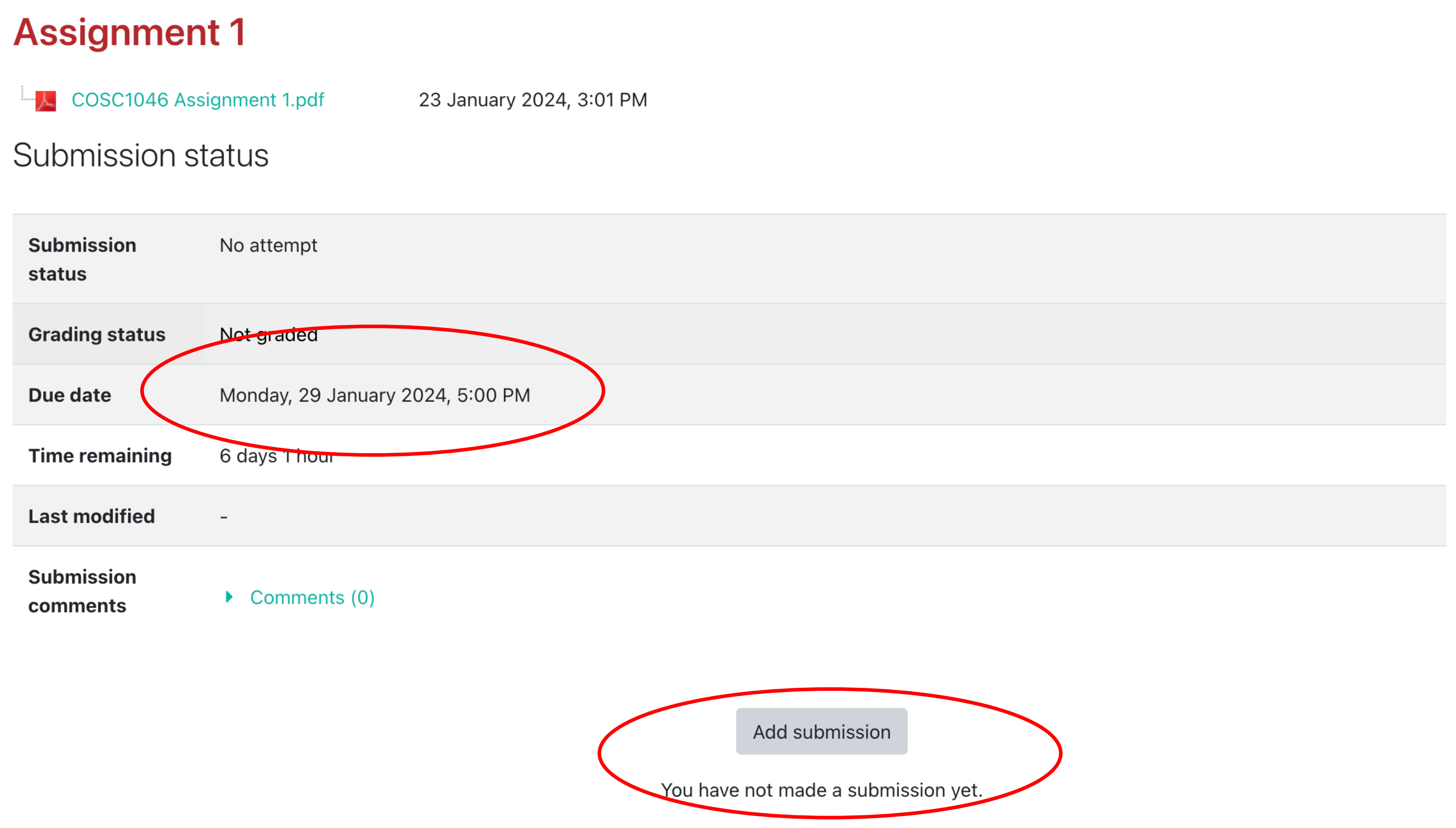
Q&A

Assignment 1

- Due: Jan 29 (Monday) 5:00 pm



Assignment 1



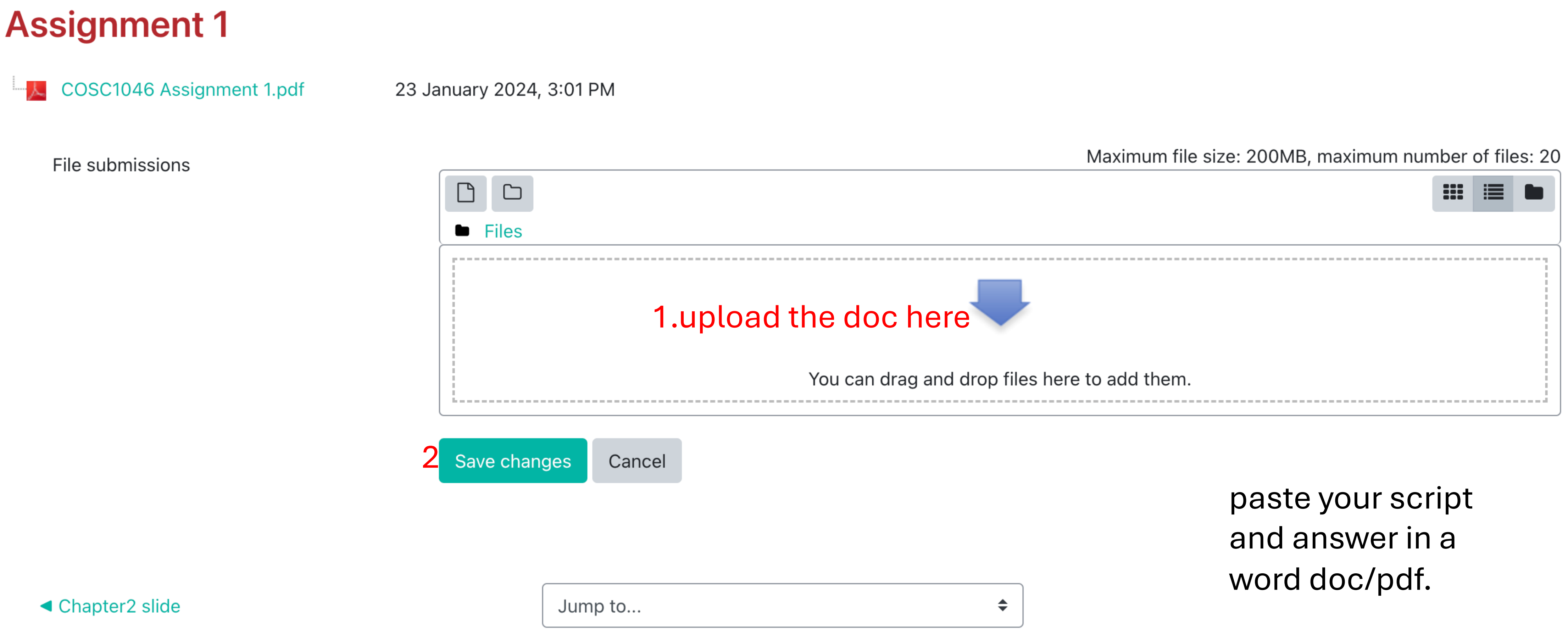
Assignment 1

COSC1046 Assignment 1.pdf 23 January 2024, 3:01 PM

Submission status

Submission status	No attempt
Grading status	Not graded
Due date	Monday, 29 January 2024, 5:00 PM
Time remaining	6 days 1 hour
Last modified	-
Submission comments	Comments (0)

You have not made a submission yet.



Assignment 1

COSC1046 Assignment 1.pdf 23 January 2024, 3:01 PM

File submissions

Maximum file size: 200MB, maximum number of files: 20

Files

1.upload the doc here

You can drag and drop files here to add them.

2 Save changes Cancel

◀ Chapter2 slide

Jump to...

paste your script and answer in a word doc/pdf.