**CHAPTER 3: SELECTIONS**

* 1. **Boolean DATA TYPE, VALUES, AND EXPRESSIONS**

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Java operator | Mathematic 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 |
|  |  |  |  |  |

**NOTE:** The result of the comparison is a Boolean value: true or false.

* 1. **If STATEMENT**

An **if statement** is a construct that enables a program to specify alternative paths of execution.

Java has several types of selection statements: **one-way if statements, two-way if-else statements, nested if statements, multi-way if-else statements, switch statements, and conditional operators.**

A **one-way if statement** executes an action if and only if the condition is true.

The syntax for a **one-way if statement** is as follows:

**if (boolean-expression) {**

**statement(s);**

**}**

Here an if statement executes statements if the boolean-expression evaluates to true.

* 1. **Two-Way if-else Statements**

An **if-else statement** decides the execution path based on whether the condition is true or false.

The actions that **a two-way if-else statement** specifies differ based on whether the condition is true or false.

Here is the syntax for a two-way if-else statement:

**if (boolean-expression) {**

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

**}**

**else {**

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

**}**

**NOTE:** An if-else statement executes statements for the true case if the Boolean-expression evaluates to true; otherwise, statements for the false case are executed.

Example:

**if (radius >= 0) { false**

**area = radius \* radius \* PI;**

**System.out.println("The area for the circle of radius " + radius + " is " + area);**

**}**

**else {**

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

**}**

**NOTE:** If radius >= 0 is **true**, area is computed and displayed; if it is, the message **"Negative input"** is displayed.

* 1. **Nested if and Multi-Way if-else Statements**

An **if statement** can be inside **another if statement** to form a **nested if statement**.

The inner if statement can contain another if statement; in fact, there is no limit to the depth of the nesting. For example,

The following is a nested if statement:

**if (i > k) {**

**if (j > k)**

**System.out.println("i and j are greater than k");**

**}**

**else**

**System.out.println("i is less than or equal to k");**

Note: The if (j > k) statement is nested inside the if (i > k) statement.

Example:

if (x > 2) {

if (y > 2) {

z = x + y; System.out.println("z is " + z);

}

}

else

System.out.println("x is " + x);

* 1. **Common Errors and Pitfalls**

Forgetting necessary braces, ending an if statement in the wrong place, mistaking == for =, and dangling else clauses are common errors in selection statements.

Duplicated statements in if-else statements and testing equality of double values are common pitfalls.

The following errors are common among new programmers.

* **Common Error 1: Forgetting Necessary Braces**
* **Common Error 2: Wrong Semicolon at the if Line**
* **Common Error 3: Redundant Testing of Boolean Values**

**To test whether a boolean variable is true or false in a test condition, it is redundant to use the equality testing operator**

* **Common Error 4: Dangling else Ambiguity**
* **Common Error 5: Equality Test of Two Floating-Point Values**

The following are common pitfalls:

* **Common Pitfall 1: Simplifying Boolean Variable Assignment**
* **Avoiding Duplicate Code in Different Cases**
  1. **Generating Random Numbers**

You can use **Math.random()** to obtain a random double value between 0.0 and 1.0, excluding 1.0

Suppose you want to develop a program for a first-grader to practice subtraction. The pro

gram randomly generates two single-digit integers, number1 and number2, with number1 >= number2, and it displays to the student a question such as “What is 9 - 2?” After the student enters the answer, the program displays a message indicating whether it is correct.

The program can work as follows:

* Generate two single-digit integers into number1 and number2.
* If number1 < number2, swap number1 with number2.
* Prompt the student to answer, "What is number1 − number2?"
* Check the student’s answer and display whether the answer is correct
  1. **Case Study: Computing Body Mass Index**

You can use nested if statements to write a program that interprets body mass index.

Body mass index (BMI) is a measure of health based on height and weight. It can be calculated by taking your weight in kilograms and dividing it by the square of your height in meters.

The interpretation of BMI for people 20 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 |

* 1. **Case Study: Computing Taxes**

You can use **nested if statements** to write a program for computing taxes.

* 1. **Logical Operators**

The logical operators **!**, **&&**, **||**,and **^** can be used to create a compound Boolean expression.

Sometimes, whether a statement is executed is determined by a combination of several conditions. You can use logical operators to combine these conditions to form a compound Boolean expression.

Logical operators, also known as Boolean operators, operate on Boolean values to create a new Boolean value.

|  |  |  |
| --- | --- | --- |
| Operator | Name | Description |
| ! | Not | Logical negation |
| && | And | Logical conjunction |
| || | Or | Logical disjunction |
| ^ | Exclusive or | Logical exclusion |

* **NOTE: (condition1 && condition2**) is the same as **!condition1 || !condition2** **(condition1 || condition2**) is the same as **!condition1 && !condition2**
* If one of the operands of an **&&** operator is **false**, the expression is **false**; if one of the operands of an **|**| operator is **true**, the expression is **true**. Java uses these properties to improve the performance of these operators.
  1. **Case Study: Determining Leap Year**

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

A leap year has 366 days. The February of a leap year has 29 days.

You can use the following Boolean expressions to check whether a year is a leap year:

**boolean isLeapYear = (year % 4 == 0);**

**isLeapYear = isLeapYear && (year % 100 != 0);**

**isLeapYear = isLeapYear || (year % 400 == 0);**

**Or you can combine all these expressions into one as follows:**

**isLeapYear = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);**

* 1. **Case Study: Lottery**

The lottery program involves **generating random numbers**, **comparing digits**, and **using Boolean operators.**

The program generates a lottery using the **random()** method and prompts the user to enter a guess).

The program checks the guess against the lottery number in this order:

* First, check whether the guess matches the lottery exactly .
* If not, check whether the reversal of the guess matches the lottery.
* If not, check whether one digit is in the lottery.
* If not, nothing matches and display "**Sorry, no match**".
  1. **switch Statements**

A switch statement executes statements based on the value of a variable or an expression.

Java provides a switch statement to simplify coding for multiple conditions.

The full syntax for switch statement is:

switch (switch-expression) {

case value1: statement1;

break;

case value2: statement2

break;

case 2: statement3;

break;

case 3: statementn;

break;

default: statement-for-default;

}

The switch statement observes the following rules:

* The switch-expression must yield a value of **char**, **byte**, **short**, **int**, or **String** 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.** Note that **value1**, ..., and **valueN** are constant expressions, meaning they cannot contain variables, such as **1 + x**.
* 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.
* The **default** case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.
* The keyword break is optional. The **break** statement immediately ends the **switch** statement.
  1. **Conditional Operators**

A conditional operator evaluates an expression based on a condition.

You might want to assign a value to a variable that is restricted by certain conditions.

For example, the following statement assigns **1** to **y** if **x** is greater than 0 and **−1** to **y** if **x** is less than or equal to **0**:

**if (x > 0)**

**y = 1;**

**else**

**y = −1;**

Alternatively, as in the following example, you can use a conditional operator to achieve the same result.

**y = (x > 0)? 1: −1;**

The symbols “**?”** and “**:”** appearing together is called a **conditional operator** (also known as a **ternary operator** because it uses three operands. It is the only ternary operator in Java.

The syntax to use the operator is as follows:

**boolean-expression? expression1: expression2**

The result of this expression is **expression1** if **boolean-expression** is true; otherwise the result is **expression2**

* 1. **Operator Precedence and Associativity**

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

The expression within parentheses is evaluated first.

When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule.

The precedence rule defines precedence for operators,

* Operators are listed in decreasing order of precedence from top to bottom.
* The logical operators have lower precedence than the relational operators, and the relational operators have lower precedence than the arithmetic operators.
* Operators with the same precedence appear in the same group.

|  |  |
| --- | --- |
| Precedence | Operator |
|  | **var++** and **var—**(Postfix) |
| **+** , **-** (Unary plus and minus), **++var** and **–var** (Prefix) |
| **!** (Not) |
| **\*** , **/**, **%** (Multiplication, division, and remainder) |
| **+**, **-** (Binary addition and subtraction) |
| **<**, **<=**, **>**, **>=**, (Relational) |
| **==**, **!=** (Equality) |
| **^** (Exclusive OR) |
| **&&** (AND) |
| **||** (OR) |
| **?:** (Ternary operator) |
| **=**, **+=**, **-=**, **\*=**, **/=**, **%=** (Assignment operators) |
|  |

**NOTE:**

* 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.
  1. **DEBUGGING**

Debugging is the process of finding and fixing errors in a program.

**Logic errors** are called **bugs**. The process of finding and correcting errors is called **debugging**.

Most helpful features in debugging:

* **Executing a single statement at a time**: The debugger allows you to execute one statement at a time so that you can see the effect of each statement.
* Tracing into or stepping over a method
* **Setting breakpoints**: You can also set a breakpoint at a specific statement.
* **Displaying variables**: The debugger lets you select several variables and display their values.
* **Displaying call stacks**: The debugger lets you trace all of the method calls.
* **Modifying variables**: Some debuggers enable you to modify the value of a variable when debugging.

**THE END!**