Chapter 4: Selections

Motivations

If you assigned a negative value for radius in Listing 2.1, ComputeArea.java, 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?

Objectives

- To declare boolean type and write Boolean expressions using comparison operators (§3.2).
- To program AdditionQuiz using Boolean expressions (§3.3).
- To implement selection control using one-way if statements (§3.4)
- To program the GuessBirthday game using one-way if statements (§3.5).
- To implement selection control using two-way if statements (§3.6).
- To implement selection control using nested if statements (§3.7).
- To avoid common errors in if statements (§3.8).
- To program using selection statements for a variety of examples (BMI, ComputeTax, SubtractionQuiz) (§3.9-3.11).
- To generate random numbers using the Math.random() method (§3.9).
- To combine conditions using logical operators (&&, ||, and !) (§3.12).
- To program using selection statements with combined conditions (LeapYear, Lottery) (§§3.13-3.14).
- To implement selection control using switch statements (§3.15).
- To write expressions using the conditional operator (§3.16).
- To format output using the System.out.printf method and to format strings using the String.format method (§3.17).
- To examine the rules governing operator precedence and associativity (§3.18).
- (GUI) To get user confirmation using confirmation dialogs (§3.19).

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);
```

Comparison Operators

Operator	*Name*
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

Problem: 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.

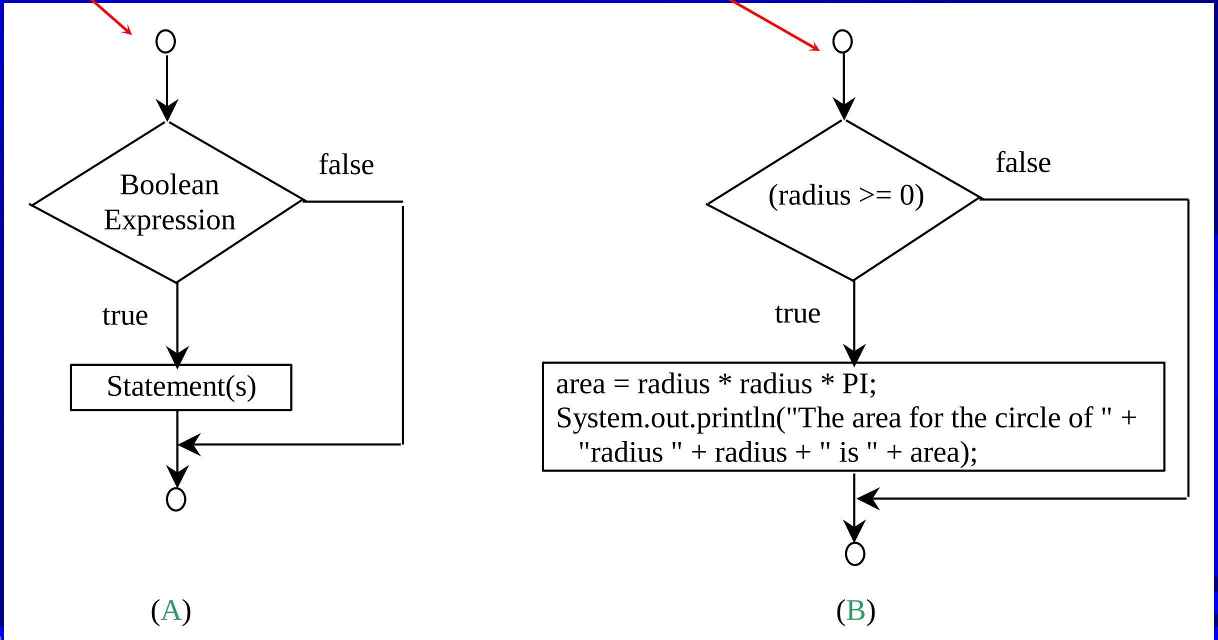
AdditionQuiz

Run

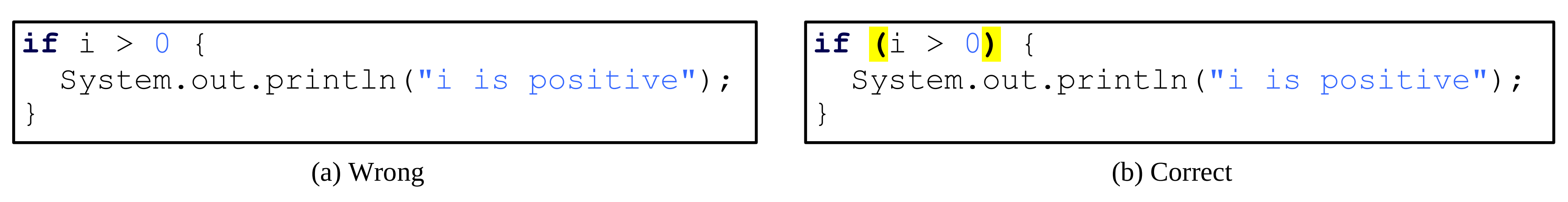
One-way `if` Statements

```
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);
}
```


Boolean
Expression
false
true
Statement(s)
(A)
(radius >= 0)
false
true
area = radius * radius * PI;
System.out.println("The area for the circle of " +
"radius " + radius + " is " + area);
(B)

Note

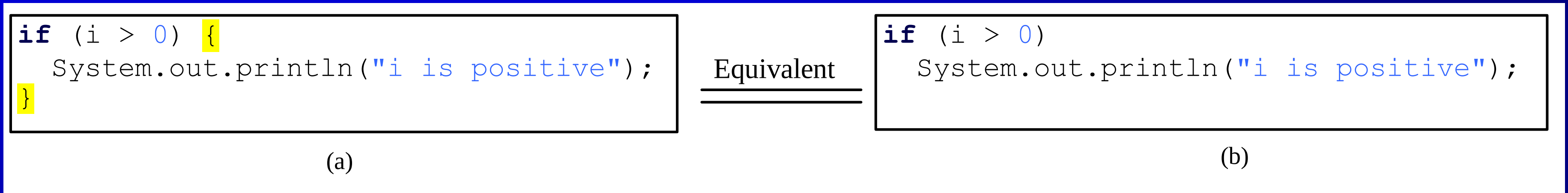


```
if i > 0 {
  System.out.println("i is positive");
}
```

(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)

Simple if Demo

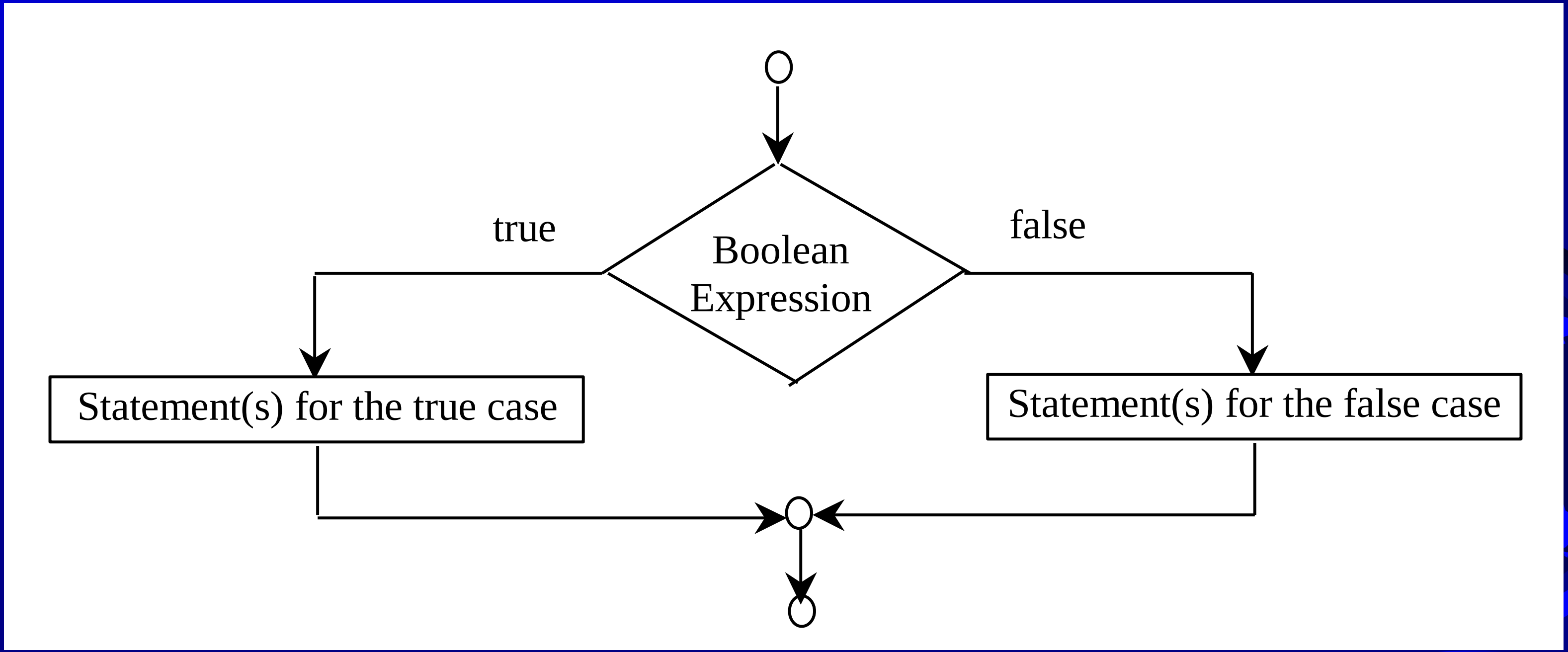
Write a program that prompts the user to enter an integer. If the number is a multiple of 5, print HiFive. If the number is divisible by 2, print HiEven.

SimpleIfDemo

Run

The Two-way `if` Statement

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


true
Boolean Expression
false
Statement(s) for the true case
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

```
if (score >= 90.0)
  grade = 'A';
else
  if (score >= 80.0)
    grade = 'B';
  else
    if (score >= 70.0)
      grade = 'C';
    else
      if (score >= 60.0)
        grade = 'D';
      else
        grade = 'F';
```

Equivalent

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

animation

Trace if-else statement

Suppose score is 70.0

The condition is false

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

animation

Trace if-else statement

Suppose score is 70.0

The condition is false

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

animation

Trace if-else statement

Suppose score is 70.0

The condition is true

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

animation

Trace if-else statement

Suppose score is 70.0

grade is C

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

animation

Trace if-else statement

Suppose score is 70.0

Exit the if statement

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Note

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

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

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

(a)

Equivalent

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

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

(b)

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.

TIP

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

(a)

Equivalent

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

(b)

CAUTION

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

(a)

Equivalent

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

(b)

Problem: An Improved Math Learning Tool

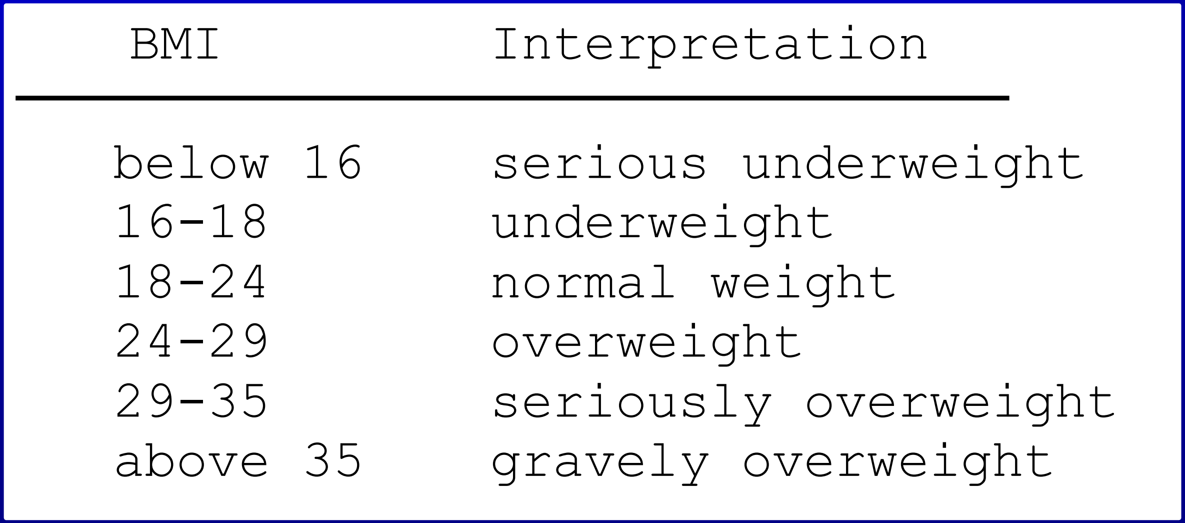
Create a program to teach a first grade child how to learn subtractions. The program randomly generates two single-digit integers number1 and number2 with number1 > number2 and displays a question such as “What is 9 – 2?” to the student. After the student types the answer in the input dialog box, the program displays a message dialog box to indicate whether the answer is correct.

SubtractionQuiz

Run

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
below 16	serious underweight
16-18	underweight
18-24	normal weight
24-29	overweight
29-35	seriously overweight
above 35	gravely overweight

ComputeBMI

Run

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 Qualified 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	$68,525 – $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+

Problem: Computing Taxes, cont.

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

ComputeTax

Run

Logical Operators

Operator	*Name*
!	not
&&	and
||	or
^	exclusive or

Truth Table for Operator !

p	!p	Example (assume age = 24, gender = 'M')
true	false	!(age > 18) is false, because (age > 18) is true.
false	true	!(gender != 'F') is true, because (grade != 'F') is false.

Truth Table for Operator &&

p1	p2	p1 && p2	Example (assume age = 24, gender = 'F')
false	false	false	(age > 18) && (gender == 'F') is true, because (age > 18) and (gender == 'F') are both true.
false	true	false	
true	false	false	(age > 18) && (gender != 'F') is false, because (gender != 'F') is false.
true	true	true	

Truth Table for Operator ||

p1	p2	p1 || p2	Example (assume age = 24, gender = 'F')
false	false	false	(age > 34) || (gender == 'F') is true, because (gender == 'F') is true.
false	true	true	
true	false	true	(age > 34) || (gender == 'M') is false, because (age > 34) and (gender == 'M') are both false.
true	true	true	

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:

TestBooleanOperators

Run

Truth Table for Operator !

p	!p
true	false
false	true

Example
!(1 > 2) is true, because (1 > 2) is false.
!(1 > 0) is false, because (1 > 0) is true.

Truth Table for Operator &&

p1	p2	p1 && p2
false	false	false
false	true	false
true	false	false
true	true	true

Example

(3 > 2) && (5 >= 5) is true, because (3 > 2) and (5 >= 5) are both true.

(3 > 2) && (5 > 5) is false, because (5 > 5) is false.

Truth Table for Operator ||

p1	p2	p1 || p2
false	false	false
false	true	true
true	false	true
true	true	true

Example

(2 > 3) || (5 > 5) is false, because (2 > 3) and (5 > 5) are both false.

(3 > 2) || (5 > 5) is true, because (3 > 2) is true.

Truth Table for Operator ^

p1	p2	p1 ^ p2	Example (assume age = 24, gender = 'F')
false	false	false	(age > 34) ^ (gender == 'F') is true, because (age > 34) is false but (gender == 'F') is true.
false	true	true	
true	false	true	(age > 34) || (gender == 'M') is false, because (age > 34) and (gender == 'M') are both false.
true	true	false	

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)));
```

TestBooleanOperators

Run

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

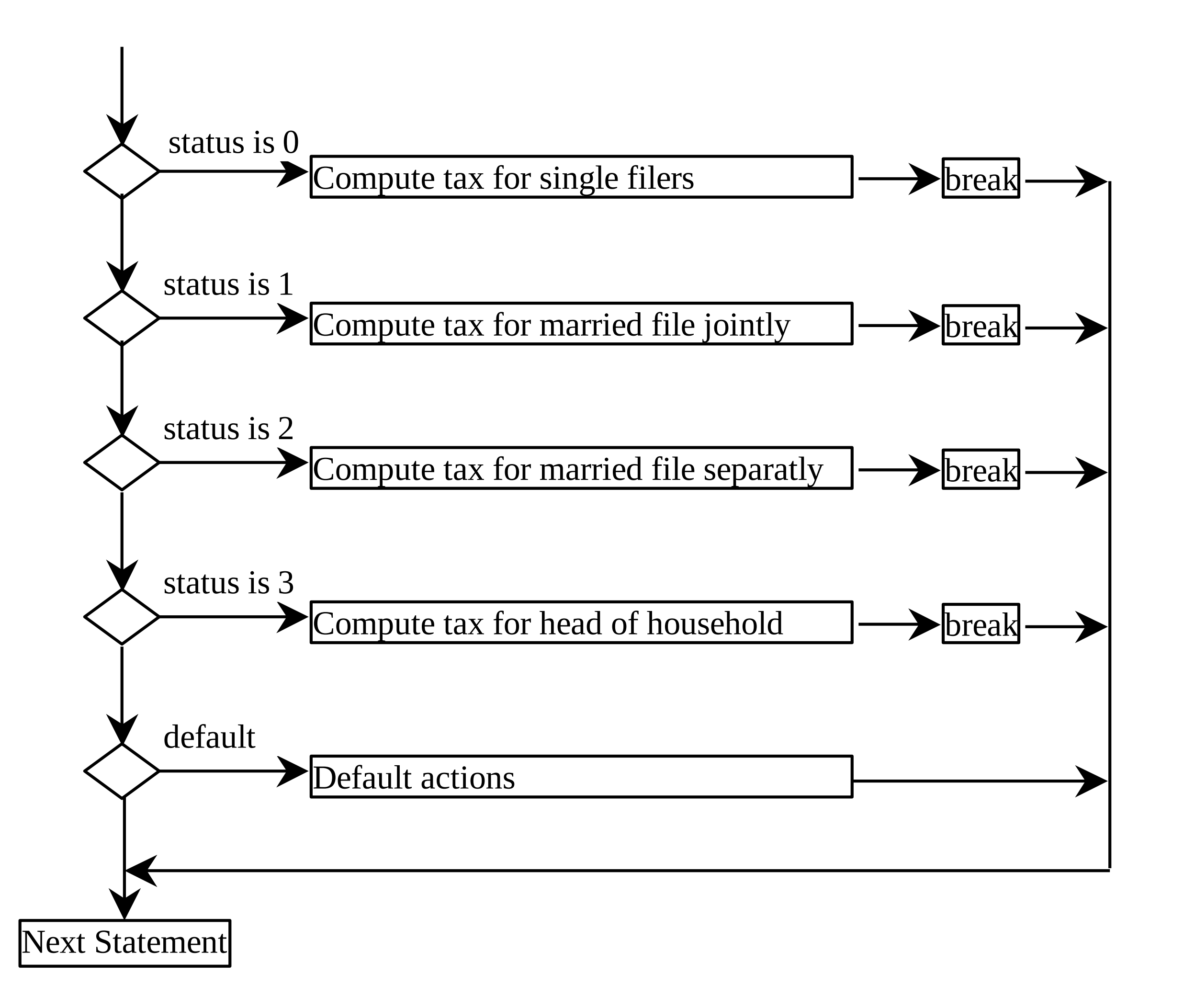
Write a program that randomly generates a lottery of a two-digit number, prompts the user to enter a two-digit number, and determines whether the user wins according to the following rule:

- 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 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(0);
}
```

`switch` Statement Flow Chart


status is 0
Compute tax for single filers
break
status is 1
Compute tax for married file jointly
break
status is 2
Compute tax for married file separatly
break
status is 3
Compute tax for head of household
break
default
Default actions
Next Statement

`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.

```
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.

The case statements are executed in sequential order, but the order of the cases (including the default case) does not matter. However, it is good programming style to follow the logical sequence of the cases and place the default case at the end.

animation

Trace switch statement

Suppose ch is 'a':

```
switch (ch) {
  case 'a': System.out.println(ch);
  case 'b': System.out.println(ch);
  case 'c': System.out.println(ch);
}
```

animation

Trace switch statement

ch is 'a':

```
switch (ch) {
  case 'a': System.out.println(ch);
  case 'b': System.out.println(ch);
  case 'c': System.out.println(ch);
}
```

animation

Trace switch statement

Execute this line

```
switch (ch) {
  case 'a': System.out.println(ch);
  case 'b': System.out.println(ch);
  case 'c': System.out.println(ch);
}
```

animation

Trace switch statement

Execute this line

```
switch (ch) {
  case 'a': System.out.println(ch);
  case 'b': System.out.println(ch);
  case 'c': System.out.println(ch);
}
```

animation

Trace switch statement

Execute this line

```
switch (ch) {
  case 'a': System.out.println(ch);
  case 'b': System.out.println(ch);
  case 'c': System.out.println(ch);
}
```

animation

Trace switch statement

Execute next statement

```
switch (ch)
   case 'a'   System.out.println(ch);
   case 'b':  System.out.println(ch);
   case '  ': System.out.println(ch);
}
```

Next statement;

animation

Trace switch statement

Suppose ch is 'a':

```
switch (ch) {
   case 'a': System.out.println(ch);
             break;
   case 'b': System.out.println(ch);
             break;
   case 'c': System.out.println(ch);
}
```

animation

Trace switch statement

ch is 'a':

```
switch (ch) {
   case 'a': System.out.println(ch);
             break;
   case 'b': System.out.println(ch);
             break;
   case 'c': System.out.println(ch);
}
```

animation

Trace switch statement

Execute this line

```
switch (ch) {
  case 'a': System.out.println(ch);
            break;
  case 'b': System.out.println(ch);
            break;
  case 'c': System.out.println(ch);
}
```

animation

Trace switch statement

Execute this line

```
switch (ch) {
  case 'a': System.out.println(ch);
            break;
  case 'b': System.out.println(ch);
            break;
  case 'c': System.out.println(ch);
}
```

animation

Trace switch statement

Execute next statement

```
switch (ch)
   case 'a': System.out.println(ch);
             break;
   case 'b': System.out.println(ch);
             break;
   case 'c': System.out.println(ch);
}
```

Next statement;

Conditional Operator

```
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 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.*

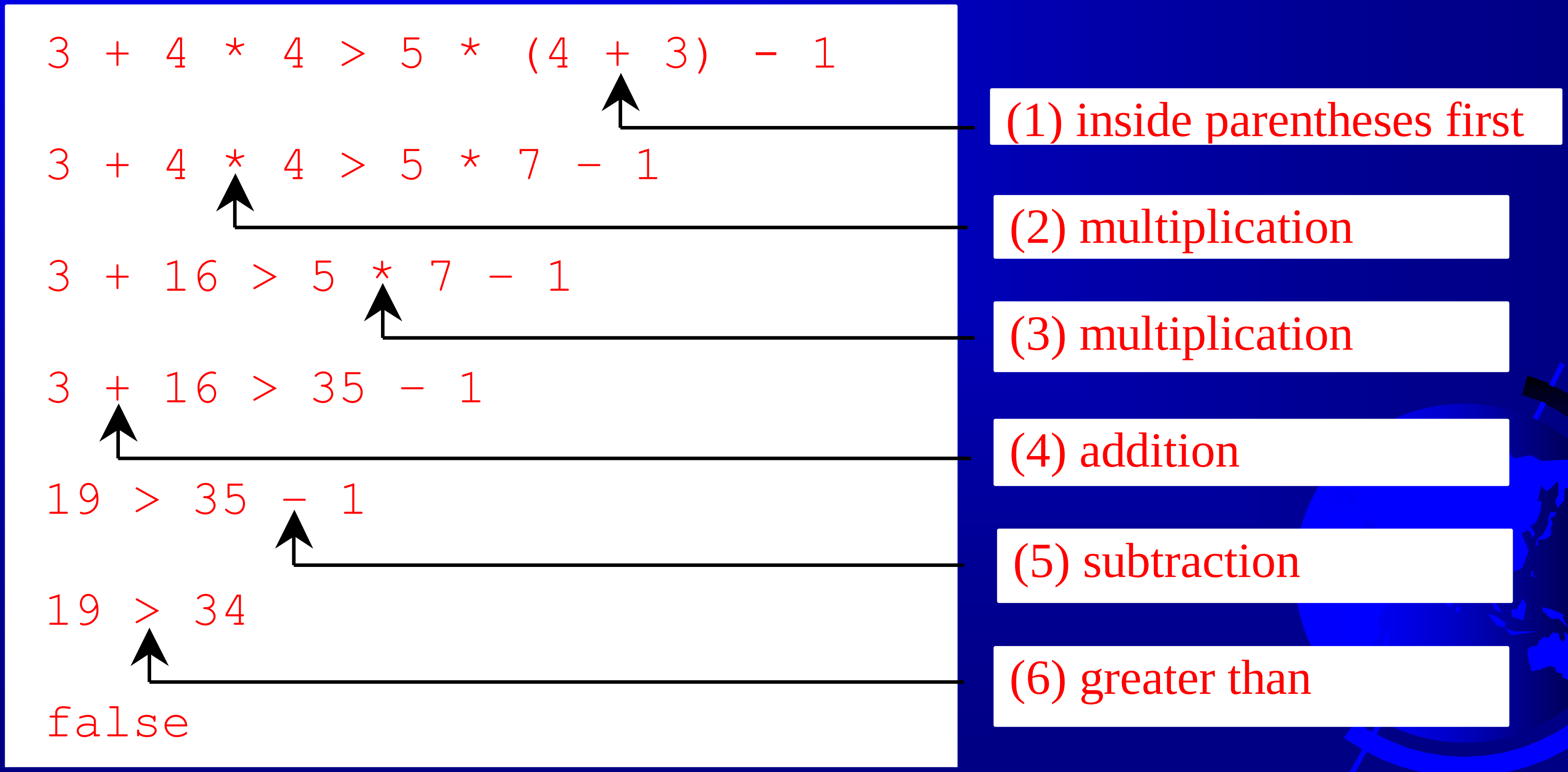
a – b + c – d is equivalent to ((a – b) + c) – d

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:



```
3 + 4 * 4 > 5 * (4 + 3) - 1

3 + 4 * 4 > 5 * 7 – 1

3 + 16 > 5 * 7 – 1

3 + 16 > 35 – 1

19 > 35 – 1

19 > 34

false
```

(1) inside parentheses first

(2) multiplication

(3) multiplication

(4) addition

(5) subtraction

(6) greater than