FAKULTAS
ILMU
KOMPUTER

Elementary Programming

Java Basics and Selections

Dasar – Dasar Pemrograman 2

Dinial Utami Nurul Qomariah

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Credits

- Liang, Introduction to Java Programming, 11th Edition, Ch. 2
- Downey & Mayfield, Think Java: How to Think Like a Computer Scientist, Ch. 2
- Slide Kuliah Dasar-dasar Pemrograman 2 Semester Genap 2021/2022

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Outline

- Basic Java Programming
- Reading Input from console
- Tipe Data
- Operator
- Numeric conversion
- Selection (Percabangan)

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Basic Java Programming

Identifiers

- Identify variable, method, class.
- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- An identifier cannot be a reserved word.
  - https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html
- An identifier cannot be true, false, or null.
- An identifier can be of any length.

Yang error?

_radius radius1 1radius $radius class radius 1

Identifiers

- Identify variable, method, class.
- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- An identifier cannot be a reserved word.
  - https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html
- An identifier cannot be true, false, or null.
- An identifier can be of any length.

Yang error?

_radius radius1 $radius

Declaring Variables

```
int x;              // Declare x to be an
                    // integer variable;
double radius;      // Declare radius to
                    // be a double variable;
char a;             // Declare a to be a
                    // character variable;
```

Assignment Statements

```
int x;            // Declare x to be an
                  // integer variable;
x = 1;             // Assign 1 to x;
double radius; // Declare radius to
                  // be a double variable;
radius = 1.0;    // Assign 1.0 to radius
char a;           // Declare a to be a
                  // character variable;
a = 'A';           // Assign 'A' to a;
```

Declaring and Initializing in One Step

```
int x = 1;

double d = 1.4;
```

Constant Variable with "final" keyword

- Means the value is only assigned once in a lifetime.
- Use a variable PI as a constant of 3.14. Then, compile & run.
  - `final double PI = 3.14;`
  - `area = radius * radius * PI;`
- Try to reset the value of PI after compute the area.
  - `PI = 3.141;`
- What happens?

Naming Conventions

- Choose meaningful and descriptive names.
- Variables and method names:
  - Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name. For example, the variables `radius` and `area`, and the method `computeArea`.

Naming Conventions

- Class names:
  - Capitalize the first letter of each word in the name. For example, the class name `ComputeArea`.

- Constants:
  - Capitalize all letters in constants, and use underscores to connect words. For example, the constant `PI and` MAX_VALUE

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Reading Input From Console

Reading Input from the Console

- Create a Scanner object.
  - `Scanner input = new Scanner(System.in);`
- Use the method nextDouble() to obtain to a double value. For example,
  - ```
    System.out.print("Enter a double value: ");
    Scanner input = new Scanner(System.in);
    double d = input.nextDouble();
    ```
    ```

Input Scanner

Methode	Deskripsi
nextByte()	Membaca nilai integer pada tipe byte
nextShort()	Membaca nilai integer pada tipe short
nextInt()	Membaca nilai integer pada tipe int
nextLong()	Membaca nilai integer pada tipe long
nextFloat()	Membaca nilai integer pada tipe float
nextDouble()	Membaca nilai integer pada tipe double

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args){
        int radius;
        int area;
        Scanner input = new Scanner(System.in);
        System.out.println("Nilai Radius =" ) ;
        radius = input.nextInt();

        area = radius * radius * 3;
        System.out.println("Nilai area =" + area) ;
    }
}
```

Tipe Data

Numerical Data Types

Name	Range	Storage Size
`byte`	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed
`short`	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
`int`	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
`long`	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
`float`	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
`double`	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

Number Literals

- A compilation error would occur if the literal were too large for the variable to hold.
  - The statement **byte b = 1000** would cause a compilation error (max byte is 127)
- By default, a **floating-point** literal is treated as a **double** type value.
  - Append the letter f or F to make a number a float
  - Append the letter d or D to make a number a double (optional)
- By default, an **integer** literal is treated as an **int** type value.
  - Append the letter l or L to make a number a long

double vs. float

The double type values are more accurate than the float type values. For example,

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

displays `1.0 / 3.0 is 0.3333333333333333`

16 digits

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays `1.0F / 3.0F is 0.33333334`

7 digits

Tipe Data Reference

Tipe data reference digunakan untuk menampung referensi Object

- Integer, Long, Float, Double, String, Array dll.

Primitif	Reference
Diawali huruf kecil → int, long, float, double, char	Diawali huruf capital → Integer, Long, Float, Double, String
Kategori keyword	Bukan keyword
Nilai default tergantung pada tipe data	Nilai default = null
Menampung nilai/ value	Menampung referensi object
Saat declare tanpa inisialisasi sudah dialokasikan ke memory	Saat declare tidak dialokasikan ke memory. baru ketika diinisialisasi dialokasikan ke memory

Operator

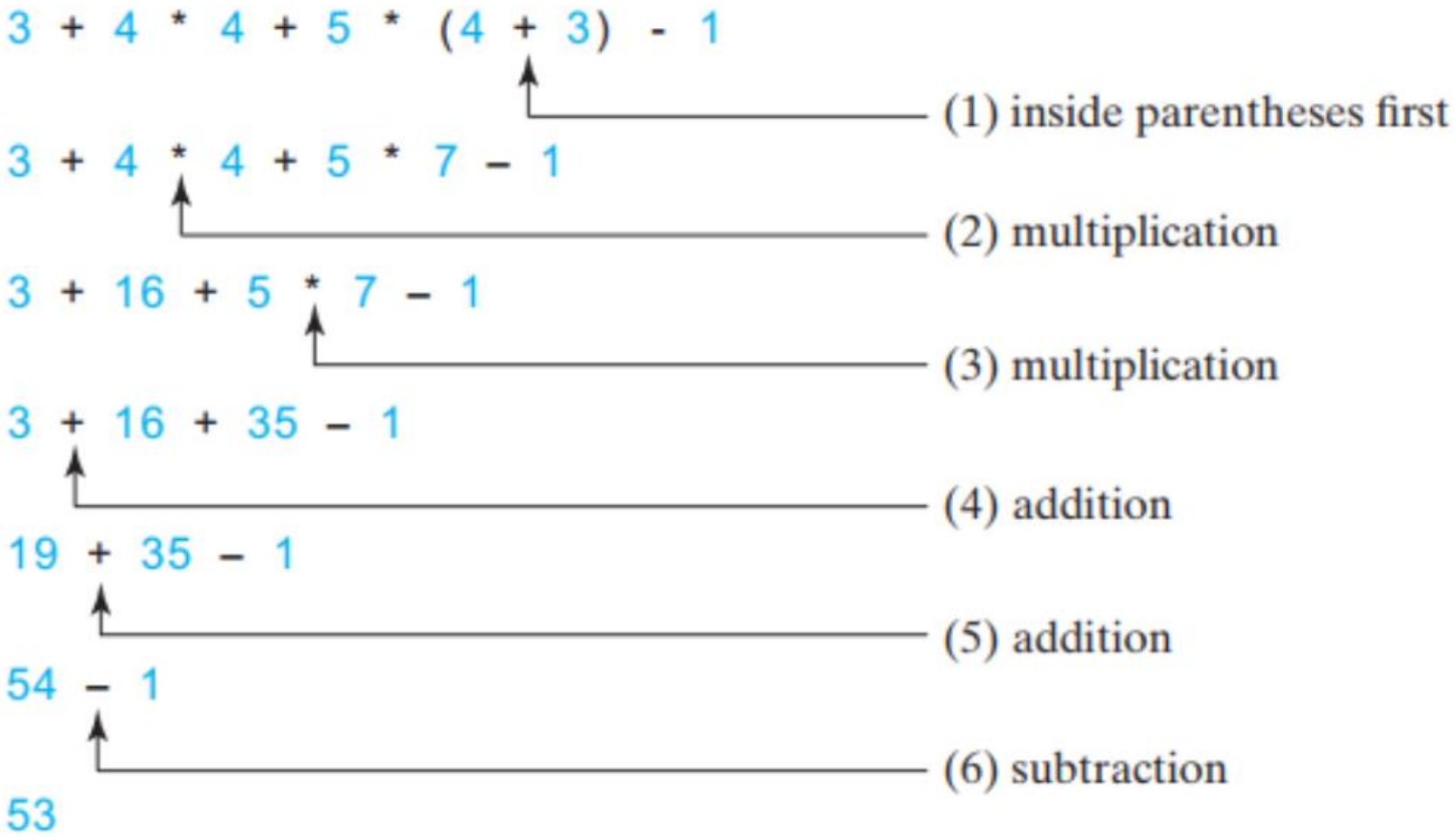
Numeric Operators

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Precedence Operators

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Precedence Operators



Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /= 8`	`i = i / 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`

area = radius * radius * 3.14;

Solution:

```
area = radius;
area *= radius;
area *= 3.14;
```

Increment and Decrement Operators

Operator	*Name*	*Description*	*Example (assume i = 1)*
`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;` `// j is 2, i is 2`
`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;` `// j is 1, i is 2`
`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;` `// j is 0, i is 0`
`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;` `// j is 1, i is 0`

What is the output?

```
int i = 2;
System.out.println(7 - ++i * 1.5 / 3 + 6 % 4 - ++i);  3.5
System.out.println(7 - i++ * 1.5 / 3 + 6 % 4 - ++i);  1.0
```

Increment and Decrement Operators

```
int i = 10;
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i;
i = i + 1;
```

```
int i = 10;
int newNum = 10 * (++i);
```

Same effect as

```
i = i + 1;
int newNum = 10 * i;
```

Numeric Conversion

Numeric Type Conversion

Consider the following statements:

```
byte i = 100;
long k = i * 3 + 4;
double d = i * 3 + k / 2;
```

Try this... And what happens?

```
int x = d + 1;
```

Numeric Type Conversion

Widening Casting (Otomatis) → byte – short – int – long – float – double

Widening Casting (Manual) → double – float – long – int – short – byte

Implicit casting

`double d = 3;` (type widening)

Explicit casting

`int i = (int)3.0;` (type narrowing)

`int i = (int)3.9;` (Fraction part is truncated)

Casting in an Augmented Expression

In Java, an augmented expression of the form **x1 op= x2** is implemented as **x1 = (T)(x1 op x2)**, where **T** is the type for **x1**. Therefore, the following code is correct.

```
int sum = 0;
sum += 4.5; // sum becomes 4 after this statement
```

sum += 4.5 is equivalent to **sum = (int)(sum + 4.5)**.

Selections

Motivations

If you assigned a negative value for radius in Circle area
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**?**

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`

Example: A Simple Math Learning Tool

- This example creates a program to let a first grader practice additions.
- The program randomly 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.

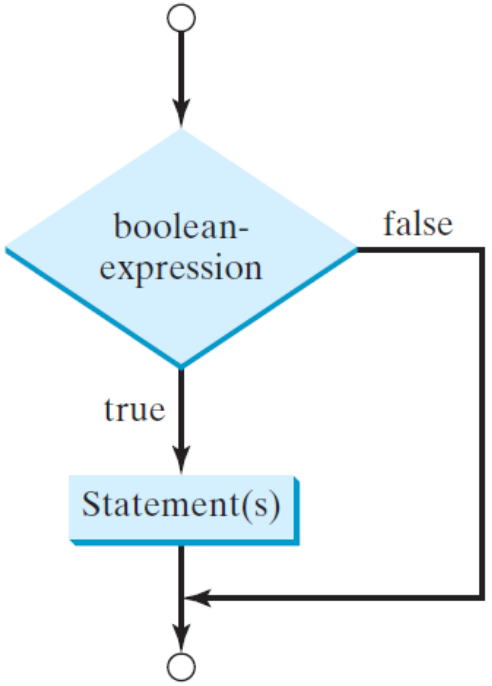
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

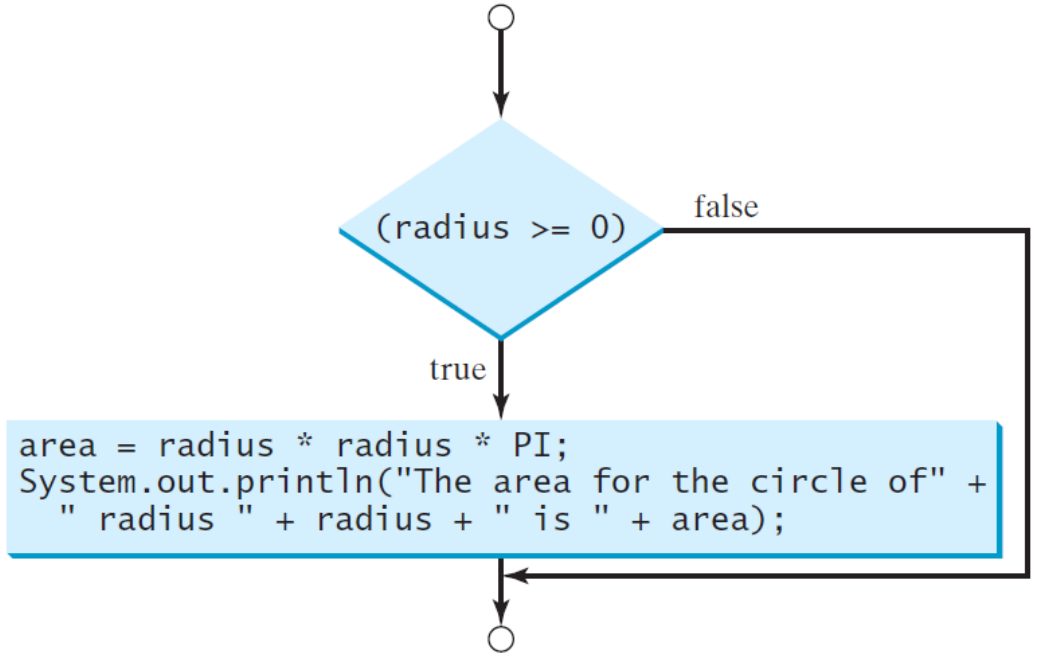
Percabangan If Else Statements

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 for One-way if statements

```
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)

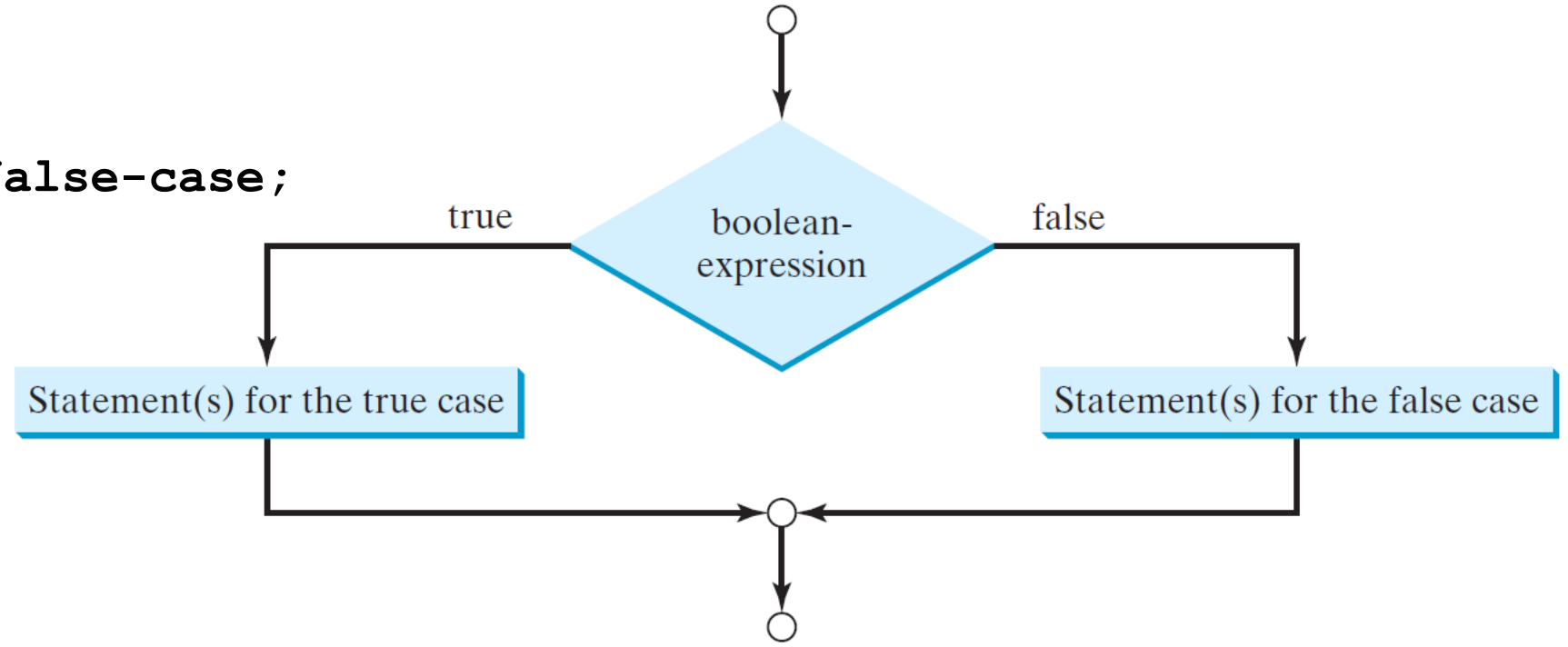
Simple if Demo

Write a program that prompts the user to enter an integer. If the number is a multiple of 5, print HiFive. If the number is divisible by 2, print HiEven.

```
import java.util.Scanner;

public class SimpleIfDemo {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter an integer: ");
    int number = input.nextInt();

    if (number % 5 == 0)
      System.out.println("HiFive");

    if (number % 2 == 0)
      System.out.println("HiEven");
  }
}
```

Two-way `if` Statements (if - else)

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



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

Cara Penulisan yang dapat digunakan

```
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)

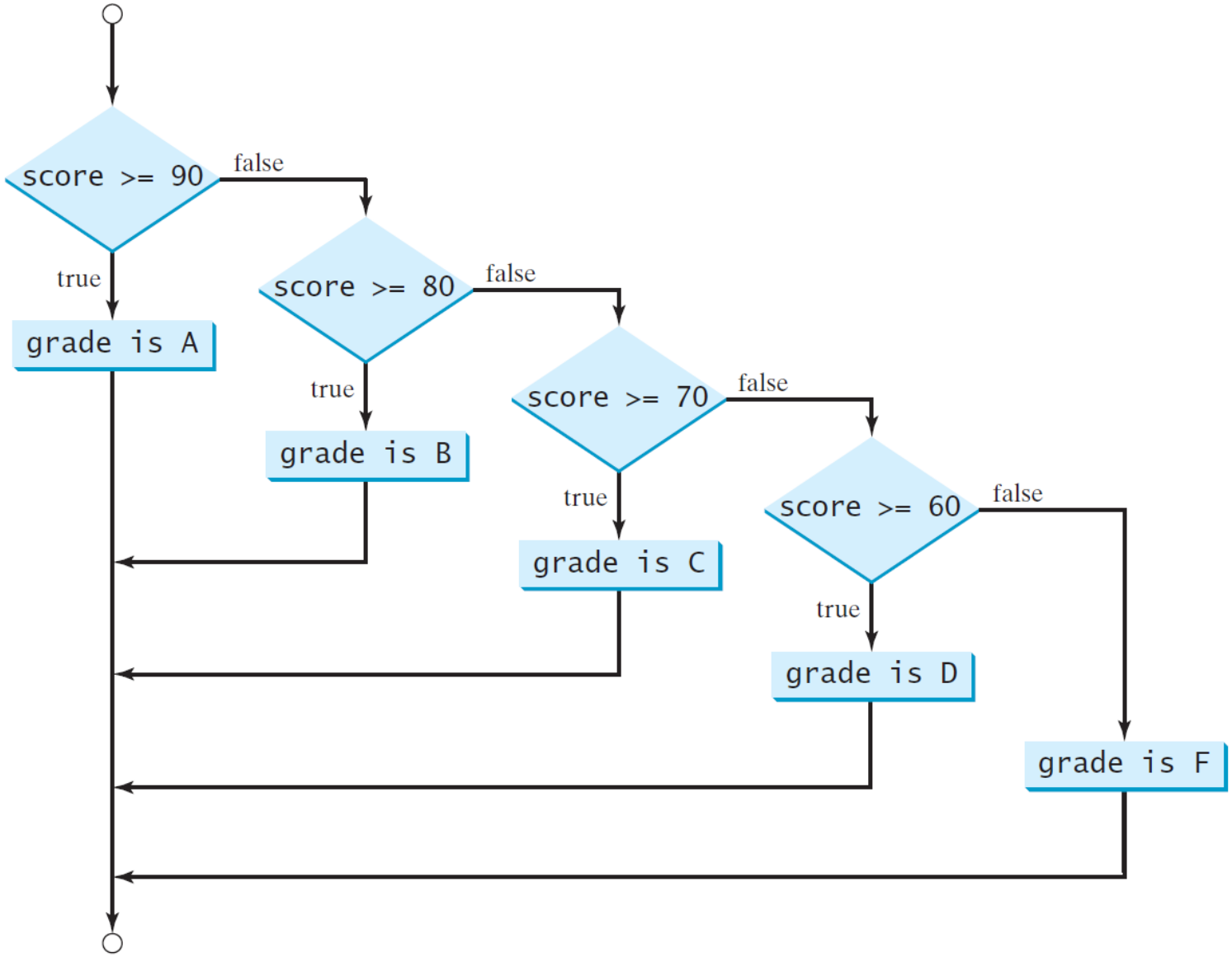
Equivalent

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)

Multiple Alternative if Statements



Trace if-else statement

Suppose score is 70.0

The condition is false

```
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

Suppose score is 70.0

The condition is false

```
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

Suppose score is 70.0

The condition is true

```
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

Suppose score is 70.0

grade is C

```
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

Suppose score is 70.0

```
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");
```

Exit the if statement

Note

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

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

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

(a)

Equivalent

This is better with correct indentation

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

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

(b)

Note Next

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);
}
```

- not a compilation or a runtime error, it is a logic error.

Note

Make it Shorter

```
if (number % 2 == 0)
   even = true;
else
   even = false;
```

(a)

Equivalent

```
boolean even
   = number % 2 == 0;
```

(b)

Logical Operators

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

The & and | Operators

- A short circuit operator (&&, ||) is an operator that doesn't necessarily check all of its operands.
- Try this…

```
int x = 1;
if((x > 1) & (x++ < 10))
System.out.println("do something");
// what is this output?
System.out.println("x = " + x);
// Try to change & with &&
```

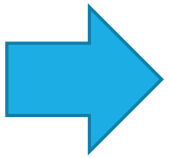
Complex Problem: Computing Taxes

- 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+

Complex Problem: Computing Taxes next…

```
if (status == 0) {
  // Compute tax for single filers
}
else if (status == 1) {
  // Compute tax for married file jointly
  // or qualifying widow(er)
}
else if (status == 2) {
  // Compute tax for married file separately
}
else if (status == 3) {
  // Compute tax for head of household
}
else {
  // Display wrong status
}
```



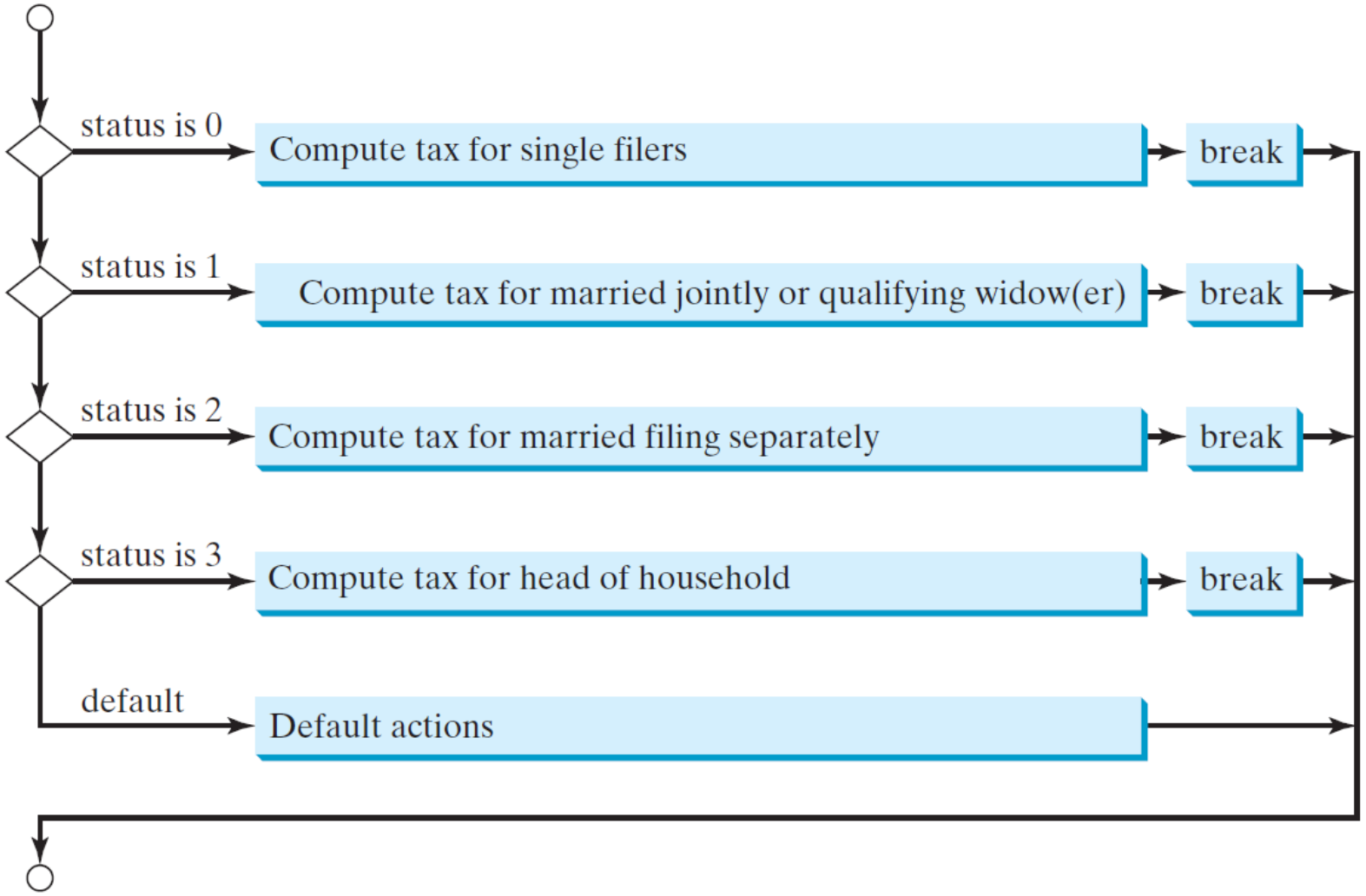
So Complicated Right?

Percabangan Switch Case Statements

`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` Statements Flow Chart



`switch` Statement Rules

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

statement are executed when the value in the case statement matches the value of the switchexpression.

`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.

```
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;
}
```

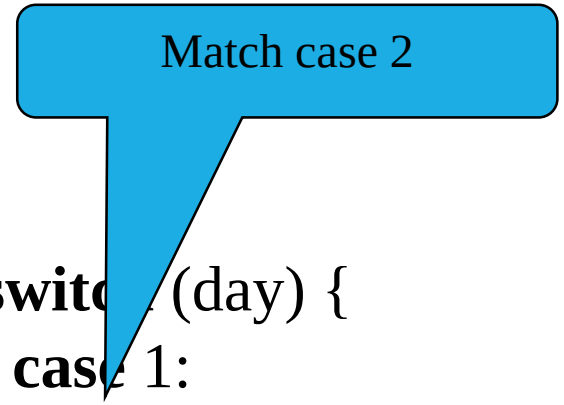
The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.

Trace Switch statement Example: Weekday or Weekend

Suppose day is 2:

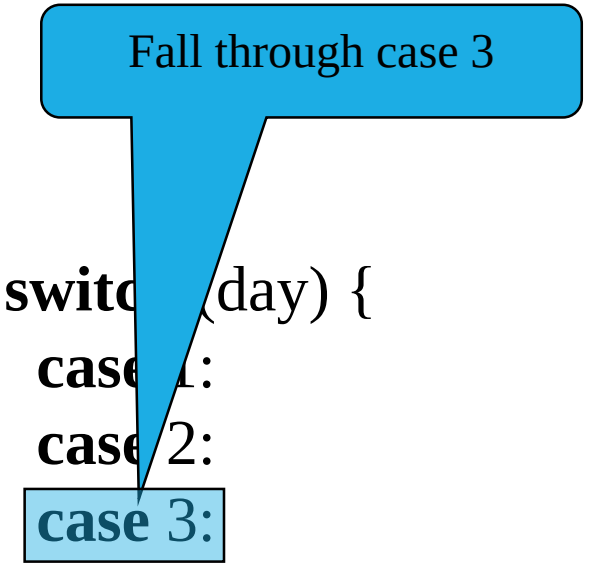
```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

Example: Weekday or Weekend



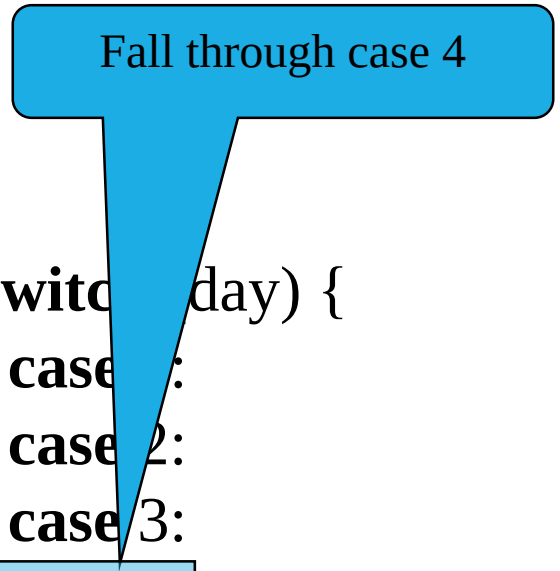

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

Example: Weekday or Weekend




```
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");
}
```

Example: Weekday or Weekend

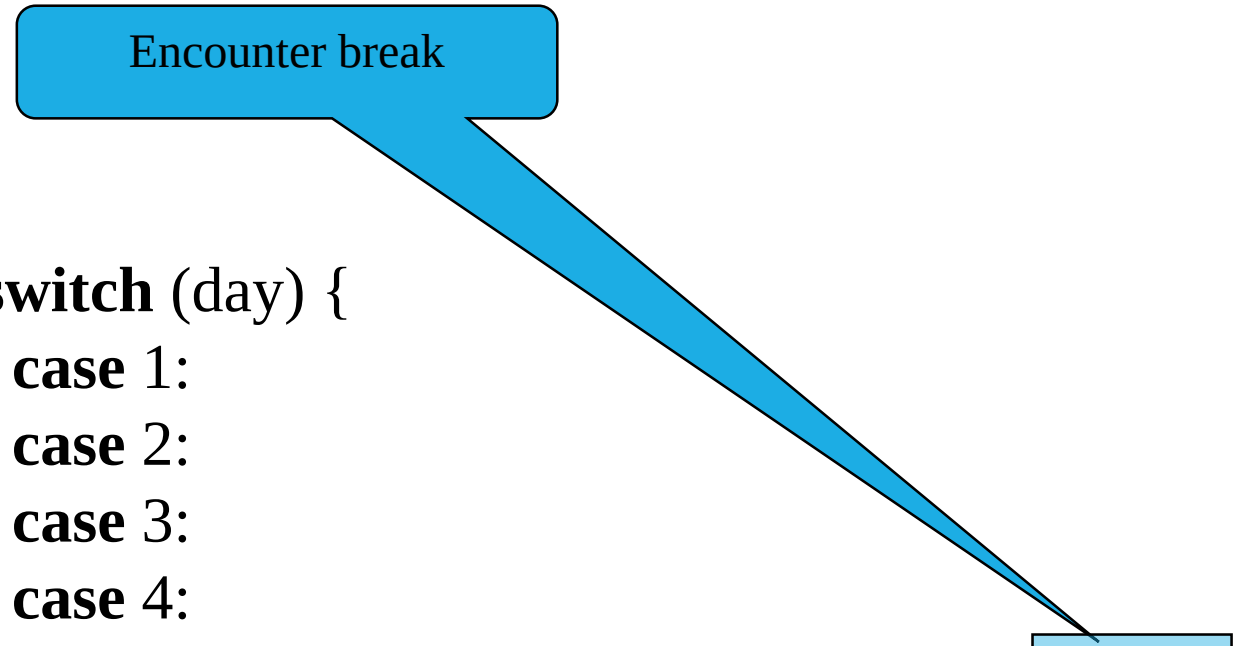



```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

Example: Weekday or Weekend

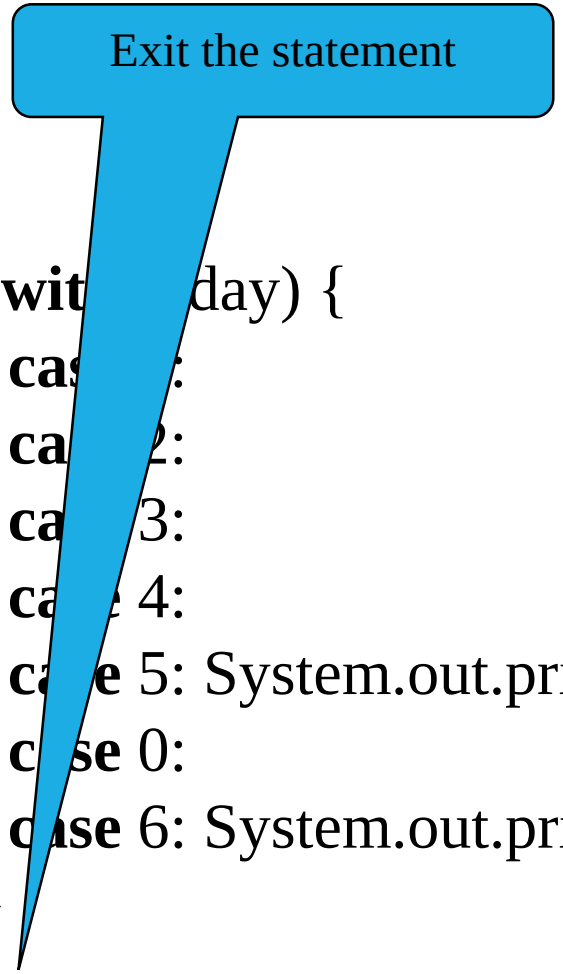

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 0:
  case 6: System.out.println("Weekend");
}
```

Example: Weekday or Weekend




```
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");
}
```

Example: Weekday or Weekend




```
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");
}
```

Conditional Expressions

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

=

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

(boolean-expression) ? expression1 : expression2

The symbols ? and : appearing together as a *conditional operator* or *ternary operator*.

Use conditional expressions to have equivalent result with this:

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