Chapter 3 Selections

Contents

[What is a Boolean Expression? 1](#_Toc41850869)

[boolean Data Type 1](#_Toc41850870)

[Comparison Operators: 2](#_Toc41850871)

[Common Errors with relational operators 3](#_Toc41850872)

[Selection Statements 3](#_Toc41850873)

[if Statements 3](#_Toc41850874)

[Two-Way if-else Statements 6](#_Toc41850875)

[if-else if -else Statements 7](#_Toc41850876)

[Generating Random Numbers 9](#_Toc41850877)

[Logical Operators 10](#_Toc41850878)

[Switch Statements 10](#_Toc41850879)

[Conditional Expressions 13](#_Toc41850880)

[System.exit(status) 13](#_Toc41850881)

[Operator Precedence 14](#_Toc41850882)

[Precedence Order 14](#_Toc41850883)

[Associativity 14](#_Toc41850884)

[Precedence and associativity of Java operators 15](#_Toc41850885)

# What is a Boolean Expression?

A Boolean expression is an expression that evaluates to a Boolean value: true or false.

# boolean Data Type

boolean data type declares a variable with the value either true or false.

* Example: boolean lightsOn = true;
* true and false are literals, like int num = 10, & are **reserved** words
* Example: boolean b = (1 > 2);

# Comparison Operators:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Java Operator** | **Mathematics Symbol** | **Name** | **Example**  radius = 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 |

Notice that the comparison operators are <=, not ≤; or !=, not ≠; or ==, not =

If you type = for == then you are assigning, not comparing! It is tedious, but very important that you pay attention to the comparison operators.

* You can compare characters. Comparing characters is the same as comparing their Unicodes.
* These comparison operators only work for primitive data types; they do not work on reference data types.
* The way the comparison operators work is to use the value of each variable’s value in main memory

main Memory

|  |  |
| --- | --- |
| n1 | 1 |
| n2 | 2 |
| areTheseTheSame | Step1: get the value of n1  Step2: get the value of n2  Step3: compare values using ==  Step4: assign false |

* + Example:

int n1 = 1;

int n2 = 2;

boolean areTheseTheSame = n1 == n2;

## Common Errors with relational operators

* 1 <= numberOfDaysInAMonth <= 31

Should be: 1 <= numberOfDaysInAMonth && numberOfDaysInAMonth <= 31

* if(a = b)
* Using relational operators for String data types. String data types are a reference data type and must use methods for comparison. We will learn those in chapter 4.

Use care when using relational operators with float and double data types because those data types do not store the values with complete accuracy.

# Selection Statements

Java provides selection statements – statements that let you choose actions with alternative courses. For example, you want to be able to do something when a variable hits a certain value.

Such as, you are the boss and want to be able to provide a pay raise to employees who worked extra hard to pass a certificate exam.

Your code would be something like:

int score = 90;

if(score >= 70){

payRate = payRate + payRate \* .03;

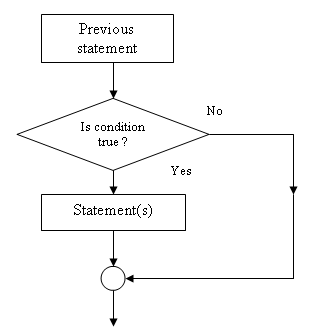
}

Thus, you are giving a 3% pay increase to those emplyees.

Selection statement use conditions that are Boolean expressions (true or false)

# if Statements

* An if statement executes the statements if the condition is **true**. When the condition is false, nothing is done. It has to have a Boolean test in the ( ) or it will be a compile error.



* Block braces can be omitted if they enclose a single statement.
* Omitting block braces makes the code shorter, but is prone to errors.
* Examples:

…

if (number % 5 == 0)

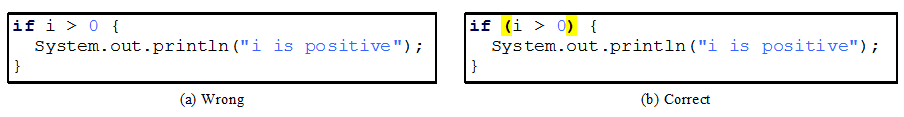
System.out.println(“HiFive!”);

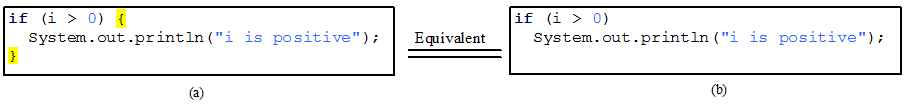
if (number % 2 == 0)

System.out.println(“HiEven!”);

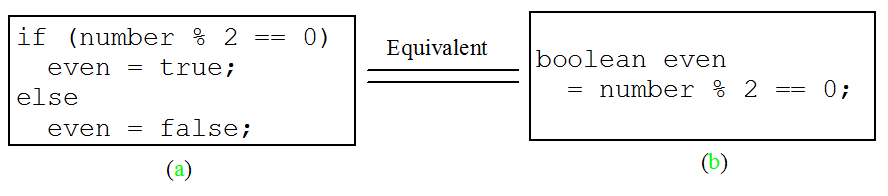
…

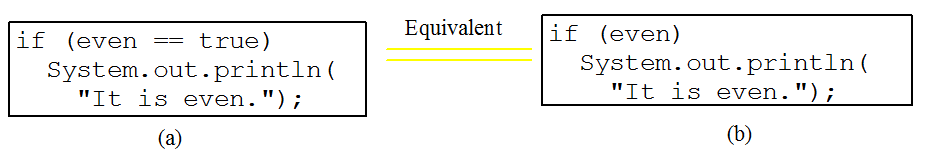
* Some examples of common errors:





**Tip:** if you are trying to assign a boolean variable, just assign – do not perform an if statement to do so. That is considered bad programming.



****

**Tip:** you do not need to perform an extra test to see if the boolean variable is true or false – it already has that kind of value. (Remember that if statements are only executed if the Boolean test is true.) That is redundant testing is considered bad form.

**Common Error**: 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);

}

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.

**Common Error:** Forgetting Necessary Braces

if (radius >= 0)

area = radius \* radius \*PI;

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

if (radius >= 0){

Is not associated with the if statement –

it is the 2nd line after the if statement

area = radius \* radius \*PI;

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

}

Keep in mind that just because the test evaluated false, does not necessarily mean the opposite of. For example:

boolean lightsOn = true;

if(lightsOn){

System.out.println(“It must be dark”);

But is that the case? What about the cars that automatically keeps lights on, or maybe it is raining and the lights are on for safety, etc.

# Two-Way if-else Statements

* An if-else statement decides which statements to execute based on the Boolean test in the if-else statement. If the Boolean expression evaluates to true, the if block is executed; if the Boolean expression evaluates to false, then the else block is executed.
* You can only test for ***one*** Boolean expression
* else does **NOT** test!
* The else is the default case – what you want to happen when the Boolean expression is false.
* Example:

if (Boolean-expression){

statement(s)-for-the-true-case;

}

else{

statement(s)-for-the-false-case;

}

if (radius >= 0) {

area = radius \* radius \* 3.14159;

System.out.println("The area for the “

+ “circle of radius " + radius +

" is " + area);

}

else {

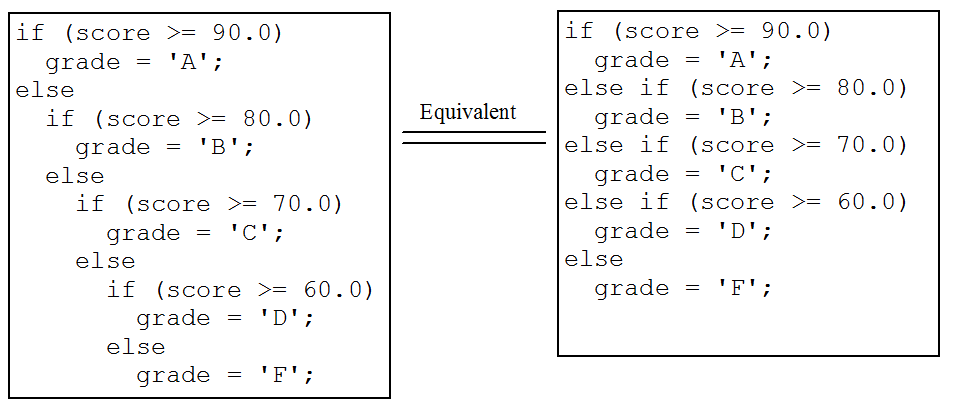
System.out.println("Negative input");

}

# if-else if -else Statements

* You use if-else if-else statements when you have multiple things to test for.
* You have a Boolean expression/test on the if, and each additional else if. The else does not test.
* You use this structure when the Boolean expressions/tests are related.
  + If the Boolean expressions/tests are not related, just use multiple if statements.
* The JVM will evaluate the sequence of Boolean expressions sequentially – in the order that you (the programmer) put the tests. So make sure to fall/travel through the sequence correctly.
* Once the JVM finds a match in the tests (the first match that evaluates to T), the JVM leaves the structure. Even if other tests could evaluate to T, those will not be evaluated. IT goes with the FIRST match & leaves.

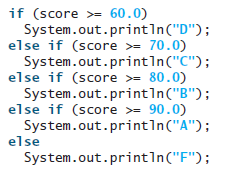




The box on the **right** **is** **correct** – it is clear and concise in format.

The box on the left, is syntactically correct, but looks messy and hard to follow at first glance.

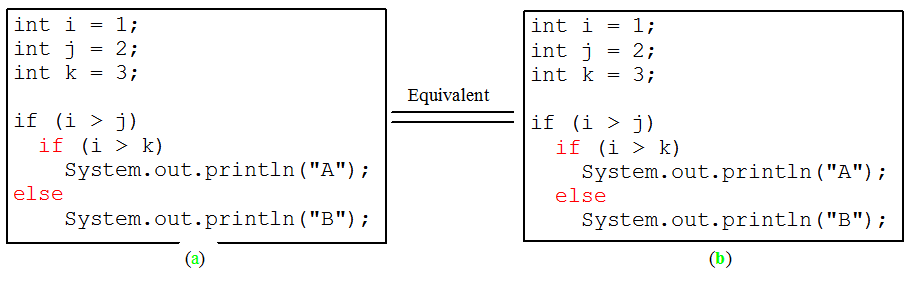
The following example is **incorrect**, the Boolean expressions/tests are not set up in a sequential order; hence, the Boolean expressions/tests produce an invalid result.



* If score is 90, what result would you get? A D, because 90 is greater than or equal to 60.0. Is that correct? No, the sequence of Boolean expressions/tests are not in proper order.

\*\*A thing to keep in mind\*\*

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



* 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.

# Generating Random Numbers

Being able to generate random numbers is a very useful ability.

The Math class has a method, random, that can do this for us.

To be able to use the method, we need to understand how it works.

* First, the method returns (provides back to us) a double value. Meaning if you need an int value, you will have to cast it.
* Second, that double value is such that the number is between 0.0 and 1.0 (including 0.0, but not including 1.0).

0.0 <= number < 1.0

* We can manipulate this value to fit whatever we need in the program. We can do this by using the formula:

(int)(a + Math.random() \* b)

This provides a number generated between a and a+b, excluding a+b

Where a is the number we want to start at, like 1, and b is how many numbers we need, like 10 .

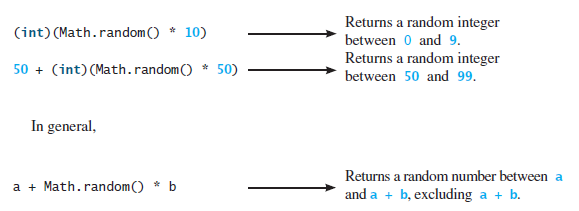
For example, we need a number between 1 and 10, not including 10:

a = 1 (the number we want to start at)

b = 10 (how many numbers we need/ the range)

(int)(1 + Math.random() \* 10;

Thus, we get a number starting at 1, that is providing 10 choices (how many numbers there are between 1 & 10).

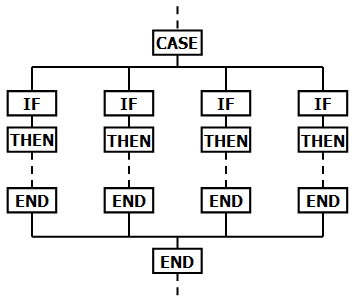


# Logical Operators

See the Truth Table Notes.

# Switch Statements

* A switch statement executes statements based on the value of a variable or an expression.
* Only works when looking for ***exact*** matches
* Simplifies overuse of nested if statements to make the program easier to read



* Switch statements observe the following rules:
  + The switch-expression must yield a value of char, byte, short, int, or String and must always be enclosed in parentheses.
  + The values must have the same data type as the value of the switch-expression and must a constant expression, not a variable.
  + When the value in a case statement matches the value of the switch-expression, the statements *starting from this case on* are all executed until it reaches the end of the statement cases – unless a break statement is included in the case statements forcing the JVM to leave the switch.
    - break keyword forces the JVM to leave the structure it is currently in.
  + The keyword break is optional. The break statement immediately ends the switch statement.
  + The default case, which is optional, can be used to perform actions when none of the specified cases match the switch-expression.
    - You use the default when you want to “catch” something when it does not match any of the cases.
  + The JVM executes the case statements in the order that you, the programmer, put them in. Meaning – make sure it falls through the statements correctly and produces a valid output.

Example:

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;

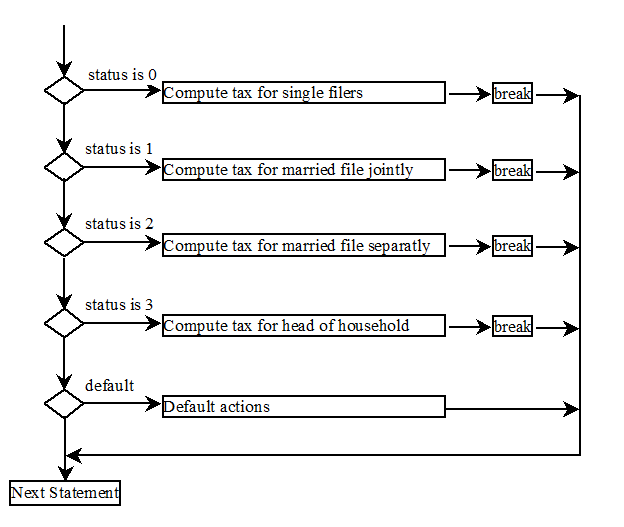
break;

default: System.out.println("Errors: invalid status");

System.exit(0);

}

Flow Chart:



# Conditional Expressions

A conditional expression evaluates an expression based on a condition.

* Helps to assign a value to a variable that is restricted by certain conditions.
* For example, the following could be replaced by a conditional expression.

if (x > 0) // boolean-expression

y = 1 // exp1

else

y = -1; // exp2

(boolean-expression) ? exp1 : exp2

(boolean-expression) ? expression1 : expression2

The above is equivalent to

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

Ternary operator – uses 3 operands

(Binary operator, Unary operator)

System.exit(status) is a method in the System class. Invoking this method terminates the program.

* 0 means that the program terminates normally.
* A nonzero status code means an abnormal termination.

# Operator Precedence

***Precedence Operator***

var++, var--

+, - (Unary plus and minus), ++var,--var

(type) Casting

! (Not)

\*, /, % (Multiplication, division, and remainder)

+, - (Binary addition and subtraction)

<, <=, >, >= (Comparison)

==, !=; (Equality)

^ (Exclusive OR)

&& (Conditional AND) Short-circuit AND

|| (Conditional OR) Short-circuit OR

=, +=, -=, \*=, /=, %= (Assignment operator)

Operator precedence and associativity determine the order in which operators are evaluated.

Some are left associated and some are right.

Precedence Order. When two operators share an operand the operator with the higher precedence goes first. For example, 1 + 2 \* 3 is treated as 1 + (2 \* 3), whereas 1 \* 2 + 3 is treated as (1 \* 2) + 3 since multiplication has a higher precedence than addition.

Associativity. When two operators with the same precedence the expression is evaluated according to its associativity. For example x = y = z = 17 is treated as x = (y = (z = 17)), leaving all three variables with the value 17, since the = operator has right-to-left associativity (and an assignment statement evaluates to the value on the right hand side). On the other hand, 72 / 2 / 3 is treated as (72 / 2) / 3 since the / operator has left-to-right associativity.

What does the following code fragment print?

|  |
| --- |
| System.out.println(1 + 2 + "abc");  System.out.println("abc" + 1 + 2); |

*Answer*: 3abc and abc12, respectively. The + operator is left associative, whether it is string concatenation or arithmetic plus.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Level** | **Associativity** |
| [] . () ++ -- | access array element access object member invoke a method post-increment post-decrement | 1 | left to right |
| + -- + - ! ~ | pre-increment pre-decrement unary plus unary minus logical NOT bitwise NOT | 2 | right to left |
| () new | cast object creation | 3 | right to left |
| \* / % | multiplicative | 4 | left to right |
| + - + | additive string concatenation | 5 | left to right |
| << >> >>> | shift | 6 | left to right |
| < <=  > >=  instanceof | relational type comparison | 7 | left to right |
| == != | equality | 8 | left to right |
| & | bitwise AND | 9 | left to right |
| ^ | bitwise XOR | 10 | left to right |
| | | bitwise OR | 11 | left to right |
| && | conditional AND | 12 | left to right |
| || | conditional OR | 13 | left to right |
| ?: | conditional | 14 | right to left |
| = += -=  \*= /= %=  &= ^= |=  <<= >>= >>>= | assignment | 15 | right to left |

# Precedence and associativity of Java operators