Elementary Programming - 2

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
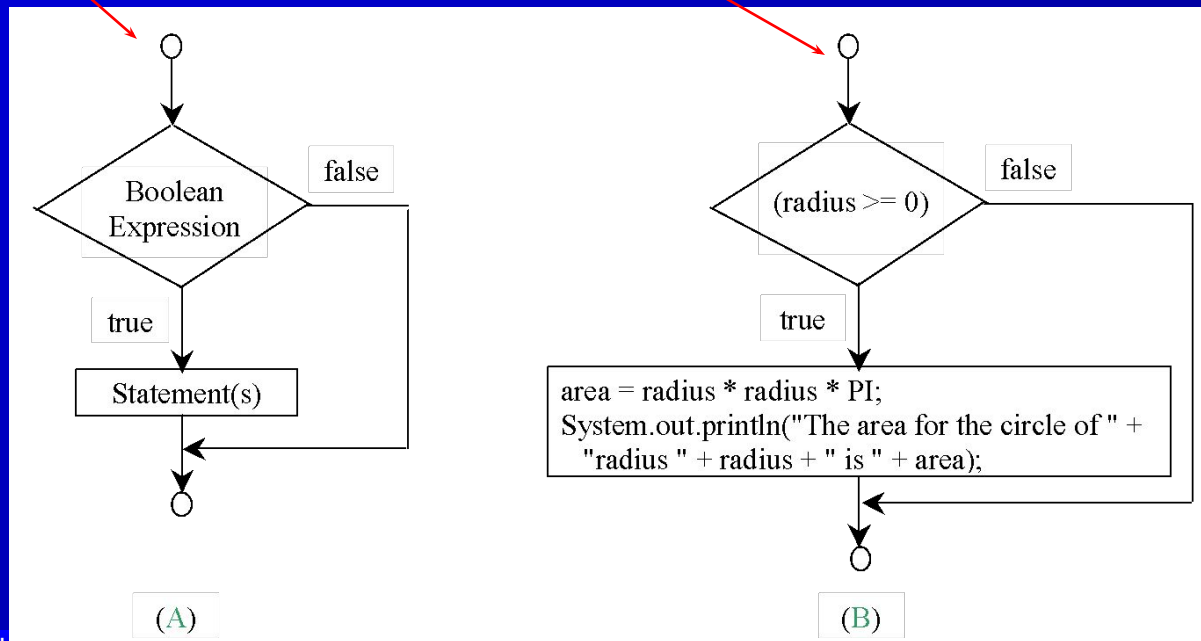
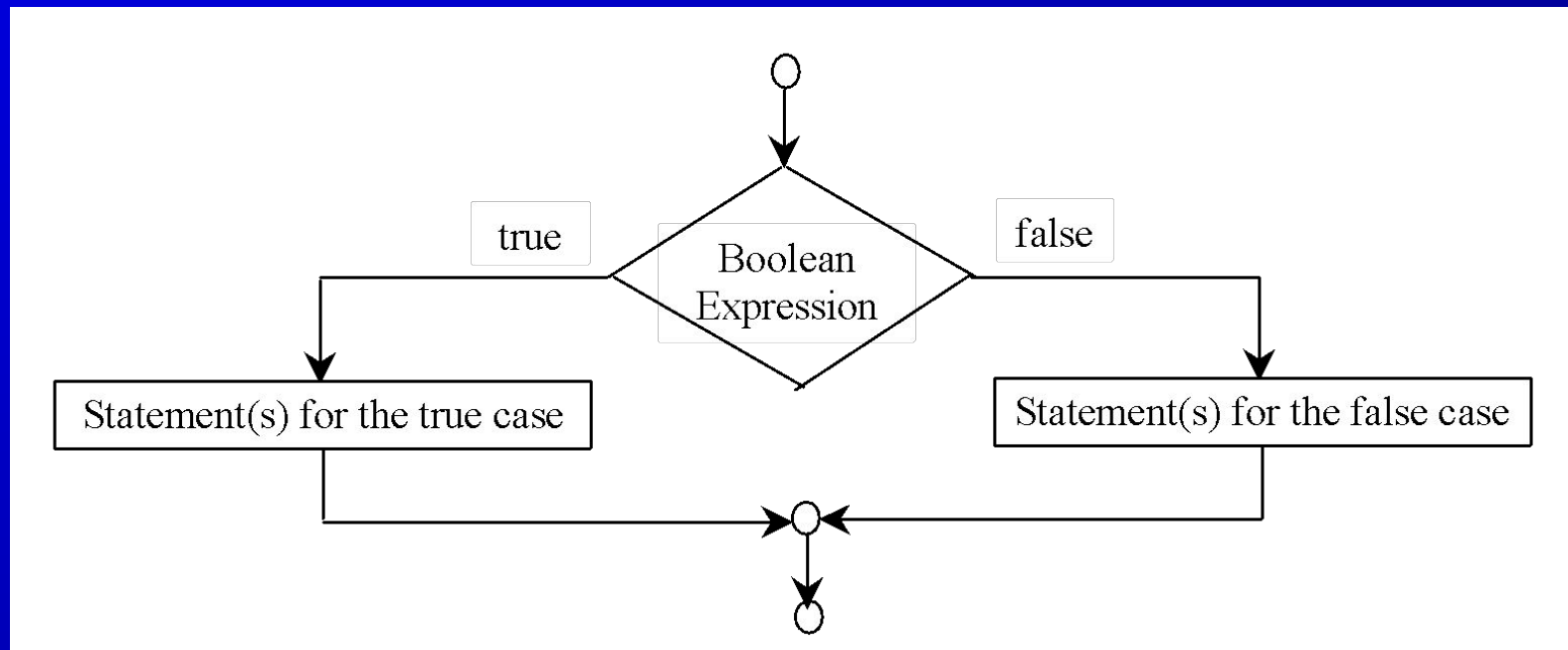>=   greater than or equal to

==   equal to

!=   not equal to

One-way `if` Statements

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

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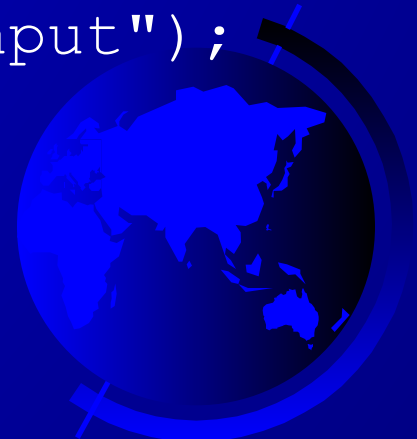

(A)

(B)

The Two-way `if` Statement

```
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

```
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';
```

Note

The <u>else</u> clause matches the most recent <u>if</u> 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)

Logical Operators

Operator   *Name*

!   not

&&   and

|| 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 <u>2</u> and <u>3</u>, whether a number is divisible by <u>2</u> or <u>3</u>, and whether a number is divisible by <u>2</u> or <u>3</u> but not both:

Problem: Determining Leap Year?

This program first prompts the user to enter a year as an <u>int</u> 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)
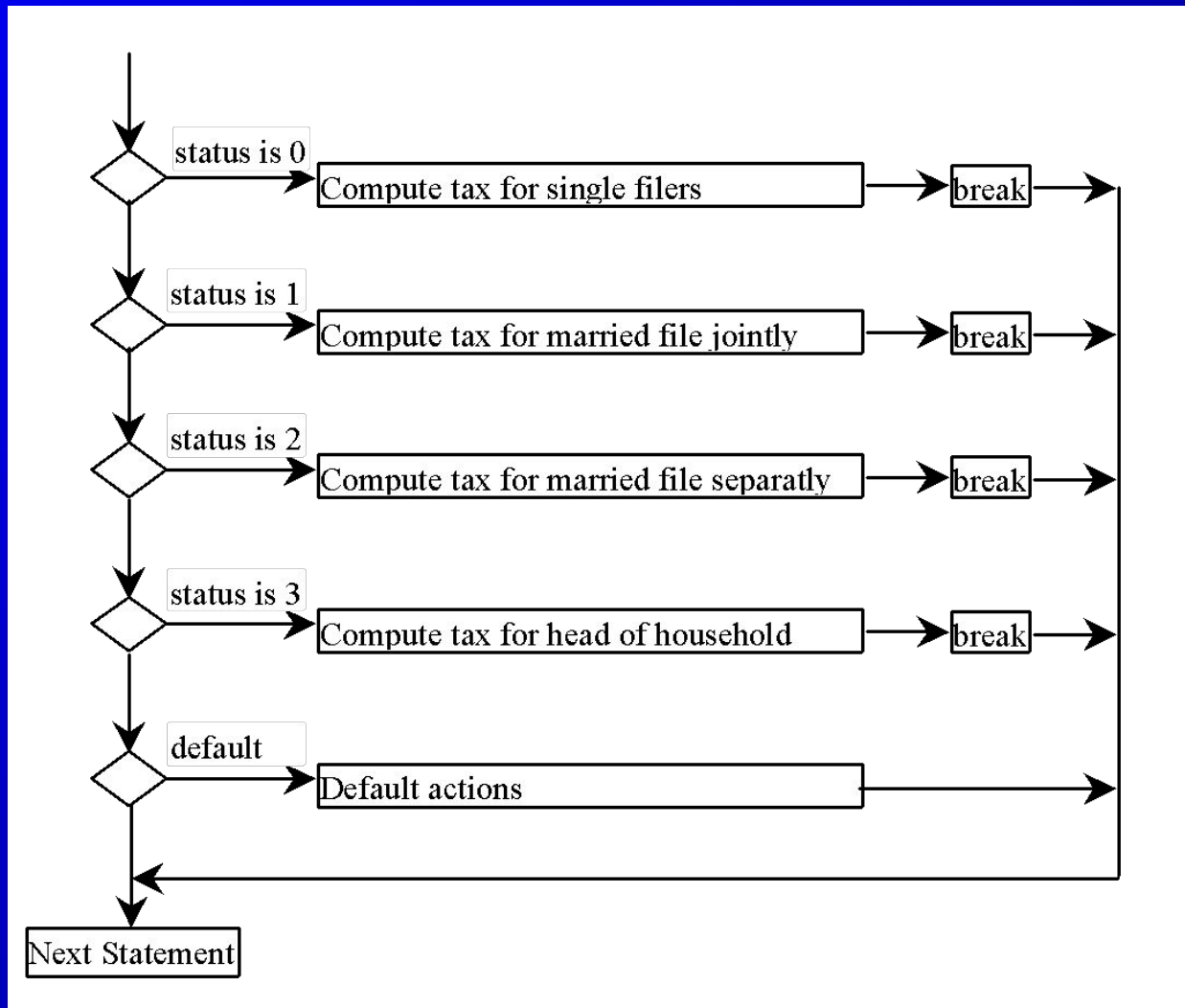
`switch` Statements

```
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

switch Statement Rules

The switch-expression must yield a value of char, byte, short, string, 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;
}
```
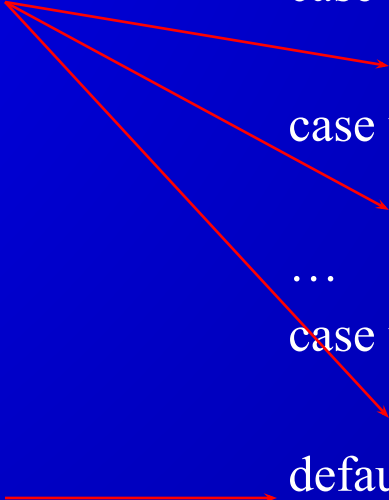
`switch` Statement Rules

The keyword <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;
}
```

The <u>case</u> 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.

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);
}
```

Next Statement

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);
}
Next Statement;
```

Formatting Output

Use the printf statement.

System.out.printf(format, items);

Where format is a string that may consist of substrings and format specifiers. A format specifier specifies how an item should be displayed. An item may be a numeric value, character, boolean value, or a string. Each specifier begins with a percent sign.

Frequently-Used Specifiers

Specifier Output   Example

%b   a boolean value   true or false

%c   a character   'a'

%d   a decimal integer   200

%f   a floating-point number   45.460000

%e   a number in standard scientific notation   4.556000e+01

%s   a string   "Java is cool"

```
int count = 5;
double amount = 45.56;                                items
System.out.printf("count is %d and amount is %f", count, amount);




display         count is 5 and amount is 45.560000
```

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))

Precedence and Associativity Table

Highest			
( )	[ ]	.	
++	– –	~	!
*	/	%	
+	–		
>>	>>>	<<	
>	>=	<	<=
==	!=		
&			
^			
&&			
||			
?:			
=	op=		
Lowest			