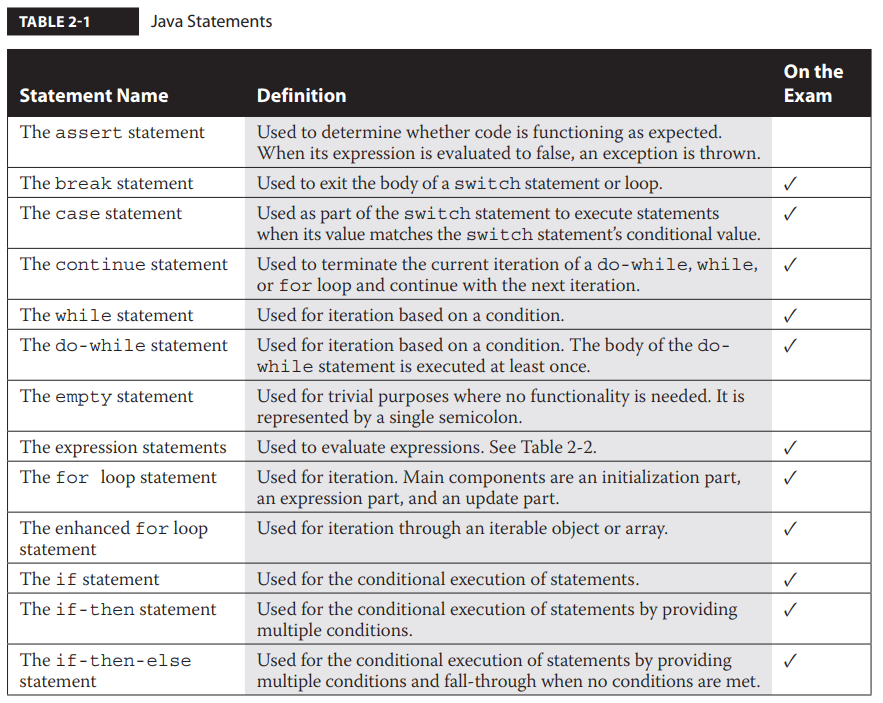
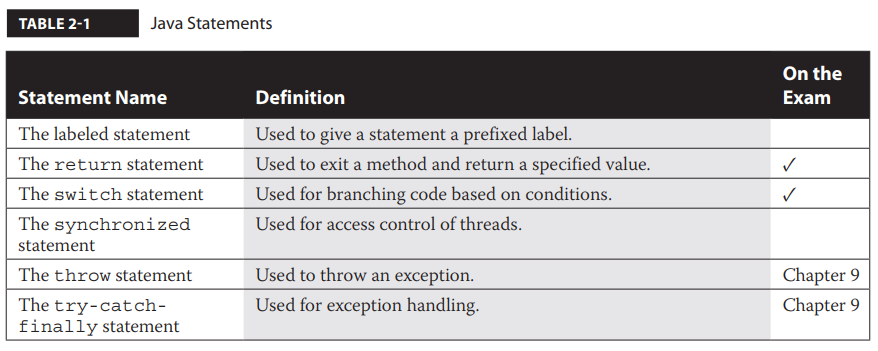
**2**

**Programming with Java Statements**





Expression statements are used for the evaluation of expressions. The assignment expression statements allow assignments to be performed on variables.

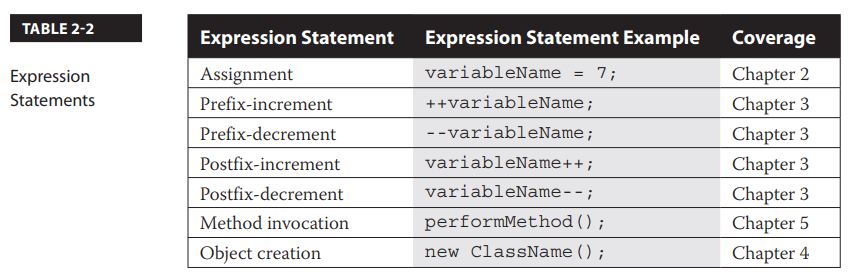
Conditional statements, also known as decision statements, assist in directing the flow of control when a decision needs to be made. Conditional statements include the if, if-then, if-then-else, and switch statements.

Iteration statements provide support in looping through blocks of code. Iteration statements include the for loop, the enhanced for loop, the while statement, and the do-while statement.

Transfer of control statements provide a means of stopping or interrupting the normal flow of control. Transfer of control statements include the continue, break, and return statements. Transfer of control statements are always seen within other types of statements.

**Understand Assignment Statements**

An assignment statement sets a value within a variable. All assignment statements are considered to be expression statements. Expressions in Java are anything that has a value or is reduced to a value. All expressions can be used as statements; the only requirement is that they end with a semicolon.



**The Assignment Expression Statement**

Assignment statements, are designed to assign values to variables. All assignment statements must be terminated with a semicolon.

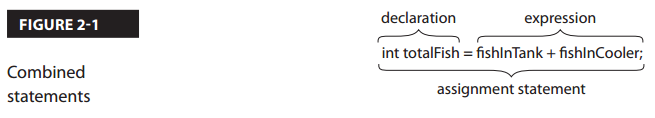
*variable* = *value*;

On the left is the variable that will be associated with the memory and type necessary to store the value. On the right is a literal value. If an expression is on the right, such as (1+2), it must be evaluated down to its literal value before it can be assigned. Lastly, an equal sign resides between the variable and value of an assignment statement.

*int variableName; // Declaration of an integer*

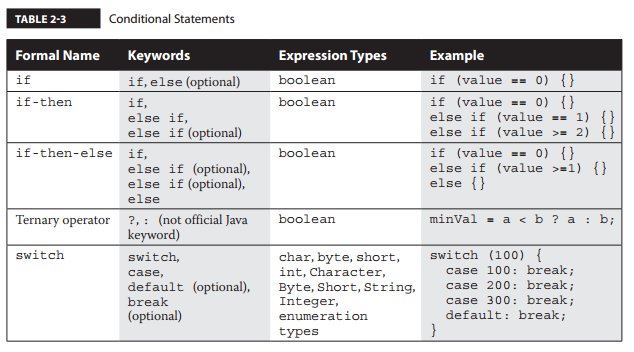
*variableName = 100; // Assignment expression statement*

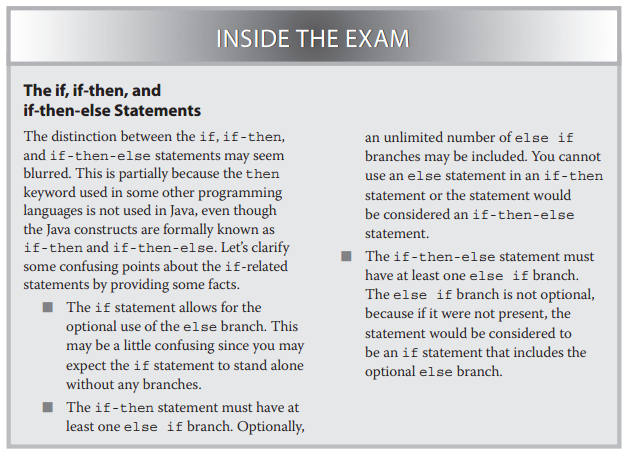
Trying to save an invalid literal to a declared primitive type variable will result in a compiler error. For example, the compilation error Exception in thread "xxxx" java.lang.RuntimeException: Uncompilable source code - incompatible types…



**Create and Use Conditional Statements**

Conditional statements are used when there is a need for determining the direction of flow based on conditions. Conditional statements include the if, if-then, if-then-else, and switch statements.





**The if Conditional Statement**

The if statement is designed to conditionally execute a statement or conditionally decide between a choice of statements. The if statement will execute only one statement upon the condition unless braces are supplied. Braces, also known as curly brackets, allow for multiple enclosed statements to be executed. This group of statements is also known as a *block*. The expression that is evaluated within if statements must evaluate to a boolean value or the application will not compile. The else clause is optional and may be omitted.

if (*expression*)

*statementA*;

else

*statementB*;

Even though relational operators (such as >=) are commonly used, assignment statements are always allowed.

if (bValue = false)

System.out.println("TRUE");

else

System.out.println("FALSE")

The assignment statements of all primitives will return their primitive values:

if (i=1) {}; // will not compile

**The if-then Conditional Statement**

The if-then conditional statement is used when multiple conditions need to flow through a decision-based scenario.

if (*expressionA*)

*statementA*;

else

if (*expressionB*)

*statementB*;

**The if-then-else Conditional Statement**

As with the if and if-then statements, all expressions must evaluate to true or false as the expected primitive type is boolean. The main difference in the if-then-else statement is that the code will fall through to the final stand-alone else when the expression fails to return true for any condition. Each statement may  
optionally be a group of statements enclosed in braces. There is no limit to the number of else if clauses.

*if (expressionA)*

*statementA;*

*else if (expressionB)*

*statementB;*

*else if (expressionC)*

*statementC;*

*…*

*else*

*statementZZ;*

**The Ternary Operator**

The ternary operator is a variation of the if-then-else statement. It is also sometimes referred to as a conditional operator. The ternary operator derives its name from the fact that it is the only operator to use three operands. The ? and : characters are used in this operation.

The ternary operator behaves similarly to the if-then-else statement but never includes any optional else if. The first part must be an expression that results in a boolean value. In this case, testCondition is tested to determine if it is greater than 0. If this expression is true, the ternary operation returns value1, the first value after the ? character. A false will result in value2, the value after the : character, to be returned.

*result = testCondition ? value1 : value2*

Ternary operators are great for checking and returning simple values. However, in a more complex situation, a normal if-then-else statement will often result in code that is easier to read.

**The switch Conditional Statement**

The switch conditional statement is used to match the value from a switch statement expression against a value associated with a case keyword. Once matched, the enclosed statement(s) associated with the matching case value are executed and subsequent case statements are executed, unless a break statement is encountered. The break statements are optional and will cause the immediate termination of the switch conditional statement.

When two case statements within the same switch statement have the same value, a compiler error will be thrown (Compiler error, Error: duplicate case label).

The expression of the switch statement must evaluate to byte, short, int, or char. Wrapper classes of type Byte, Short, Integer, and Character are also allowed because they are automatically unboxed to primitive types. Enumerated types (that is, enum) are permitted as well. Additionally, Java SE 7 added support for evaluation of the String object in the expression.

*switch (expression) {*

*case valueA:*

*// Sequences of statements*

*break;*

*case valueB:*

*// Sequences of statements*

*break;*

*default:*

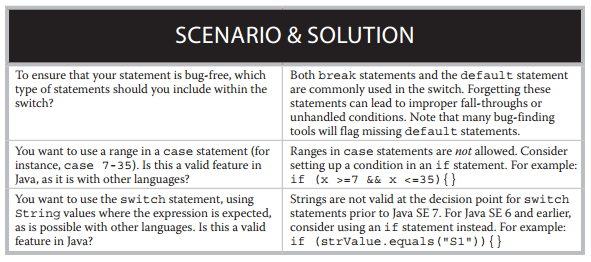
*// Sequences of statements*

*…*

*}*

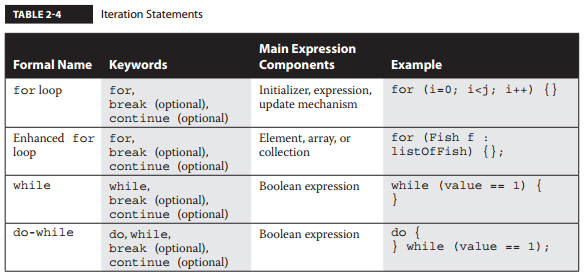
Remember that without break statements, the switch block will continue with its fall-through from the point that the condition has been met.

The default case is often listed last for code readability. Default can be placed anywhere in switch. If the default is placed anywhere but in the end, the cases will be first evaluated. If the case matching the value is after the default statement, the case will be executed. But in case none of the cases match the value, the default will be executed and if the “break” is missing, all cases that precedes it will be executed.



**Create and Use Iteration Statements**

Iteration statements are used when there is a need to iterate through pieces of code. Iteration statements include the for loop, enhanced for loop, and the while and do-while statements. The break statement is used to exit the body of any iteration statement. The continue statement is used to terminate the current iteration and continue with the next iteration.



**The for Loop Iteration Statement**

It has main parts that include an initialization part, an expression part, and an iteration part. The initialization does not need to declare a variable as long as the variable is declared before the for statement. Be aware, though, that the scope of the variable declared within the initialization part of the for loop ends once the for loop terminates. The expression within the for loop statement must evaluate to a boolean value. The iteration, also known as the update part, provides the mechanism that will allow the iteration to occur.

Here’s the general usage of the for statement:

*for ( initialization; expression; iteration) {*

*// Sequence of statements*

*}*

**The Enhanced for Loop Iteration Statement**

The enhanced for loop is used to iterate through an array, a collection, or an object that implements the interface iterable. The enhanced for loop is also commonly known as the for each loop and the for in loop. Iteration occurs for each element in the array or iterable class. Remember that the loop can be terminated at any time by the inclusion of a break statement. And as with the  
other iteration statements, the continue statement will terminate the current iteration and start with the next iteration.

Here’s the general usage of the for statement:

*for (type variable : collection) statement-sequence*

**The while Iteration Statement**

The while statement is designed to iterate through code. The while loop statement evaluates an expression and executes the while loop body only if the expression evaluates to true. Typically, an expression within the body will affect the result of the expression.

Here’s the general usage of the while statement:

*while (expression) {*

*// Sequences of statements*

*}*

**The do-while Iteration Statement**

The do-while statement is designed to iterate through code. It is very similar to the while loop statement, except that it always executes the body at least once. The do-while loop evaluates an expression and continues to execute the body only if it evaluates to true. Typically, an expression within the body will affect the result of the expression.

Here’s the general usage of the do-while statement:

*do {*

*// Sequence of statements*

*}*

*while (expression)*