Lecture #5
CSC 200-04L Fall 2018

Ambrose Lewis
tjl274@email.vccs.edu

Agenda

- Javadoc
  - Introduction
  - What is an API?
  - How to Navigate
  - Example
- Java Selection

Agenda

- Javadoc
  - Introduction
  - What is an API?
  - How to Navigate
  - Example
- Java Selection

Javadoc: Introduction

- What is Javadoc?
  - Javadoc is a tool that you get with the Java Developer Kit (JDK).
  - This tool generates HTML API documentation for your Java source code based in specially formed comments
    - HTML = Hypertext Markup Language (web page)
    - API = Application Programming Interface (described on the next page)
  - Javadoc output files exist for all of the classes provided in the Java language.
    - You can find them on-line or you can download onto your computer
    - This is a quick and easy way to find classes and to learn (or remember) how to use these classes.
- Javadoc output is created directly from a set of Java source files.
  - This tightly couples the documentation to the source code, which is a very good thing. (In the past, documentation often got out of synch with the code)
  - The Javadoc tool looks for specially formatted Java comments within the source code.
  - Today we will cover how to read and use the Java language's Javadoc files. After we discuss Object Orientate Programming, we will learn how to make Javadoc files for our own Java programs

From: http://en.wikipedia.org/wiki/Javadoc

Javadoc: What is an API?

- API = Application Programming Interface, documentation for a programmer
- Formal Definition:
  - an API specifies a software component in terms of its operations, the operation's inputs and outputs, and underlying types.
  - Its main purpose is to define a set of functionalities that are independent of their implementation, allowing both definition and implementation to vary without compromising each other.
- What does the API tell the programmer?
  - Describes the component (answers what is it? what does it do?
  - Lists the different ways to create it
  - Lists the methods or functions that you can call and how to get data into / out of them
- In Java, the software component is a Java Class

API Formal Def from: http://en.wikipedia.org/wiki/Application_programming_interface

Java Organization: Object

- In order to navigate Javadoc output, you'll need to understand a little more about the organization of Java…Let's define some terms:
- What is an object?
  - In Object Orientated Programming, an object is a software bundle of related state and behavior.
    - Software objects are often used to model the real-world objects that you find in everyday life.
    - Dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail).
    - Bicycles also have state (current gear, current pedal cadence, current speed) and behavior (changing gear, changing pedal cadence, applying brakes).
    - Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.

Based on: http://docs.oracle.com/javase/tutorial/java/concepts/index.html

Java Organization: Class

- What is a Class? A class is the blueprint from which individual objects are created.
  - In the real world, you'll often find many individual objects all of the same kind.
  - There may be thousands of other bicycles in existence, all of the same make and model.
  - Each bicycle was built from the same set of blueprints and therefore contains the same components.
  - In object-oriented terms, we say that your bicycle is an instance of the class of objects known as bicycles.

Based on: http://docs.oracle.com/javase/tutorial/java/concepts/index.html

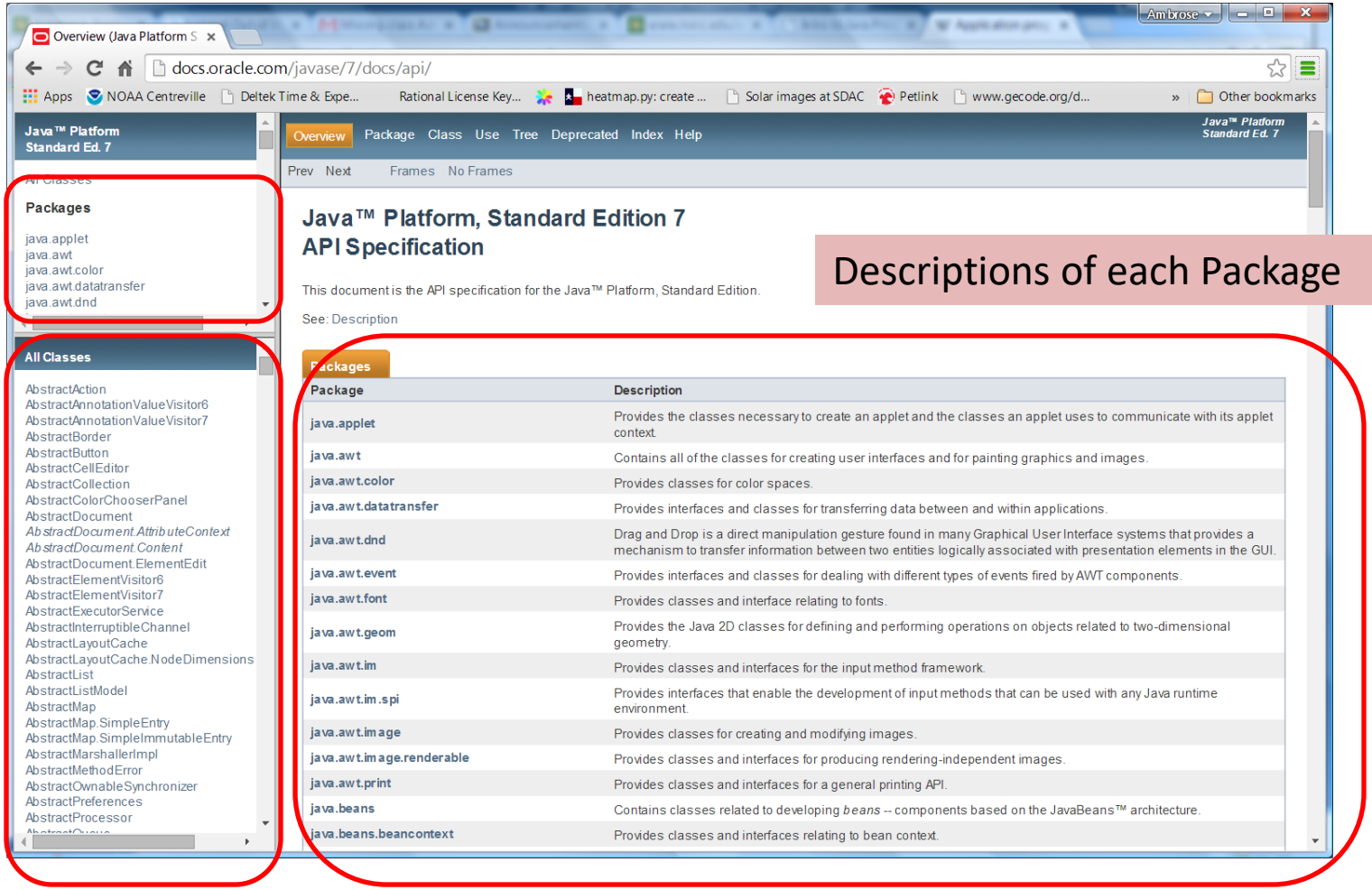
Java Organization: Package

- A Package is a container that holds a set of related classes
  - Conceptually you can think of packages as being similar to different folders on your computer.
- Some example Java packages:
  - Java.lang (basics of the Java language, like base data types, Math, etc.)
  - Java.io (getting input & sending output, like reading/writing files, etc.)
  - Java.utils (utilities that help with things like times and dates, and other miscellaneous utilities (a string tokenizer, a random-number generator, etc.)

Javadoc: How to Navigate

List of Package Names

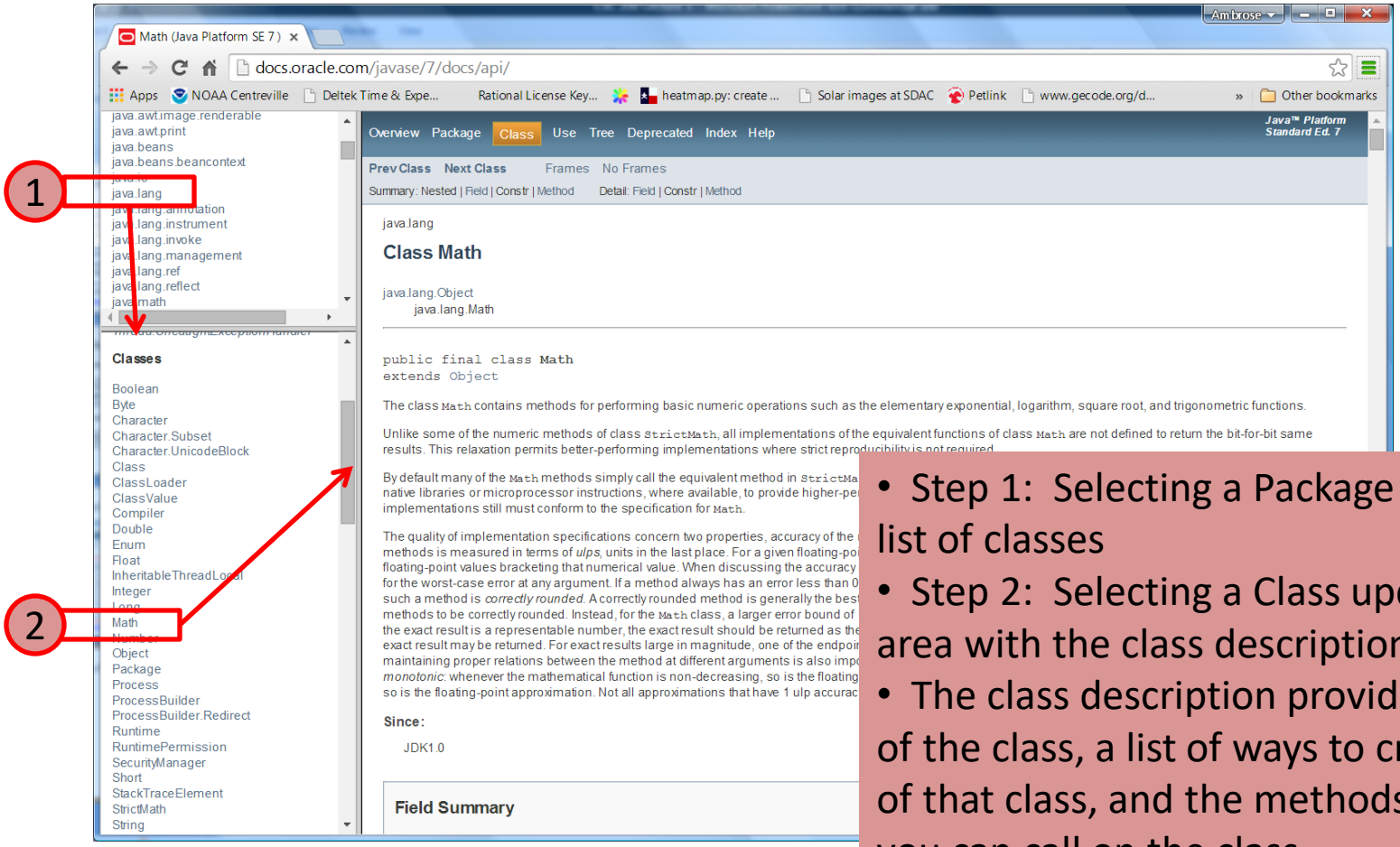
List of Class Names, can be filtered by Package



Descriptions of each Package

See http://docs.oracle.com/javase/7/docs/api/

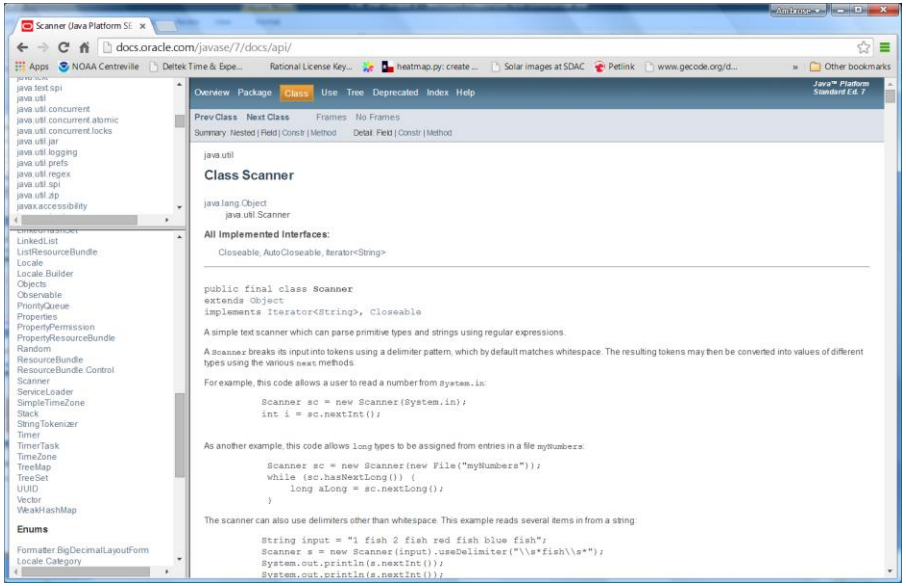
Javadoc: How to Navigate



- Step 1: Selecting a Package updates the list of classes
- Step 2: Selecting a Class updates the main area with the class description
- The class description provides an overview of the class, a list of ways to create instances of that class, and the methods (or functions) you can call on the class

Javadoc Example: Scanner

- The Java Scanner Class is a part of the java.util package
- We can quickly learn what a scanner does, how to create an instance of one, and how to use it in our program



Agenda

- Javadoc
  - Introduction
  - What is an API?
  - How to Navigate
  - Example
- Java Selection

Selection: Motivation

- Suppose you created a program to compute the area of a circle from a user supplied radius.
- If the user gave a negative value for the radius, the program would print an invalid result.
- If the radius is negative, you don't want the program to compute the area.
- How can you deal with this situation? Via Selection

Objectives

- To declare **boolean** variables and write Boolean expressions using relational operators (§3.2).
- To implement selection control using one-way **if** statements (§3.3).
- To implement selection control using two-way **if-else** statements (§3.4).
- To implement selection control using nested **if** and multi-way **if** statements (§3.5).
- To avoid common errors and pitfalls in **if** statements (§3.6).
- To generate random numbers using the **Math.random()** method (§3.7).
- To program using selection statements for a variety of examples (**SubtractionQuiz**, **BMI**, **ComputeTax**) (§§3.7–3.9).
- To combine conditions using logical operators (**&&**, **||**, and **!**) (§3.10).
- To program using selection statements with combined conditions (**LeapYear**, **Lottery**) (§§3.11–3.12).
- To implement selection control using **switch** statements (§3.13).
- To write expressions using the conditional expression (§3.14).
- To examine the rules governing operator precedence and associativity (§3.15).
- To apply common techniques to debug errors (§3.16).

The `boolean` Type and Operators

Often in a program you need to compare two values, such as whether i is greater than j. 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.

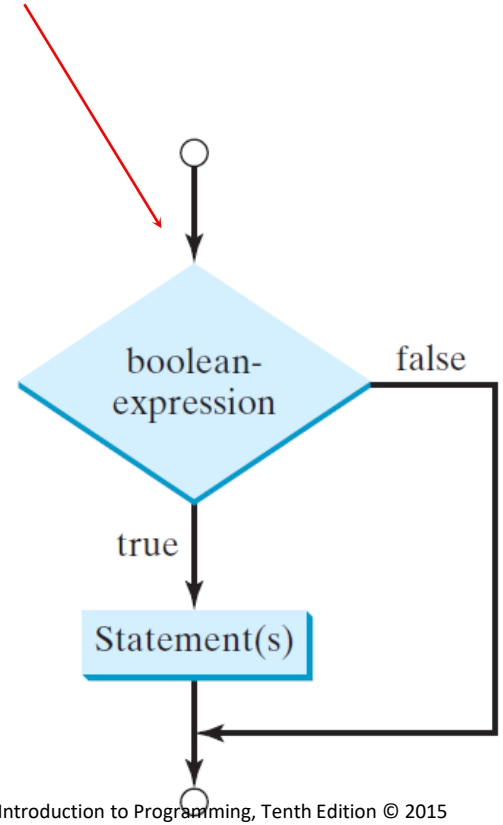
```
boolean b = (1 > 2);
```

Relational Operators

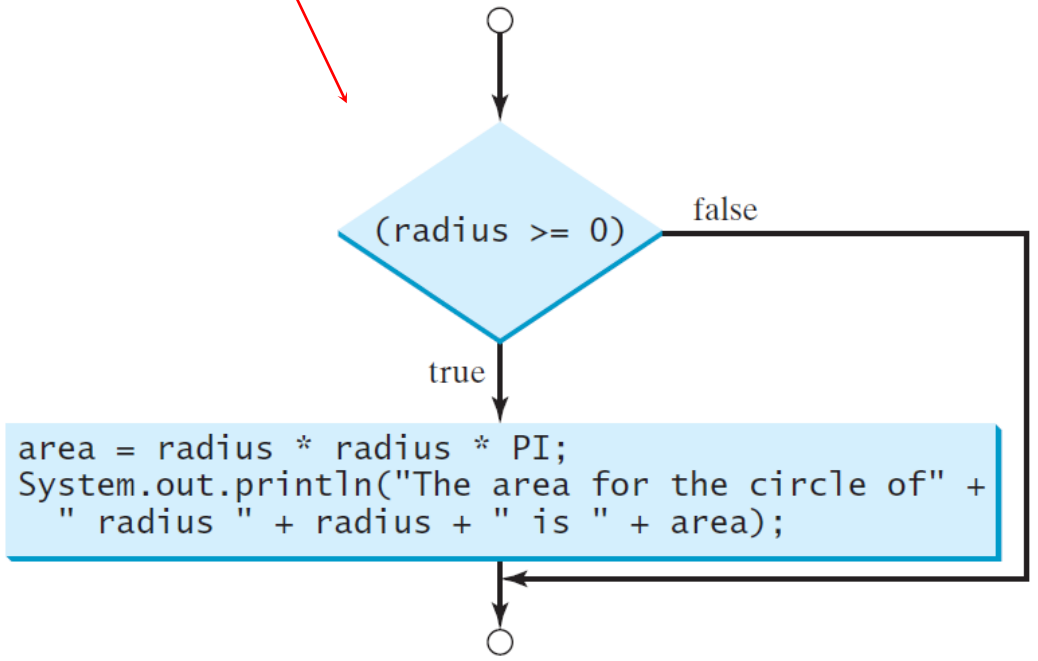
Java Operator	Mathematics Symbol	Name	Example (radius is 5)	Result
`<`	<	less than	`radius < 0`	`false`
`<=`	≤	less than or equal to	`radius <= 0`	`false`
`>`	>	greater than	`radius > 0`	`true`
`>=`	≥	greater than or equal to	`radius >= 0`	`true`
`==`	=	equal to	`radius == 0`	`false`
`!=`	≠	not equal to	`radius != 0`	`true`

One-way `if` Statements

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




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



Note

```
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)

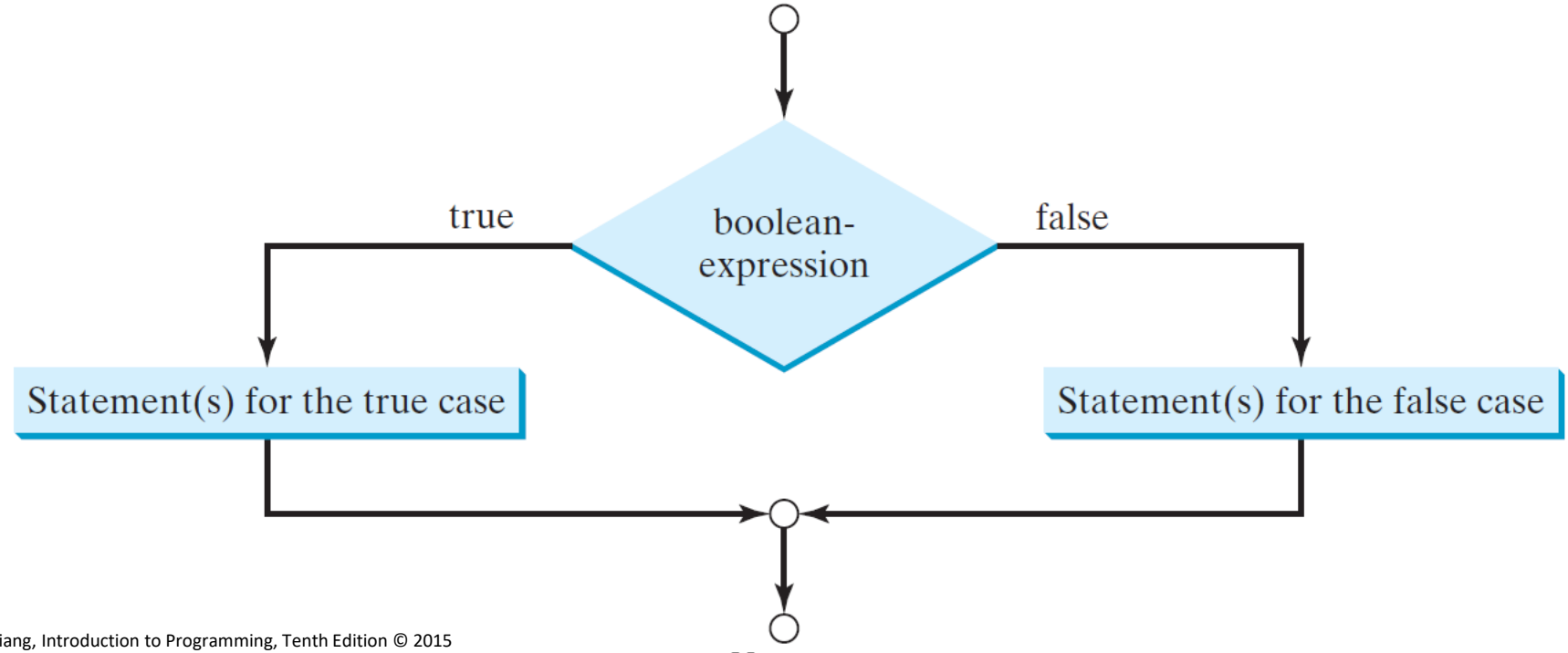
Note: This is legal Java syntax, however I recommend you always put the braces in, like shown on the left!

Simple if Demo

Write a program that prompts the user to enter an integer. If the number is a multiple of 5, print “Hi Five”. If the number is divisible by 2, print “Hi Even”.

The Two-way `if` Statement

```
if (boolean-expression) {
  "true" statement(s)
}
else {
  "false" statement (s)
}
```



if-else Example

```
if (radius >= 0) {
  area = radius * radius * 3.14159;

  System.out.println("The area for the “
    + “circle of radius " + radius +
    " is " + area);
}
else {
  System.out.println("Negative input");
}
```

Multiple Alternative if Statements

```
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

(a)

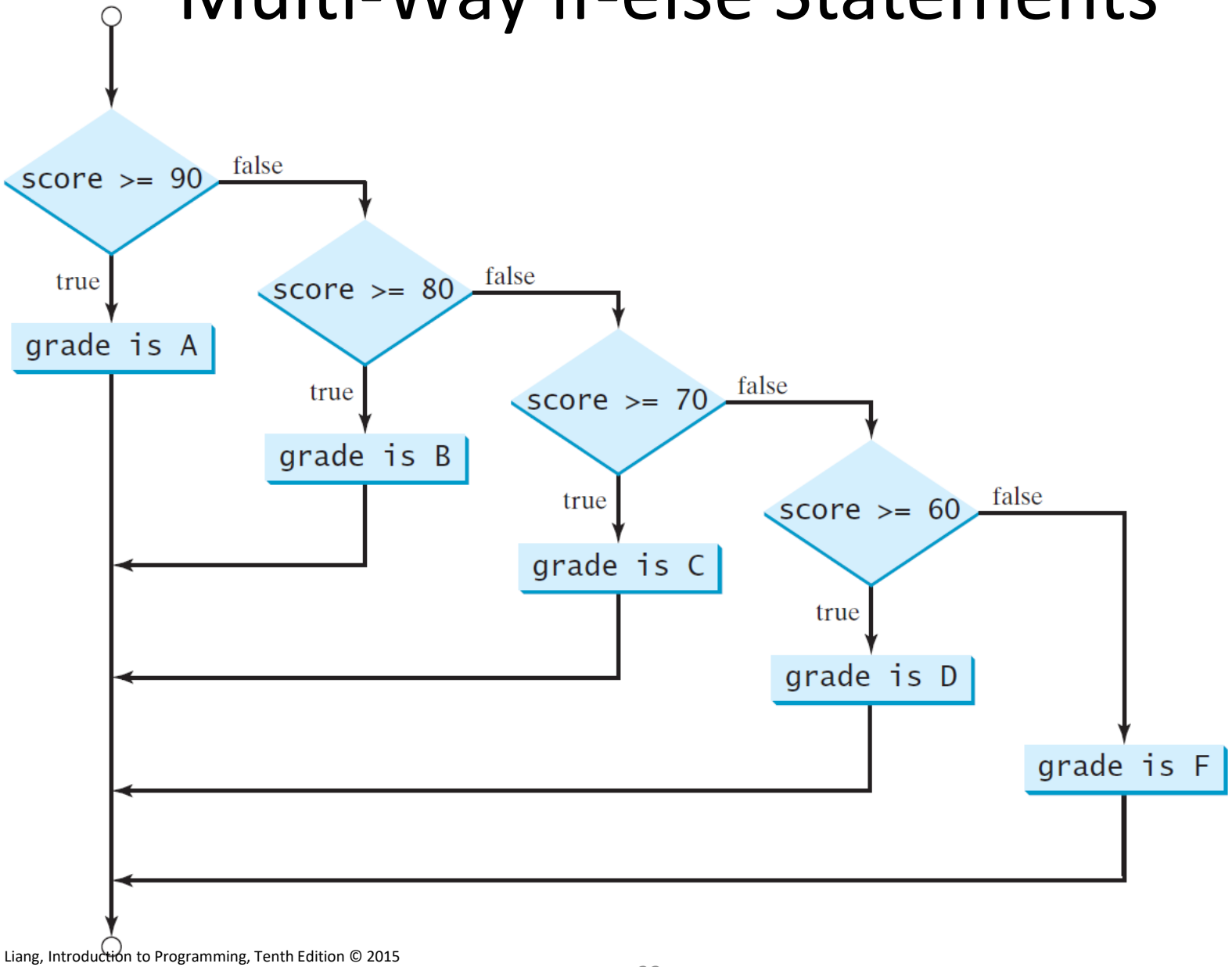
Equivalent

This is better

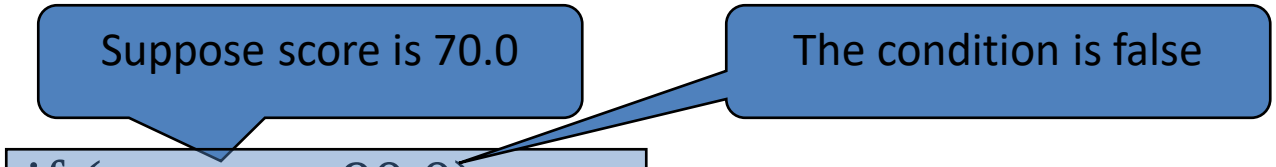
```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

(b)

Multi-Way if-else Statements

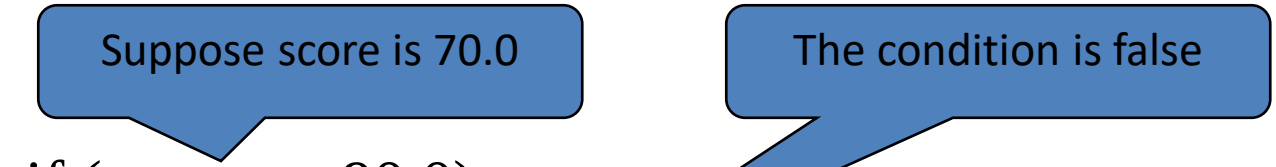


Trace if-else statement



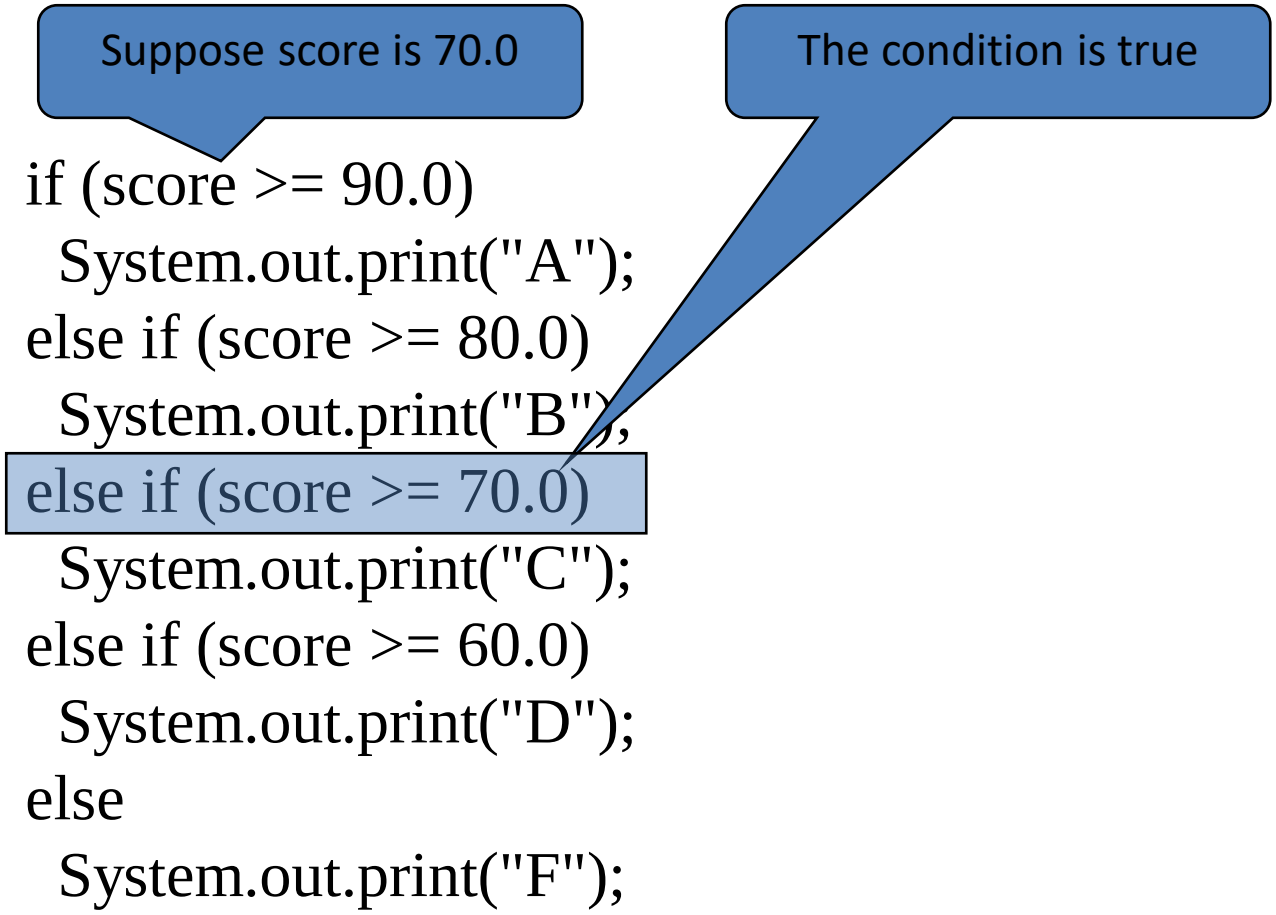

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

Trace if-else statement

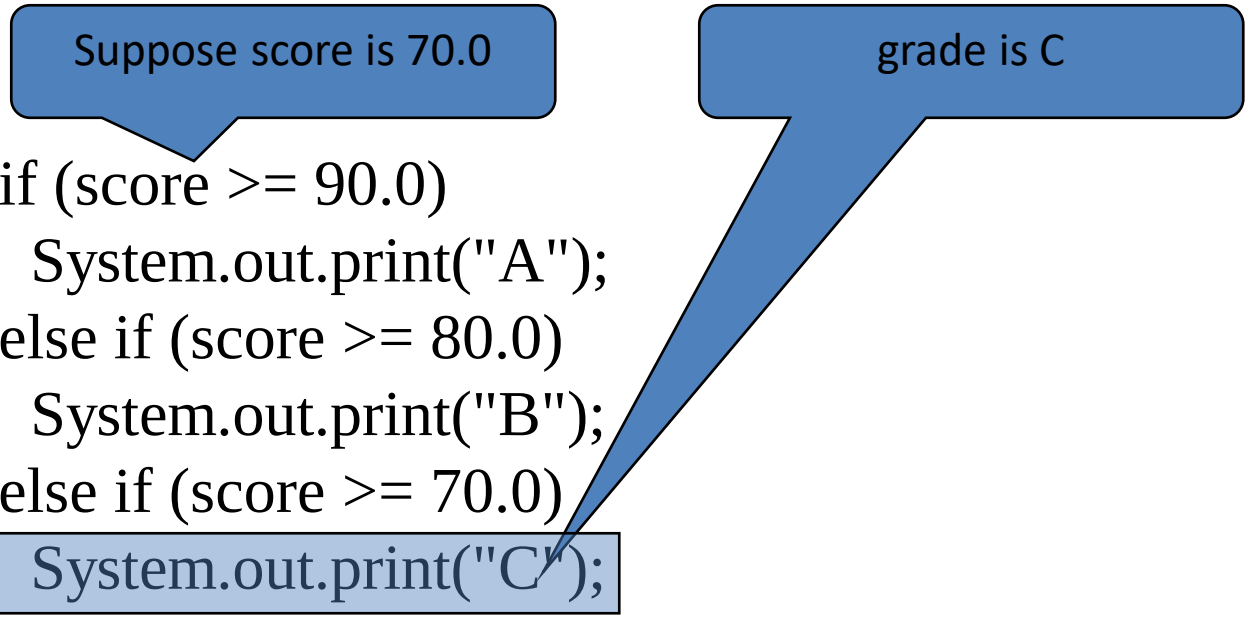



```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

Trace if-else statement



Trace if-else statement



animation

Trace if-else statement

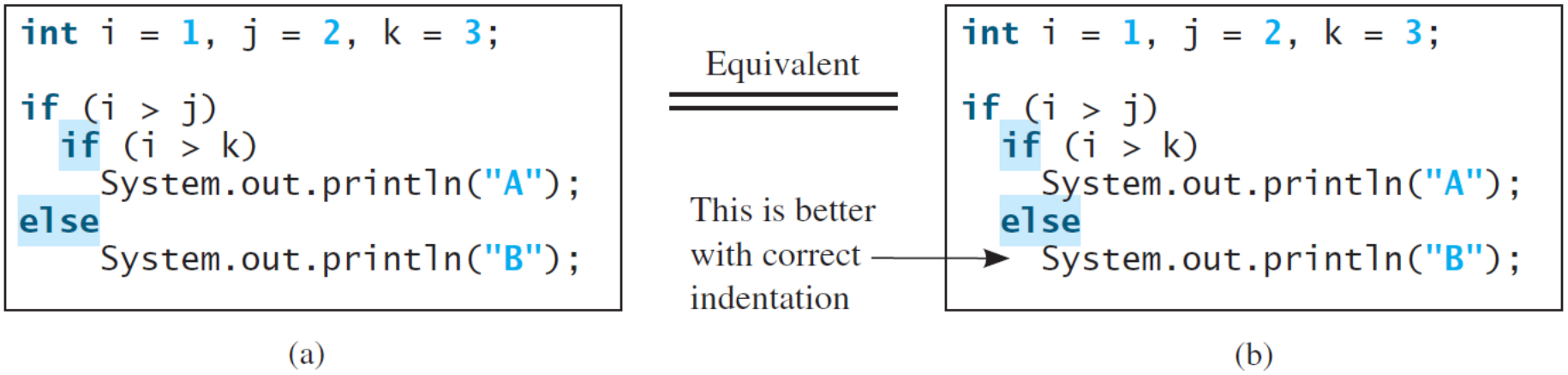
Suppose score is 70.0

Exit the if statement

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

Note

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



Note, cont.

Nothing is printed from the preceding statement. To force the else clause to match the first if clause, you must add a pair of braces:

```
int i = 1;
int j = 2;
int k = 3;
if (i > j) {
  if (i > k)
    System.out.println("A");
}
else
  System.out.println("B");
```

This statement prints B.

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 for the circle of radius " +
    radius + " is " + area);
}
```

Wrong

This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error.

This error often occurs when you use the next-line block style.

CAUTION

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

(a)

Equivalent

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

(b)

Note: The right style is preferred, you can user the binary negation operator (!) if your want to do something when a condition is false

Problem: A Math Learning Tool

This example creates a program to teach a first grade child how to learn subtractions. The program randomly generates two single-digit integers number1 and number2 with number1 >= number2 and displays a question such as "What is 9 – 2?" to the student. After the student types the answer, the program displays whether the answer is correct.

Problem: Body Mass Index

Body Mass Index (BMI) is a measure of health on weight. It can be calculated by taking your weight in kilograms and dividing by the square of your height in meters. The interpretation of BMI for people 16 years or older is as follows:

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

Logical Operators

Operator	Name	Description
!	not	logical negation
&&	and	logical conjunction
||	or	logical disjunction
^	exclusive or	logical exclusion

Truth Table for 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.

Truth Table for 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 (age > 18) and (weight <= 140) 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.

Truth Table for Operator ||

p_1	p_2	p_1 || p_2	Example (assume age = 24, weihgt = 140)
false	false	false	
false	true	true	(age > 34) || (weight <= 140) is true, because (age > 34) is false, but (weight <= 140) is true.
true	false	true	(age > 14) || (weight >= 150) is false, because (age > 14) is true.
true	true	true	

Truth Table for Operator ^

p_1	p_2	p_1 ^ p_2	Example (assume age = 24, weight = 140)
false	false	false	(age > 34) ^ (weight > 140) is true, because (age > 34) is false and (weight > 140) is false.
false	true	true	(age > 34) ^ (weight >= 140) is true, because (age > 34) is false but (weight >= 140) is true.
true	false	true	(age > 14) ^ (weight > 140) is true, because (age > 14) is true and (weight > 140) is false.
true	true	false	

Examples

Here is 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:

Examples

```
System.out.println("Is " + number + " divisible by 2 and 3? " +
  ((number % 2 == 0) && (number % 3 == 0)));

System.out.println("Is " + number + " divisible by 2 or 3? " +
  ((number % 2 == 0) || (number % 3 == 0)));

 System.out.println("Is " + number +
   " divisible by 2 or 3, but not both? " +
   ((number % 2 == 0) ^ (number % 3 == 0)));
```

Problem: Determining Leap Year?

This program first prompts the user to enter a year as an int value and checks if it is a leap year.

A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.

```
(year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)
```

Problem: Lottery

Write a program that randomly generates a lottery of a two-digit number, prompts the user to enter a two-digit number, and determines whether the user wins according to the following rule:

- If the user input matches the lottery in exact order, the award is $10,000.
- If the user input matches the lottery, the award is $3,000.
- If one digit in the user input matches a digit in the lottery, the award is $1,000.

Switch Statement: Motivation

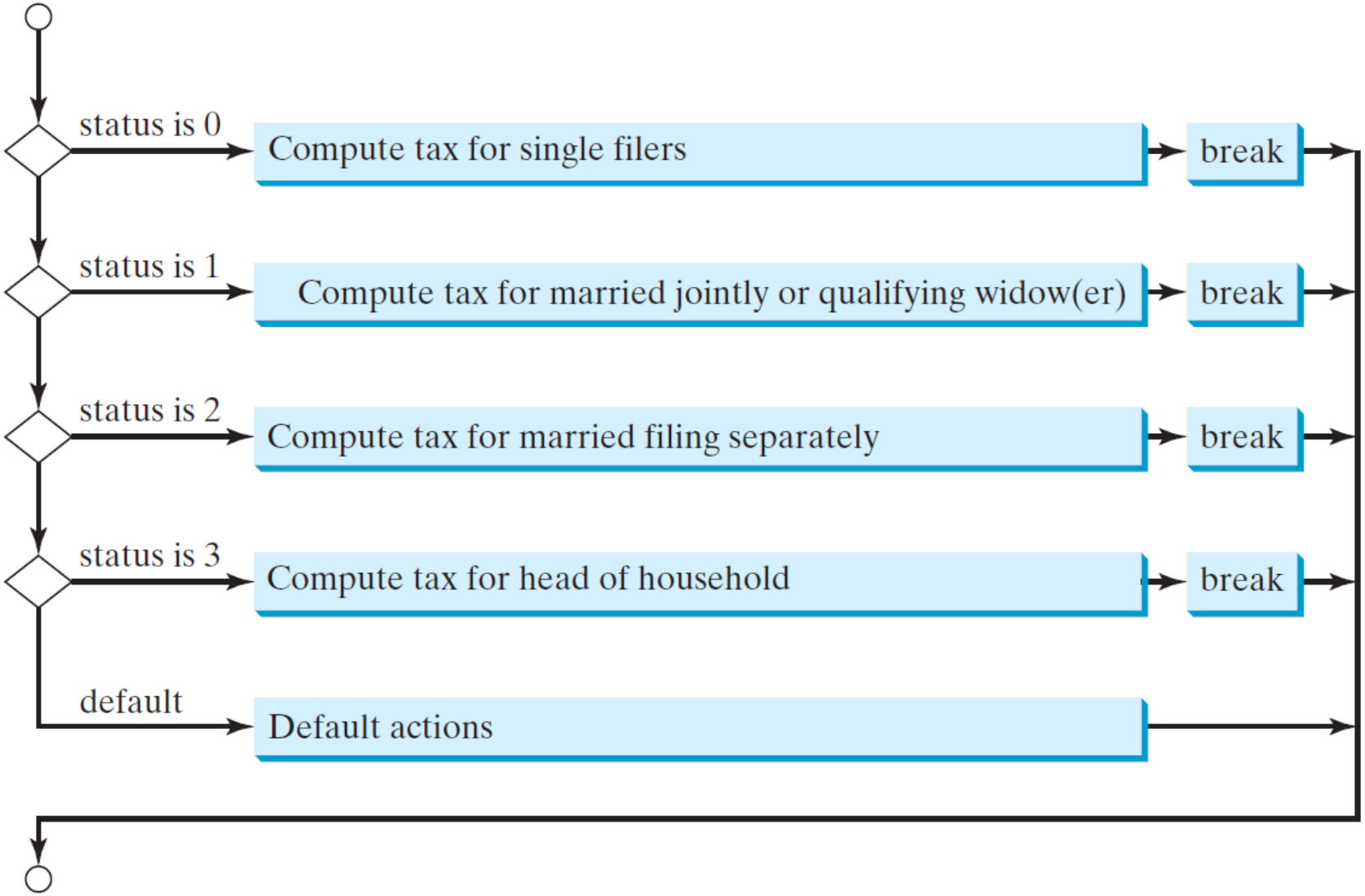
- Sometime, things can get messy if there are many alternative cases. Java has a switch statement to help with complicate selections
- For example:
  - The US federal personal income tax is calculated based on the filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates for 2009 are shown below.

Marginal Tax Rate	*Single*	*Married Filing Jointly or Qualifying Widow(er)*	*Married Filing Separately*	*Head of Household*
10%	$0 – $8,350	$0 – $16,700	$0 – $8,350	$0 – $11,950
15%	$8,351 – $33,950	$16,701 – $67,900	$8,351 – $33,950	$11,951 – $45,500
25%	$33,951 – $82,250	$67,901 – $137,050	$33,951 – $68,525	$45,501 – $117,450
28%	$82,251 – $171,550	$137,051 – $208,850	$68,526 – $104,425	$117,451 – $190,200
33%	$171,551 – $372,950	$208,851 – $372,950	$104,426 – $186,475	$190,201 – $372,950
35%	$372,951+	$372,951+	$186,476+	$372,951+

`switch` Statements

```
switch (status) {
  case 0:  compute taxes for single filers;
           break;
  case 1:  compute taxes for married file jointly;
           break;
  case 2:  compute taxes for married file separately;
           break;
  case 3:  compute taxes for head of household;
           break;
  default: System.out.println("Errors: invalid status");
           System.exit(1);
}
```

`switch` Statement Flow Chart



`switch` Statement Rules

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;
}
```

`switch` Statement Rules

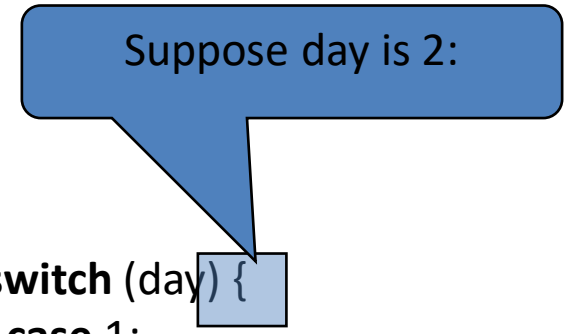
The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement. If the break statement is not present, the next case statement will be executed.

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

```
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;
}
```

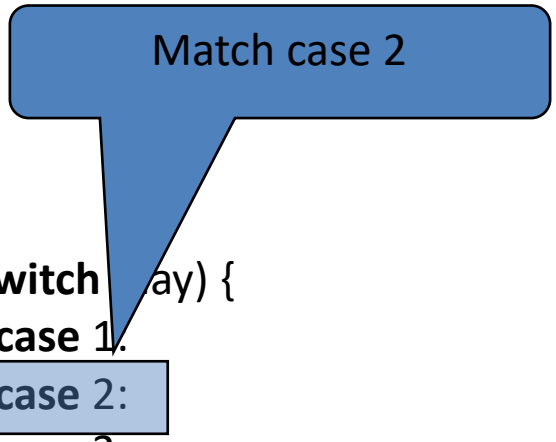
When the value in a **case** statement matches the value of the **switch-expression**, the statements *starting from this case* are executed until either a **break** statement or the end of the **switch** statement is reached.

Trace switch statement



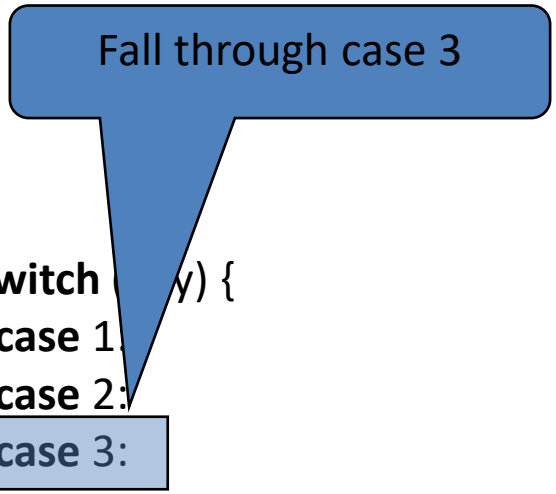

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

Trace switch statement




```
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");
}
```

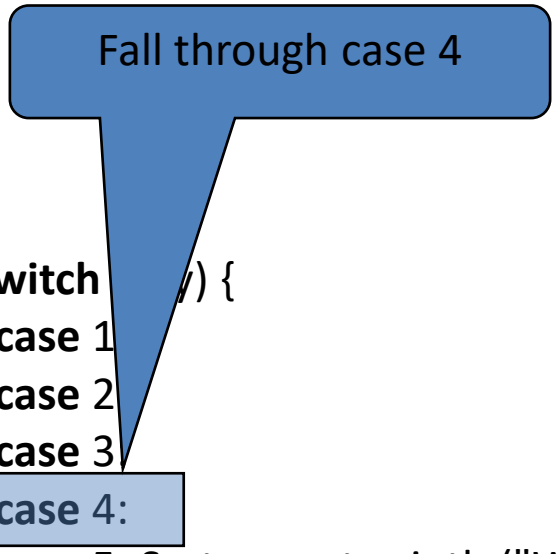
Trace switch statement




```
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");
}
```

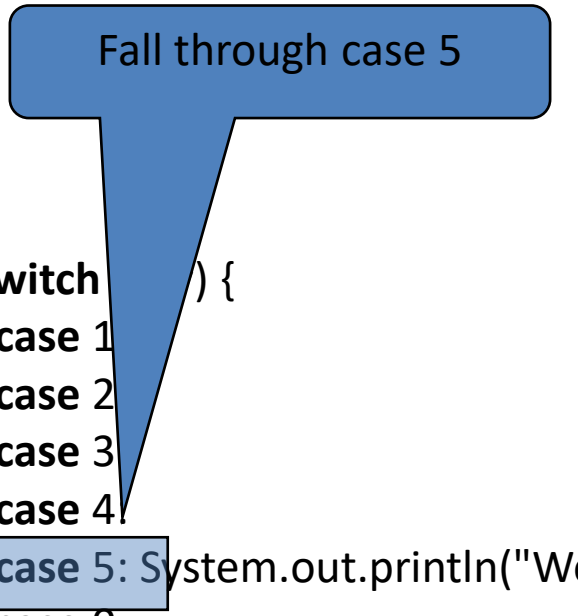
animation

Trace switch statement



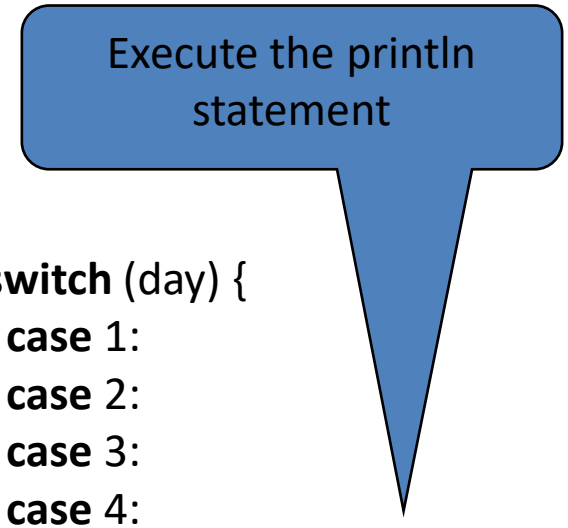

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

Trace switch statement



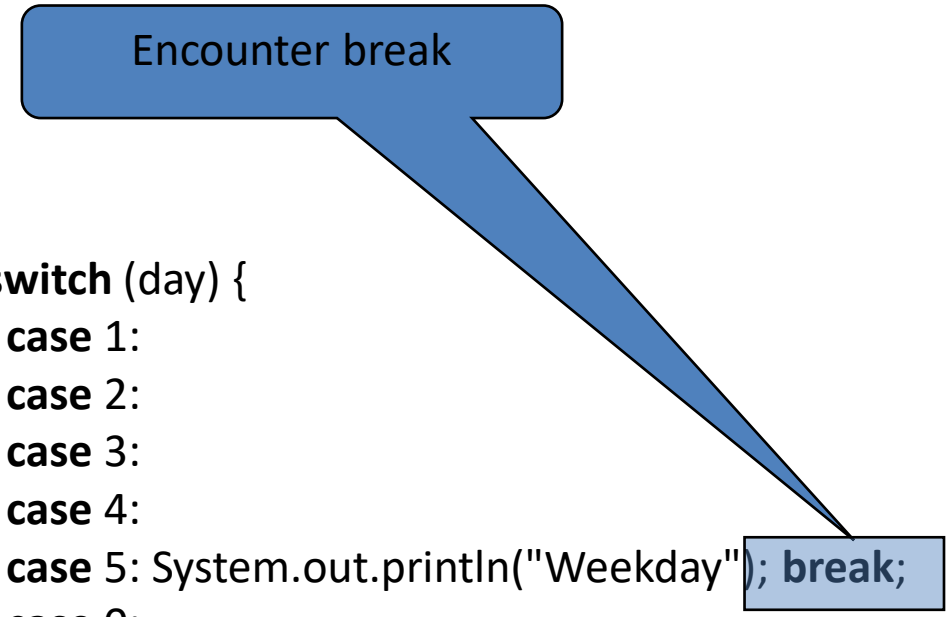

```
switch (      ) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

Trace switch statement



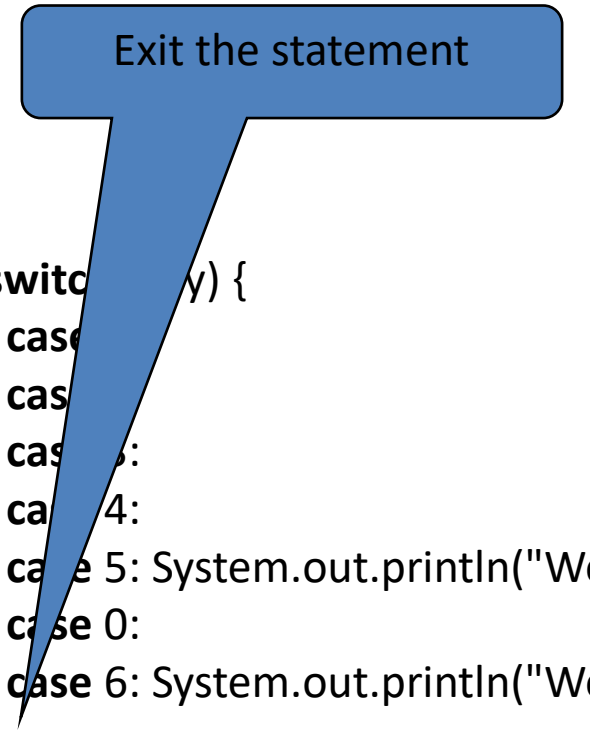

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

Trace switch statement




```
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");
}
```

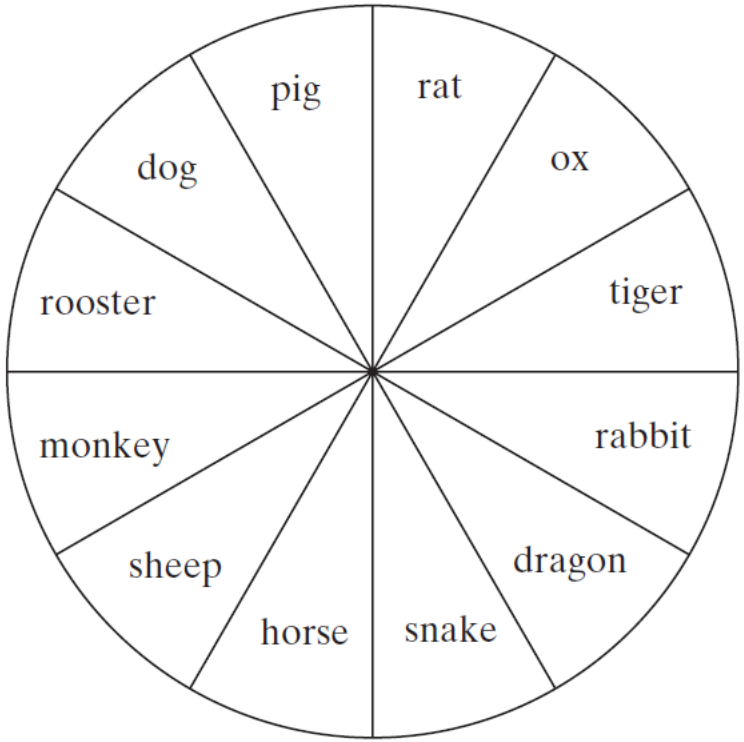
Trace switch statement




```
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: Chinese Zodiac

Write a program that prompts the user to enter a year and displays the animal for the year.



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 Expressions

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

is equivalent to

```
y = (x > 0) ? 1 : -1;
```

(boolean-expression) ? expression1 : expression2

Ternary operator
Binary operator
Unary operator

Conditional Operator

```
if (num % 2 == 0)
  System.out.println(num + "is even");
else
  System.out.println(num + "is odd");
```

```
System.out.println(
  (num % 2 == 0)? num + "is even" :
  num + "is odd");
```

Conditional Operator, cont.

```
boolean-expression ? exp1 : exp2
```

Operator Precedence

- **var++, var--**
- **+, - (Unary plus and minus), ++var,--var**
- **(type) Casting**
- **! (Not)**
- ***, /, % (Multiplication, division, and remainder)**
- **+, - (Binary addition and subtraction)**
- **<, <=, >, >= (Relational operators)**
- **==, !=; (Equality)**
- **^ (Exclusive OR)**
- **&& (Conditional AND) Short-circuit AND**
- **|| (Conditional OR) Short-circuit OR**
- **=, +=, -=, *=, /=, %= (Assignment operator)**

Operator Precedence and Associativity

The expression in the parentheses is evaluated first. (Parentheses can be nested, in which case the expression in the inner parentheses is executed first.) When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule.

If operators with the same precedence are next to each other, their associativity determines the order of evaluation. All binary operators except assignment operators are left-associative.

Operator Associativity

When two operators with the same precedence are evaluated, the *associativity* of the operators determines the order of evaluation. All binary operators except assignment operators are *left-associative*.

a – b + c – d is equivalent to ((a – b) + c) – d

Assignment operators are *right-associative*. Therefore, the expression

a = b += c = 5 is equivalent to a = (b += (c = 5))

Example

Applying the operator precedence and associativity rule, the expression 3 + 4 * 4 > 5 * (4 + 3) - 1 is evaluated as follows: