2 Programming with Java Statements

CERTIFICATION OBJECTIVES

- Understanding Fundamental Statements
- Implementing Statement-Related Algorithms from Pseudo-code

✓ Two-Minute Drill

Q&A Self Test

Software applications are composed of various statements. It is these language statements that allow for the proper sequence of execution and associated functionality to occur. The more statement types a software language has, the more effective the language can be. Table 2-1 provides short definitions of the Java statement types defined in the *Java Language Specification, Third Edition*, by James Gosling, Bill Joy, Guy Steele, and Gilad Bracha (Addison Wesley, June 2005). Those covered on the exam and in this chapter are accompanied by a checkmark. You can refer to the language specification for more details on the statements that are not on the exam.

TABLE 2-1 Java Statements

Statement Name	Definition	On the Exam
The `assert` statement	Used to determine if code is functioning as expected. When its expression is evaluated to false, an exception is thrown.	
The `break` statement	Used to exit the body of a `switch` statement or loop.	✓
The `case` statement	Used as part of the `switch` statement to execute statements when its value matches the `switch` statement's conditional value.	✓
The `continue` statement	Used to terminate the current iteration of a `do-while`, `while`, or `for` loop and continue with the next iteration.	✓
The `while` statement	Used for iteration based on a condition.	✓
The `do-while` statement	Used for iteration based on a condition. The body of the `do-while` statement is executed at least once.	✓
The `empty` statement	Used for trivial purposes where no functionality is needed. It is represented by a single semicolon.	
The expression statements	Used to evaluate expressions. See Table 2-2.	✓
The `for` loop statement	Used for iteration. Main components are an initialization part, an expression part, and an update part.	✓
The enhanced `for` loop statement	Used for iteration through an iterable object or array.	✓
The `if` statement	Used for the conditional execution of statements.	✓
The `if-then` statement	Used for the conditional execution of statements by providing multiple conditions.	✓

TABLE 2-1 Java Statements (*Continued*)

Statement Name	Definition	On the Exam
The `if-then-else` statement	Used for the conditional execution of statements by providing multiple conditions and fall-through when no conditions are met.	✓
The *labeled* statement	Used to give a statement a prefixed label.	
The `return` statement	Used to exit a method and return a specified value.	✓
The `switch` statement	Used for branching code based on conditions.	✓
The `synchronized` statement	Used for access control of threads.	
The `throw` statement	Used to throw an exception.	
The `try-catch-finally` statement	Used for exception handling.	

To be an effective Java programmer, you must master the basic statements. Sun knows this so they have included complete coverage of the basic statements on the exam. In addition, it is common practice for developers to work out detailed algorithms in pseudo-code before coding them up. As such, the exam contains questions to validate your ability to match implemented Java code with pseudo-code algorithms. This chapter will teach you how to recognize and code Java statements, as well as implement and recognize code produced from pseudo-code algorithms.

CERTIFICATION OBJECTIVE

Understanding Fundamental Statements

Exam Objective 4.1 Describe, compare, and contrast these three fundamental types of statements: assignment, conditional, and iteration, and given a description of an algorithm, select the appropriate type of statement to design the algorithm.

The Java programming language contains a variety of statement types. Even though the various statement types serve different purposes, the ones covered against this objective can be grouped into four main categories: expression statements, conditional statements, iteration statements, and transfer of control statements.

Expression statements are used for the evaluation of expressions. The only expression statement required for this objective is the assignment statement. Assignment statements allow for the ability to perform assignments 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` and the `do-while` statements. Transfer of control statements provide a means to stop or interrupt 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. The goal of this chapter is for you to gain the knowledge of when and how to use all of the necessary types of Java statements that will be seen on the SCJA exam. In the upcoming sections, we will examine the following statements:

- Assignment statements
- Conditional statements
- Iteration statements

Assignment Statements

An assignment statement is considered an expression statement. Let's briefly discuss expression statements first. Expression statements essentially work with expressions. Expressions in Java are considered to be anything that has a value or is reduced to a value. Typically, expressions evaluate to primitive types, such as in the case of adding two numbers [for example, `(1+2)`]. Concatenating strings together with the concatenation `(+)` operator results in a string, and is also considered an expression. All expressions can be used as statements; the only requirement is that they end with a semicolon. Of the Java expression statements represented in Table 2-2, the only expression statement covered on the exam via the objectives in this chapter is the assignment expression statement.

The Assignment Expression Statement

Assignment expression statements, commonly known as just assignment statements, are designed to assign values to variables. All assignment statements must be terminated with a semicolon. Having this ability to store information in variables provides the main characteristic of usefulness to computer applications.

TABLE 2-2 Expression Statements

Expression Statement	Expression Statement Example	Coverage
Assignment	`variableName = 7;`	This chapter
Pre-increment	`++variableName;`	Chapter 4
Pre-decrement	`--variableName;`	Chapter 4
Post-increment	`variableName++;`	Chapter 4
Post-decrement	`variableName--;`	Chapter 4
Method invocation	`performMethod();`	Chapter 5
Object creation	`new ClassName();`	Chapter 4

The general usage of the assignment statement:

variable = value;

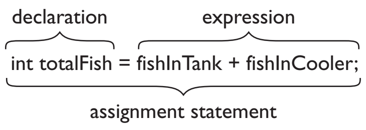
Given the declaration of an integer primitive, let's look at an assignment in its most basic form. There are three key elements. On the left you will find the variable that will be associated with the memory and type necessary to store the value. On the right is a literal value. If an expression is on the right, such as (`1+2`), it must be evaluated down to its literal value before it can be assigned. Lastly, an equal sign will reside between the variable and value of an assignment statement.

```
int variableName; // Declaration of an integer
variableName = 100; // Assignment expression statement
```

For as long as the application is running and the object in which the variable exists is still alive, the value for `variableName` will remain the assigned value, unless it is explicitly changed with another expression statement. The statement, illustrated in Figure 2-1, combines a declaration, an expression, and an assignment statement. In addition, it uses the values stored from previous assignment statements.

```
int fishInTank = 100; int fishInCooler = 50;
int totalFish = fishInTank + fishInCooler;
```

FIGURE 2-1



Combined statements

Trying to save an invalid literal to a declared primitive type variable will result in a compiler error. For more information about working with primitives, see Chapter 4.

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 conditional statements represented in Table 2-3 will be seen on the exam.

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.

TABLE 2-3 Conditional Statements

Formal Name	Keywords	Expression Types	Example
`if`	`if`, `else` (optional)	`boolean`	`if (value == 0) {}`

`if-then`	`if`,
	`else if`,
	`else if` (optional)
		`boolean`	`if (value == 0) {}`
			`else if (value == 1) {}`
			`else if (value >= 2) {}`

`if-then-else`	`if`,
	`else if`,
	`else if` (optional),
	`else`
		`boolean`	`if (value == 0) {}`
			`else if (value >=1) {}`
			`else {}`

`switch`	`switch`,
	`case`,
	`default` (optional),
	`break`
	(optional)
		`char`, `byte`, `short`,
		`int`, `Character`,
		`Byte`, `Short`,
		`Integer`,
		enumeration types
			`switch (100) {`
			  `case 100: break;`
			  `case 200: break;`
			  `case 300: break;`
			  `default: break;`
			`}`

INSIDE THE EXAM

Peculiarities with *if-related* Statements

The distinction between the `if`, `if-then`, and `if-then-else` statements may seem blurred. This is partially because the `then` keyword used in some other programming languages is not used in Java, even though the Java constructs are formally known as `if-then` and `if-then-else`. Let's clarify some confusing points about the `if`-related statements by providing some facts.

The `if` statement allows for the optional use of the `else` branch. This may be a little confusing since you may expect the `if` statement to stand alone without any branches, but it is what it is.

The `if-then` statement must have at least one `else if` branch. Optionally, an unlimited amount of `else if` branches may be included. You cannot use an `else` statement in an `if-then` statement, or the statement would be considered an `if-then-else` statement.

The `if-then-else` statement must have at least one `else if` branch. The `else if` branch is not optional, because if it was not present, the statement would be considered to be an `if` statement that includes the optional `else` branch.

The general usage of the `if` statement:

if (*expression*)
  *statement*A;
else
  *statement*B;

In the following example, we look at the most basic structure of an `if` statement. Here, we check to see if a person (`isFisherman`) is a fisherman, and if so, the expression associated with the `if` statement would evaluate to `true`. Because it is `true`, the example's fishing trip value (`isFishingTrip`) is modified to `true`. No action would have been taken if the `isFisherman` expression evaluated to `false`.

```
boolean isFisherman = true;
boolean isFishingTrip = false;
if (isFisherman)
   isFishingTrip = true;
```

Let's change the code up a little bit. Here, you will see that a fishing trip will only occur if there are one or more fishermen as the expression reads (`fishermen >= 1`). See Chapter 3 for more details on relationship operators (for example, `<`, `<=`, `>`, `>=`, `==`, `!=`). We also see that when "one or more fishermen" evaluates to `true`, a block of statements will be executed.

```
int fishermen = 2;
boolean isFishingTrip = false;
if (fishermen >= 1) {
    isFishingTrip = true;
    System.out.print("Going Fishing!");
}
$ Going Fishing!
```

Executing statements in relationship to `false` conditions is also common in programming. In the following example, when the expression evaluates to `false`, the statement associated with the `else` part of the `if` statement is executed:

```
boolean isFisherman = false;
if (isFisherman) System.out.println("Going fishing!");
else System.out.println("I'm doing anything but fishing!");
$ I'm doing anything but fishing!
```

The `if-then` Conditional Statement

The `if-then` conditional statement—also known as the `if else if` statement—is used when multiple conditions need to flow through a decision-based scenario.

The general usage of the `if-then` statement:

```
if (expressionA)
   statementA;
else if (expressionB)
   statementB;
```

The expressions must evaluate to `boolean` values. Each statement may optionally be a group of statements enclosed in braces.

Let's look at an example. For those not familiar with surf-fishing, when fishing off the beach, a lead pyramid-shaped sinker is used to keep the line on the bottom

INSIDE THE EXAM

The most important thing to remember about the expression in the `if` statement is that it can accept any expression that returns a `boolean` value. Even though relational operators (for example, >=) are commonly used, assignment statements are allowed. Review and understand the following code examples:

```
boolean b;
boolean bValue = (b = true); // Evaluates to true
if (bValue) System.out.println("TRUE");
else System.out.println("FALSE");
if (bValue = false) System.out.println("TRUE");
else System.out.println("FALSE");
if (bValue == false) System.out.println("TRUE");
else System.out.println("FALSE");
$ TRUE
$ FALSE
$ TRUE
```

You also need to be aware that the assignment statements of all primitives will return their primitive values. So, if it's not an assignment of a `boolean` type, then the return value will not be `boolean`.

As such, the following code will not compile:

```
int i; // Valid declaration
int iValue = (i=1); // Valid evaluation to int
/* Fails here as boolean value is expected in the expression */
if (iValue) {};
```

Similarly, this code will not compile:

```
/* Fails here as boolean value is expected in the expression */
if (i=1) {};
```

The compile error will look like this:

```
Error: incompatible types; found: int, required: boolean
```

of the ocean. In the following code segment, conditions are evaluated matching the appropriate `pyramidSinker` by weight against the necessary tide:

```
int pyramidSinker = 3;
System.out.print("A pyramid sinker that weighs " + pyramidSinker
  + "ounces is ");
if (pyramidSinker == 2)
  System.out.print("used for a slow moving tide. ");
else if (pyramidSinker == 3)
  System.out.print("used for a moderate moving tide. ");
else if (pyramidSinker == 4)
  System.out.print("used for a fast moving tide. ");
$ A pyramid sinker that weighs 3 ounces is used for a moderate
  moving tide.
```

We used the string concatenation (+) operator in this example. While the functionality is straightforward, you will want to see Chapter 3 for more information on its behavior.

exam watch

The `if` family of statements evaluate expressions that must result in a `boolean` type where the value is `true` or `false`. Be aware that an object from the `Boolean` wrapper class is also allowed because it will go through unboxing in order to return the expected type. Unboxing is the automatic production of its primitive value in cases where it is needed. The following code demonstrates the use of a `Boolean` wrapper class object within the expression of an `if` statement:

```
Boolean wrapperBoolean = new Boolean ("true");
/* Valid */
boolean primitiveBoolean1 = wrapperBoolean.booleanValue();
/* Valid because of unboxing */
boolean primitiveBoolean2 = wrapperBoolean;
if (wrapperBoolean)
System.out.println("Works because of unboxing");
```

For more information on autoboxing and unboxing, see Chapter 4.

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.

The general usage of the `if-then-else` statement:

```
if (expressionA)
  statementA;
else if (expressionB)
  statementB;
else if (expressionC)
  statementC;
…
else
  statementZZ;
```

In the following code listing, the method `getCastResult()` represents the efforts of a fisherman casting his line out into the ocean. The return value will be a `String` of value "fish," "shark," or "skate" and in this application the value is stored into the `resultOfCast` variable. This `String` value is evaluated against the stipulated string passed into the `equals` method. If the criteria are met for any `if` or `else if` condition, the associated block of code is executed, otherwise the code related to the final `else` is executed. This code clearly demonstrates a complete `if-then-else` scenario.

```
…
private FishingSession fishingSession = new FishingSession();
…
public void castForFish() {
  fishingSession.setCatch();
  String resultOfCast = fishingSession.getCastResult();
  if (resultOfCast.equals("fish")) {
    Fish keeperFish = new Fish();
    keeperFish = fishingSession.getFishResult();
    String type = keeperFish.getTypeOfFish();
    System.out.println("Wahoo! Keeper fish: " + type);
  } else if (resultOfCast.equals("shark")) {
    System.out.println("Need to throw this one back!");
  } else if (resultOfCast.equals("skate")) {
    System.out.println("Yuck, Leo can take this one off the
      hook!");
  } else {
    System.out.println("Darn, no catch!");
  }
}
…

$ Wahoo! Keeper fish: Striped Bass
```

Note that the `Fish` class and associated methods were deliberately not shown since the scope of this example was the `if-then-else` scenario only.

on the job

If abrupt termination occurs during the evaluation of the conditional expression within an `if` statement, then all subsequent `if-then` (that is, `else if`) and `if-then-else` (that is, `else`) statements will end abruptly as well.

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.

exam watch

When two* `case` *statements within the same* `switch` *statement have the same value, a compiler error will be thrown.

```
switch (intValue){
  case 200: System.out.println("Case 1");
  /* Compiler error, Error: duplicate case label */
  case 200: System.out.println("Case 2");
}
```

The expression of the `switch` statement must evaluate to `byte`, `short`, `int`, or `char`. Wrapper classes of type `Byte`, `Short`, `Int`, and `Character` are also allowed since they are automatically unboxed to primitive types. Enumerated types are permitted as well.

The general usage of the `switch` statement:

```
switch (expression) {
  case valueA:
    // Sequences of statements
    break;
  case valueB:
    // Sequences of statements
    break;
  default:
    // Sequences of statements
  …
}
```

Let's take a look at a complete `switch` conditional statement example. In the following `generateRandomFish` method, we use a random number generator to produce a value that will be used in the `switch` expression. The number generated will either be a 0, 1, 2, or 3. The `switch` statement will use the value to match it to the value of a `case` statement. In the example, a `String` with the name `randomFish` will be set depending on the `case` matched. In this example, the only possible value that does not have a matching `case` statement is the number 3. Therefore, this condition will be handled by the `default` statement. Whenever a `break` statement is hit, it will cause immediate termination of the `switch` statement.

```
public String generateRandomFish() {
  String randomFish;
  Random randomObject = new Random();
  int randomNumber = randomObject.nextInt(4);
  switch (randomNumber) {
    case 0:
      randomFish = "Blue Fish";
      break;
    case 1:
      randomFish = "Red Drum";
      break;
      case 2:
        randomFish = "Striped Bass";
        break;
      default:
        randomFish = "Unknown Fish Type";
        break;
    }
    return randomFish;
  }
```

The `case` statements can be organized in any manner. The default case is often listed last for code readability. Remember that without `break` statements, the switch block will continue with its fall-through, from the point that the condition has been met. The following code is a valid `switch` conditional statement that uses an enumeration type for its expression value:

```
private enum ClamBait {FRESH,SALTED,ARTIFICIAL}
...
ClamBait bait = ClamBait.SALTED;
switch (bait) {
default:
   System.out.println("No bait");
   break;
  case FRESH:
   System.out.println("Fresh clams");
   break;
  case SALTED:
   System.out.println("Salted clams");
   break;
  case ARTIFICIAL:
   System.out.println("Artificial clams");
   break;
  }
```

Knowing what you can and cannot do with `switch` statements will help expedite your development efforts.

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, the `while` and the `do-while` statements. The `break` statement is used to exit the body of any

SCENARIO & SOLUTION

To ensure your statement is bug free, which type of statements should you include within the switch?	Both `break` statements and the `default` statement are commonly used in the switch. Forgetting these statements can lead to improper fall-throughs or unhandled conditions. Note that many bug-finding tools will flag missing `default` statements.
You wish to use a range in a `case` statement (for instance, `case 7-35`). Is this a valid feature in Java, as it is with other languages?	Ranges in `case` statements are *not* allowed. Consider setting up a condition in an `if` statement. For example: `if (x >=7 && x <=35){}`
You wish to use the `switch` statement, using `String` values where the expression is expected, as is possible with other languages. Is this a valid feature in Java?	Strings are not valid at the decision point for `switch` statements. Consider using an `if` statement instead. For example: `if (strValue.equals("S1")){}`

iteration statement. The `continue` statement is used to terminate the current iteration and continue with the next iteration. The iteration statements detailed in Table 2-4 will be seen on the exam.

TABLE 2-4 Iteration Statements

Formal Name	Keywords	Main Expression Components	Example
`for` loop	`for`, `break` (optional), `continue` (optional)	Initializer, expression, update mechanism	`for (i=0; i<j; i++) {}`
Enhanced `for` loop	`for`, `break` (optional), `continue` (optional)	Element, array, or collection	`for (Fish f : listOfFish) {};`

`while`	`while`, `break` (optional), `continue` (optional)	Boolean expression	`while (value == 1) {`
			 `}`

`do-while`	`do`, `while`, `break` (optional), `continue` (optional)	Boolean expression	`do {`
			 `} while (value == 1);`

The `for` Loop Iteration Statement

The `for` loop statement is designed to iterate through code. 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. So, "`int x = 0;`" and "`x=0;`" are both acceptable in the initialization part. 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. A basic update part is represented as "`i++;`".

The general usage of the `for` statement:

```
for ( initialization; expression; iteration) {
  // Sequence of statements
}
```

The following is an example of a basic `for` loop where the initialization variable is declared outside the `for` loop statement:

```
int m;
for (m = 1; m < 5; m++) {
  System.out.print("Marker " + m + ", ");
}
System.out.print("Last Marker " + m + "\n");
$ Marker 0, Marker 1, Marker 2, Marker 3, Marker 4, Last Marker 5
```

The following is a similar example, but with the variable declared in the `for` loop:

```
for (int m = 1; m < 5; m++) {
  System.out.print("Marker " + m + ", ");
}
```

Declaring the initialize variable in the `for` loop is allowed and is the common approach. However, you can't use the variable once you have exited the loop. The following will result in a compilation error:

```
for (int m = 1; m < 5; m++) {
  System.out.print("Marker " + m + ", ");
}
System.out.print("Last Marker " + m + "\n"); // m is out of scope
# Error: variable m not found in class [ClassName].
```

INSIDE THE EXAM

Exposing Corner Cases with Your Compiler

The exam designers were not satisfied with just validating your knowledge of the fundamental Java material. They took the time to work in corner cases as well as modify the structure of the code in such a slight manner that it appears to be correct but is not. When you work through the examples in this book, take the time to modify things a bit, intentionally introducing errors, to see how the compiler reacts. Being able to think like the compiler will help you score higher on the exam.

Third-party developers of Java development kits can define their own text for compiler error messages. Where they will likely try to model the messages provided by Sun's JDK, sometimes care will be taken to make the messages more precise. Consider invoking compiler errors with the latest Sun JDK compiler, as well as a compiler providing an IDE such as the Eclipse SDK. Compare the similarities and differences.

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.

The general usage of the `for` statement:

for (type variable : collection) statement-sequence

The following code segment demonstrates how a `for` loop can easily dump out the contents of an array. Here, the enhanced `for` loop iterates over each `hook` integer in the array `hookSizes`. For each iteration, the hook size is printed out.

```
int hookSizes[] = { 1, 1, 1, 2, 2, 4, 5, 5, 5, 6, 7, 8, 8, 9 };
for (int hook: hookSizes) System.out.print(hook + " ");
$ 1 1 1 2 2 4 5 5 5 6 7 8 8 9
```

The enhanced `for` loop is frequently used for searching through items in a collection. Here, the enhanced `for` loop iterates over each `hook Integer` in the collection `hookSizesList`. For each iteration, the `hook` size is printed out. This example demonstrates the use of collections and generics.

```
Integer hookSizeList;
ArrayList<Integer> hookSizesList = new ArrayList<Integer>();
hookSizesList.add(1);
hookSizesList.add(4);
hookSizesList.add(5);
for (Integer hook : hookSizesList) System.out.print(hook + " ");
$ 1 4 5
```

See *Java Generics and Collections* by Maurice Naftalin and Philip Wadler (O'Reilly, October 2006) for comprehensive coverage of the Generics and Collections frameworks.

EXERCISE 2-1

Iterating Through an `ArrayList` While Applying Conditions

This exercise will have you iterating through an `ArrayList` of floats. Specifically, this exercise will have you printing out only the legal sizes of keeper fish.

1. Create an `ArrayList` of floats called `fishLengthList`. This list will represent the sizes of a few striped bass.
2. Add the following floats to the list: 10.0, 15.5, 18.0, 29.5, 45.5. These numbers represent the length in inches of the bass.
3. Iterate through the list, printing out only the numbers larger than the required length. Assume the required length is 28 inches.

The `while` Iteration Statement

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

The general usage of the `while` statement:

```
while (expression) {
  // Sequences of statements
}
```

The following code example demonstrates the use of the `while` statement. Here, a fisherman will continue fishing until his fish limit has been reached. Specifically, when the `fishLimit` variable within the body of the `while` statement reaches `10`, the fisherman's `session` will be set to inactive. Since the `while` statement demands that the `session` be active, its loop will terminate upon the change.

```
fishingSession.setSession("active");
/* WHILE STATEMENT */
while (fishingSession.getSession().equals("active")) {
 castForFish(); // Updates fishLimit instance variable
 if (fishLimit == 10) {
   fishingSession.setSession("inactive");
 }
}
```

on the job

Various formatting styles can be followed when formatting your code. Formatting considerations include indentation, white space usage, line wrapping, code separation, and braces handling. You should select a style and maintain it throughout your code. For demonstration purposes, here are two distinct ways that braces are handled:

K&R style braces handling:

```
while (x==y) {
 performSomeMethod();
}
```

Allman style brace handling:

```
while (x==y)
{
 performSomeMethod();
}
```

Most IDEs support customizable formatting that can often be applied by selecting a format option from a menu. Using an IDE to ensure formatting is properly and consistently applied is a good idea. A popular Java code beautifier that is available as a plug-in to many tools is Jalopy: http://jalopy.sourceforge.net/jalopy/manual.html.

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 only continues to execute the body if it evaluates to true. There is typically an expression within the body that will affect the result of the expression.

The general usage of the `do-while` statement:

```
do {
  // Sequence of statements
} while (expression)
```

EXERCISE 2-2

Performing Code Refactoring

In the following code example, we want to make sure the fisherman gets at least one cast in. While this appears to make logical sense, you always need to think about corner cases. What if a fox steals the fisherman's bait before he gets a chance to cast? In this case, the `piecesOfBait` variable would equal zero, but the fisherman would still cast as the body of the `do-while` loop is guaranteed at least one iteration. See if you can refactor this code with a `while` statement to avoid the possible condition of casting with no bait.

```
fishingSession.setSession("active");
int piecesOfBait = 5;
piecesOfBait = 0; // Fox steals the bait!
 do {
  castForFish();
  /* Check to see if bait is available */
  if (fishingSession.isBaitAvailable() == false) {
    /* Place a new piece of bait on the hook */
    fishingSession.setBaitAvailable(true);
    piecesOfBait--;
  }
 } while (piecesOfBait != 0);
```

SCENARIO & SOLUTION	
You wish to iterate though a collection. Which iteration statement would be the best choice?	You will need to use the enhanced `for` loop statement.
You wish to execute a statement based on the result of a boolean expression. Which conditional statement would be the best choice?	You will need to use the `if` statement.
You wish to provide conditional cases in relationship to enumeration values. What conditional statement would be your only choice?	You will need to use the `switch` statement.
You wish to execute a block of statements and then iterate through the block based on a condition. What iteration statement would be your only choice?	You will need to use the `do-while` statement.
You wish to permanently exit a case statement. What transfer of control statement would you choose?	You will need to use the `break` statement.

Selecting the right statement types during development can make coding your algorithms easier. Proper statement selection will also promote the ease of software maintenance efforts if the code ever needs to be modified. It's important to realize that statements are used for different purposes and one particular type of statement cannot solve all development needs. You will find it not uncommon to use a combination of statement types to implement the code for many algorithms. Having a strong foundation of what the main purposes are of the different types of statements will assist you when you need to use them together.

CERTIFICATION OBJECTIVE

Implementing Statement-Related Algorithms from Pseudo-code

Exam Objective 4.3 Given an algorithm as pseudo-code, develop method code that implements the algorithm using conditional statements (if and switch), iteration statements (for, for-each, while, and do-while), assignment statements, and break and continue statements to control the flow within switch and iteration statements.

Pseudo-code is a structured means to allow algorithm designers to express computer programming algorithms in a human-readable format. Pseudo-code is informally written and has the characteristics of being very compact and high-level in nature. Even though pseudo-code does not need to be tied to any specific software language, the designer will typically script the pseudo-code algorithms based on the structural conventions of their target software language. You may be thinking, "Hey, pseudo-code sound great! Where do I get started writing high-quality algorithms in pseudo-code!" Well, don't get too excited. No standards exist for writing pseudo-code, since its main purpose is to help designers build algorithms in their own language. With so many different languages having varying structural differences and paradigms, creating a pseudo-code standard that applies to them all would be impossible. Essentially, writing pseudo-code allows for the quick and focused production of algorithms based on logic, not language syntax. The following topics presented in the next sections will discuss working with basic pseudo-code and converting pseudo-code algorithms into Java code with an emphasis on statements:

- Pseudo-code algorithms
- Pseudo-code algorithms and Java

Pseudo-code Algorithms

The exam will present pseudo-code algorithms to you. In turn, you will have options to decide which Java code segment correctly implements the algorithms. This can be tricky since the pseudo-code algorithms do not need to represent Java syntax in any way, but the Java code segments must be structurally and syntactically accurate to be correct.

Let's take a look at a pseudo-code algorithm.

```
value := 20
IF value >= 1
  print the value
ELSEIF value = 0
  print the value
ELSE
  print "less than zero"
ENDIF
```

When converting this algorithm to Java, you may naturally want to use "ELSEIF", "ELSE", and "ENDIF". This is a gotcha… because they are not keywords. Let's do an exercise to review valid keywords that may be seen on the exam.

EXERCISE 2-3

Knowing Your Statement-Related Keywords

Table 2-5 represents all of the valid Java keywords. This exercise will allow you to use the table to assist you in deducing the keywords you may see while using the various types of statements.

On a tangent, the following bullets are included here to remove confusion about some of the keywords:

- Keywords `const` and `goto` are reserved Java keywords but are not functionally used. Since they are commonly used C++ keywords, the Java language designers felt that providing them as keywords would allow the IDEs and compilers to provide better error messages when these keywords are encountered.
- The `assert` keyword was added with J2SE 1.4.
- The `enum` keyword was added with J2SE 5.0.
- Reserved literals named `true`, `false`, and `null` are not keywords.
- Java keywords cannot be used as identifiers.

TABLE 2-5

Java EE 5 Keywords

Java Keywords				
abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Let's start the exercise.

1. List the primary keywords you may see in conditional statements.
2. List the primary keywords you may see in iteration statements.
3. List the primary keywords you may see in transfer of control statements.
4. Bonus: List the primary keywords you may see in exception handling statements.

While surfing the Internet, you'll find there is no universally accepted convention for pseudo-code. However, Table 2-6 gives you a general idea as to what you may expect on the exam. This information was reduced from the

TABLE 2-6 Pseudo-code Conventions

Pseudo-code Element	Pseudo-code Convention	Java Example
Assignment	`variable := value`	`wreckYear = 1511;`
if statement	`IF condition THEN`	`if (wreckYear == 1502)`
	`  //statement sequence`	`  wreck = "Santa Ana";`
	`ELSEIF`	`elseif (wreckYear == 1503)`
	`  //statement sequence`	`  wreck = "Magdalena";`
	`ELSE`	`else`
	`  //statement sequence`	`  wreck = "Unknown";`
	`ENDIF`	
switch statement	`CASE expression OF`	`switch (wreckYear) {`
	`   Condition A: statement sequence`	`  case 1502:`
	`   Condition B:`	`    wreck = "Santa Ana";`
	`statement sequence`	`    break;`
	`   Default: sequence of statements`	`  case 1503:`
	`ENDCASE`	`    wreck = "Magdalena";`
		`    break;`
		`  default:`
		`    wreck = "Unknown"`
		`}`
while statement	`WHILE condition`	`while (n < 4) {`
	`  //statement sequence`	`   System.out.println(i);`
	`ENDWHILE`	`   n++;`
		`}`
for statement	`FOR iteration bounds`	`for (int i=0; i<j; i++) {`
	`  //statement sequence`	`   System.out.println(i);`
	`ENDFOR`	`}`

"PSEUDOCODE STANDARD" page of the Cal Poly State University web site. The authors did a pretty good job proposing a standard.

Pseudo-code Algorithms and Java

The exam will give you a piece of pseudo-code and ask you to select the correct Java source code conversion. The code will likely be a fragment of a complete source file and while it is okay that some primitive declarations may be missing, conditional and iteration statements are always represented completely. Let's take a look at an example: Given:

```
fishingRods := 5
fishingReels := 4
IF fishingRods does not equal fishingReels THEN
  print "We are missing fishing equipment"
ELSE
  print "The fishing equipment is all here"
ENDIF
```

Answer:

```
int fishingRods = 5;
int fishingReels = 4;
if (fishingRods != fishingReels)
  System.out.print("We are missing fishing equipment");
else
  System.out.print("The fishing equipment is all here");
```

exam watch

Three objectives on the exam have to do with producing Java code from pseudo-code algorithms. These objectives cover statement-related (objective 4.3), operator-related (objective 4.5), and variable scope–related (objective 4.2) pseudo-code algorithms.

The following three exercises will assist you in preparing for the pseudo-code-related questions on the exam.

EXERCISE 2-4

Implementing Pseudo-code Algorithm #1

Given:

```
location := Corson's Inlet
IF location != NULL THEN
  print "Fishing spot: " + location
ENDIF
```

1. Convert the pseudo-code to Java.
2. Compile the Java source code and debug any errors and warnings.
3. Interpret the compiled bytecode.

EXERCISE 2-5

Implementing Pseudo-code Algorithm #2

Given:

```
IF waterTemperature greater than or equal to 69 THEN
  isStripersMostActive := false // Stripers are less active
ELSEIF waterTemperature less than 69 but greater than 47 THEN
  isStripersMostActive := true // Stripers are most active
ELSEIF waterTemperature less than or equal to 47 THEN
  isStripersMostActive := false // Stripers are less active
ENDIF
```

1. Convert the pseudo-code to Java.
2. Compile the Java source code and debug any errors and warnings.
3. Interpret the compiled bytecode.

EXERCISE 2-6

Implementing Pseudo-code Algorithm #3

Given:

```
fishingList = rods, reels, bait, lunch
FOR EACH variable in fishingList
  print variable

ENDFOR
```

1. Convert the pseudo-code to Java.
2. Compile the Java source code and debug any errors and warnings.
3. Interpret the compiled bytecode.

CERTIFICATION SUMMARY

This chapter on fundamental statements discussed details related to the fundamental statement types. By studying this chapter, you should now be able to recognize and develop the following types of statements:

- Expression statements, with a focus on the assignment statement
- Conditional statements (`if`, `if-then`, `if-then-else`, and `switch`)
- Iteration statements (`for`, enhanced `for`, `while`, and `do-while`)
- Transfer of control statements (`continue`, `break`, and `return`)

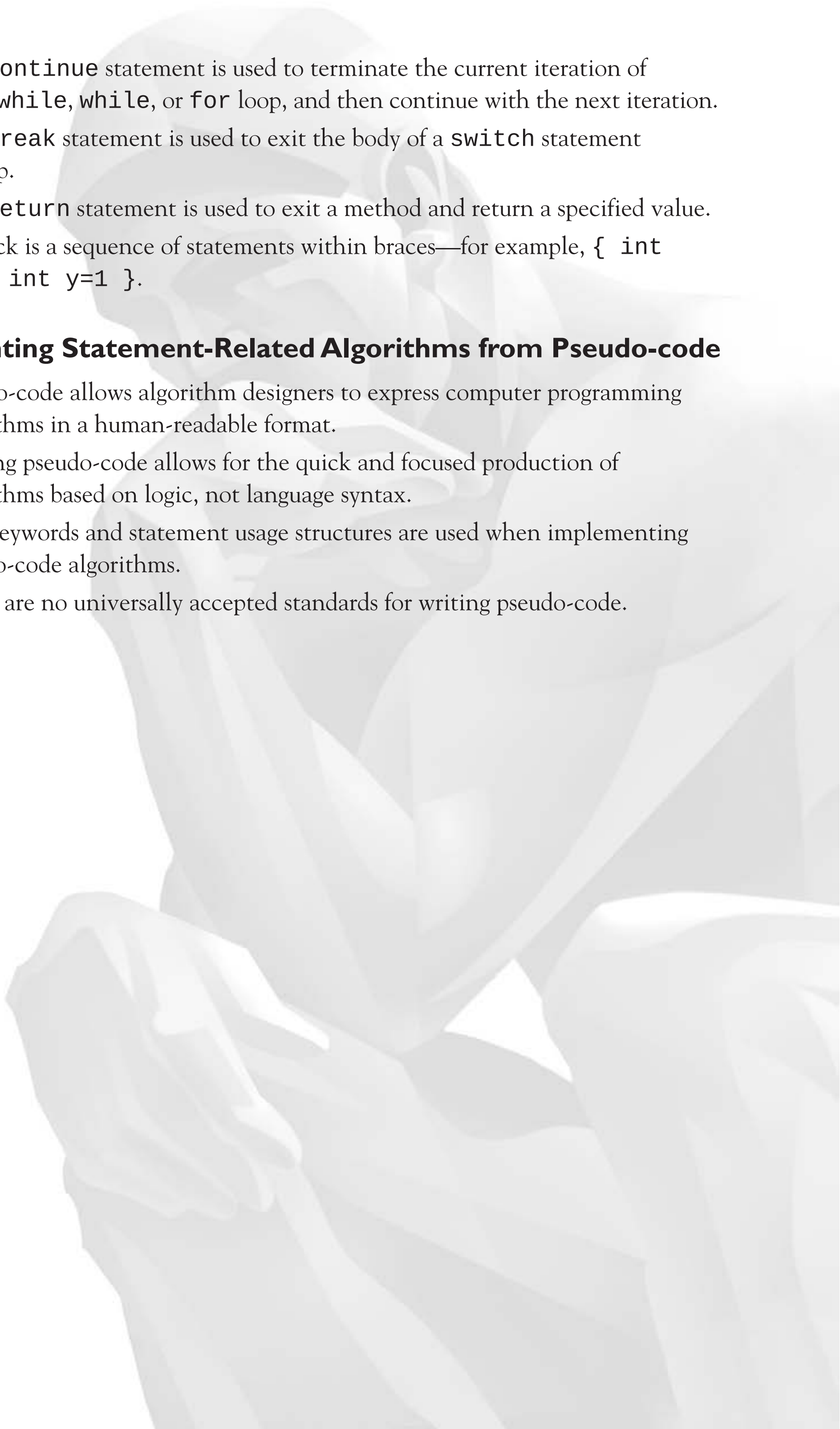
The types of pseudo-code-related questions you can expect to see on the exam were covered in this chapter. To prepare you, we explored the basics of pseudo-code in detail and discussed how to convert statement-related pseudo-code algorithms to pure Java code.

At this point, you should be well prepared for exam questions covering Java statements and relative pseudo-code being converted to Java.

✓ TWO-MINUTE DRILL

Understanding Fundamental Statements

- ❑ Assignment statements assign values to variables.
- ❑ Assignment statements that do not return `boolean` types will cause the compiler to report an error when used as the expression in an `if` statement.
- ❑ Trying to save an invalid literal to a declared primitive type variable will result in a compiler error.
- ❑ Conditional statements are used for determining the direction of flow based on conditions.
- ❑ Types of conditional statements include the `if`, `if-then`, `if-then-else`, and `switch` statements.
- ❑ The `default` case statement can be placed anywhere in the body of the `switch` statement.
- ❑ The expressions used in `if` statements must evaluate to `boolean` values, or the application will fail to compile.
- ❑ `Boolean` wrapper classes are allowed as expressions in `if` statements since they are unboxed. Remember that unboxing is the automatic production of primitive values from their related wrapper classes when the primitive value is required.
- ❑ Iteration statements are designed for iterating through pieces of code.
- ❑ Iteration statements include the `for` loop, enhanced `for` loop, and the `while` and `do-while` statements.
- ❑ The `for` loop statement has main components that include an initialization part, an expression part, and an update part.
- ❑ The enhanced `for` loop statement is used for iteration through an iterable object or array.
- ❑ The `while` loop statement is used for iteration based on a condition.
- ❑ The `do-while` statement is used for iteration based on a condition. The body of this statement is always executed at least once.
- ❑ Transfer of control statements interrupt or stop the flow of execution.
- ❑ The transfer of control statements include the `continue`, `break`, and `return` statements.



- ❑ The `continue` statement is used to terminate the current iteration of a `do-while`, `while`, or `for` loop, and then continue with the next iteration.
- ❑ The `break` statement is used to exit the body of a `switch` statement or loop.
- ❑ The `return` statement is used to exit a method and return a specified value.
- ❑ A block is a sequence of statements within braces—for example, `{ int x=0; int y=1 }`.

Implementing Statement-Related Algorithms from Pseudo-code

- ❑ Pseudo-code allows algorithm designers to express computer programming algorithms in a human-readable format.
- ❑ Writing pseudo-code allows for the quick and focused production of algorithms based on logic, not language syntax.
- ❑ Java keywords and statement usage structures are used when implementing pseudo-code algorithms.
- ❑ There are no universally accepted standards for writing pseudo-code.

SELF TEST

Understanding Fundamental Statements

1. Given x is declared with a valid integer, which conditional statement will not compile?

A. `if (x == 0) {System.out.println("True Statement");}`

B. `if (x == 0) {System.out.println("False Statement");}`

C. `if (x == 0) {;} elseif (x == 1) {System.out.println("Valid Statement");}`

D. `if (x == 0) ; else if (x == 1){} else {;}`

2. Which is not a type of statement?

A. Conditional statement

B. Assignment statement

C. Iteration statement

D. Propagation statement

3. What type of statement would be used to code the following equation: y = (m*x) + b?

A. Conditional statement

B. Assignment statement

C. Assertion statement

D. Transfer of control statement

4. You need to update a value of a hash table (that is, HashMap) where the primary key must equal a specified string. Which statements would you need to use in the implementation of this algorithm?

A. Iteration statement

B. Expression statement

C. Conditional statement

D. Transfer of control statement

5. Which keyword is part of a transfer of control statement?

A. `if`

B. `return`

C. `do`

D. `assert`

6. A `switch` statement works with which wrapper class/reference type?

A. `Character`

B. `Byte`

C. `Short`

D. `Int`

7. Which statements correctly declare `boolean` variables?

A. `Boolean isValid = true;`

B. `boolean isValid = TRUE;`

C. `boolean isValid = new Boolean (true);`

D. `boolean isValid = 1;`

8. Which of the following statements will not compile?

A. `if (true) ;`

B. `if (true) {}`

C. `if (true) {:}`

D. `if (true) {;}`

9. Given:

```
public class Dinner {
  public static void main (String[] args)
  {
    boolean isKeeperFish = false;
    if (isKeeperFish = true) {
      System.out.println("Fish for dinner");
    } else {
      System.out.println("Take out for dinner");
    }
  }
}
```

What will be the result of the application's execution?

A. `Fish for dinner` will be printed.

B. `Take out for dinner` will be printed.

C. A compilation error will occur.

10. The `for` loop has been enhanced in Java 5.0. Which is NOT a common term for the improved *for* loop.

A. The "for in" loop

B. The specialized *for* loop

C. The "*for each*" loop

D. The enhanced `for` loop

Implementing Statement-Related Algorithms from Pseudo-code

11. Given:

```
COUNTER := 1
WHILE COUNTER LESS THAN 10
  PRINT COUNTER AND A NEW LINE
  COUNTER := COUNTER + 1
ENDWHILE
```

Which Java code segment implements the pseudo-code algorithm?

A.
```
INT counter = 1;
WHILE (counter < 10) {
  System.out.print(counter + "\n");
  counter++;
}
```

B.
```
int counter = 1;
while {counter < 10} {
  System.out.print(counter + "\n");
  counter++;
}
```

C.
```
int counter = 1;
while (counter < 10) {
  System.out.println(counter);
  counter++;
}
```

D.
```
int counter = 1;
while (counter < 10) {
  System.out.println(counter + "\n");
  counter++;
}
```

12. Given:

```
ISRECORD := FALSE
FLOAT RECORD := 78.8
IF WEIGHT > RECORD
THEN ISRECORD := TRUE
ELSE ISRECORD := FALSE
ENDIF
```

Which Java code segment implements the pseudo-code algorithm?

A.
```
boolean isRecord = false
float record = 78.8f
if (weight > record) isRecord = true
else isRecord = false
```

B.
```
boolean isRecord = false;
float record = 78.8f;
if (weight > record) isRecord = true;
else isRecord = false;
endif
```

C.
```
boolean isRecord = FALSE;
float record = 78.8f;
if (weight > record) isRecord = true;
else isRecord = false;
```

D.
```
boolean isRecord = false;
float record = 78.8f;
if (weight > record) isRecord = true;
else isRecord = false;
```

SELF TEST ANSWERS

Understanding Fundamental Statements

1. Given x is declared with a valid integer, which conditional statement will not compile?

A. `if (x == 0) {System.out.println("True Statement");}`

B. `if (x == 0) {System.out.println("False Statement");}`

C. `if (x == 0) {;} elseif (x == 1) {System.out.println("Valid Statement");}`

D. `if (x == 0) ; else if (x == 1){} else {;}`

Answer:

☑ **C.** The statement will not compile. Without a space between the `else` and `if` keywords, the compiler will be thrown an error similar to "Error: method elseif (boolean) not found…"

☒ **A, B,**and **D** are incorrect. All of these conditional statements will compile successfully.

2. Which is not a type of statement?

A. Conditional statement

B. Assignment statement

C. Iteration statement

D. Propagation statement

Answer:

☑ **D.** There is no such thing as a propagation statement.

☒ **A, B,**and **C** are incorrect. Conditional, assignment, and iteration are all types of statements.

3. What type of statement would be used to code the following equation: y = (m*x) + b?

A. Conditional statement

B. Assignment statement

C. Assertion statement

D. Transfer of control statement

Answer:

☑ **B.** An assignment statement would be used to code the given example of y = (m*x) + b.

☒ **A, C,**and **D** are incorrect. The conditional, assertion, and transfer of control statements are not used to perform assignments.

4. You need to update a value of a hash table (that is, HashMap) where the primary key must equal a specified string. Which statements would you need to use in the implementation of this algorithm?

A. Iteration statement

B. Expression statement

C. Conditional statement

D. Transfer of control statement

Answer:

☑ **A, B,**and **C.** An Iteration, expression, and conditional statements would be used to implement the algorithm. The following code segment demonstrates the use of these statements by programmatically replacing the ring on the little finger of a person's left hand. The statements are prefaced by comments that identify their types.

```
import java.util.HashMap;
public class HashMapExample {
  public static void main(String[] args) {
    HashMap<String,String> leftHand = new HashMap<String,String>();
    leftHand.put("Thumb", null);
    leftHand.put("Index finger", "Puzzle Ring");
    leftHand.put("Middle finger", null);
    leftHand.put("Ring finger", "Engagement Ring");
    leftHand.put("Little finger", "Pinky Ring");
    // Iteration statement
    for (String s : leftHand.keySet()) {
      // Conditional statement
      if (s.equals("Little finger")) {
        System.out.println(s + " had a " + leftHand.get(s));
        // Expression Statement
        leftHand.put("Little finger", "Engineer's Ring");
```

```
            System.out.println(s + " has an " + leftHand.get(s));
          }
        }
      }
    }
    $ Little finger had a Pinky Ring
    $ Little finger has an Engineer's Ring
```

☒ **D** is incorrect. There is no transfer of control statement in the algorithm.

5. Which keyword is part of a transfer of control statement?

A. `if`

B. `return`

C. `do`

D. `assert`

Answer:

☑ **B.** The keyword `return` is used as part of a transfer of control statement.

☒ **A, C,**and **D** are incorrect. The keywords `if`, `do`, and `assert` are not part of any transfer of control statements.

6. A `switch` statement works with which wrapper class/reference type?

A. `Character`

B. `Byte`

C. `Short`

D. `Int`

Answer:

☑ **A, B,**and **C.** The `switch` statements work with `Character`, `Byte`, and `Short` wrapper classes as well as the `Integer` wrapper class.

☒ **D** is incorrect. There is no such thing as an `Int` wrapper type. This was a trick question. The `switch` statement works with either the `int` primitive or the `Integer` wrapper type.

7. Which statements correctly declare `boolean` variables?

A. `Boolean isValid = true;`

B. `boolean isValid = TRUE;`

C. `boolean isValid = new Boolean (true);`

D. `boolean isValid = 1;`

Answer:

☑ **A** and **C.** These statements properly declare `boolean` variables. Remember, the only valid literal values for the `boolean` primitives are `true` and `false`.

☒ **B** and **D** are incorrect. **B** is incorrect because TRUE is not a valid literal value. **D** is incorrect because you cannot assign the value `1` to a `boolean` variable.

8. Which of the following statements will not compile?

A. `if (true) ;`

B. `if (true) {}`

C. `if (true) {:}`

D. `if (true) {;}`

Answer:

☑ **C.** A colon is invalid by itself.

☒ **A, B,** and **D** are incorrect. All of the statements represent compilable code.

9. Given:

```
public class Dinner {
  public static void main (String[] args)
  {
    boolean isKeeperFish = false;
    if (isKeeperFish = true) {
      System.out.println("Fish for dinner");
    } else {
      System.out.println("Take out for dinner");
    }
  }
}
```

What will be the result of the application's execution?

A. `Fish for dinner` will be printed.

B. `Take out for dinner` will be printed.

C. A compilation error will occur.

Answer:

☑ **A.** Since only one equals sign (that is, assignment statement) was used in the `if` statement, the `isKeeperFish` variable was assigned the value of `true`.

☒ **B** and **C** are incorrect.

10. The `for` loop has been enhanced in Java 5.0. Which is NOT a common term for the improved for loop.

A. The "for in" loop

B. The specialized for loop

C. The "for each" loop

D. The enhanced `for` loop

Answer:

☑ **B.** The enhanced `for` loop is not commonly referenced as a specialized *for* loop.

☒ **A, C,**and **D** are incorrect. The enhanced `for` loop is also commonly referenced as the *for in* loop and the *for each* loop.

Implementing Statement-Related Algorithms from Pseudo-code

11. Given:

```
COUNTER := 1
WHILE COUNTER LESS THAN 10
  PRINT COUNTER AND A NEW LINE
  COUNTER := COUNTER + 1
ENDWHILE
```

Which Java code segment implements the pseudo-code algorithm?

A.
```
INT counter = 1;
WHILE (counter < 10) {
  System.out.print(counter + "\n");
  counter++;
}
```

B.
```
int counter = 1;
while {counter < 10} {
  System.out.print(counter + "\n");
  counter++;
}
```

C.
```
int counter = 1;
while (counter < 10) {
  System.out.println(counter);
  counter++;
}
```

D.
```
int counter = 1;
while (counter < 10) {
  System.out.println(counter + "\n");
  counter++;
}
```

Answer:

☑ **C.**The answer implements the pseudo-code algorithm correctly.

☒ **A, B,**and **D** are incorrect. Answer **A** is incorrect because INT and WHILE are not Java keywords. **B** is incorrect because the expression for the `while` statement is enclosed in braces where parentheses are expected. **D** is incorrect because two new lines are printed, one with (\n) and one with the `println` method.

12. Given:

```
ISRECORD := FALSE
FLOAT RECORD := 78.8
IF WEIGHT > RECORD
THEN ISRECORD := TRUE
ELSE ISRECORD := FALSE
ENDIF
```

Which Java code segment implements the pseudo-code algorithm?

A.
```
boolean isRecord = false
float record = 78.8f
if (weight > record) isRecord = true
else isRecord = false
```

B.
```
boolean isRecord = false;
float record = 78.8f;
if (weight > record) isRecord = true;
else isRecord = false;
endif
```

C.
```
boolean isRecord = FALSE;
float record = 78.8f;
if (weight > record) isRecord = true;
else isRecord = false;
```

D.
```
boolean isRecord = false;
float record = 78.8f;
if (weight > record) isRecord = true;
else isRecord = false;
```

Answer:

☑ **D.** The answer implements the pseudo-code algorithm correctly.

☒ **A, B,**and **C** are incorrect. **A** is incorrect because all of the required semicolons are missing. **B** is incorrect because the answer uses `endif`, which is not a Java keyword. Answer **C** is incorrect because the answer incorrectly uses FALSE. The only valid Java boolean literals are `true` and `false`.