ANSWER KEY

1. B
2. E
3. D
4. E
5. E
6. C
7. B
8. D
9. D
10. A
11. C
12. D
13. A
14. C
15. A
16. D
17. A
18. C
19. D
20. D
21. B
22. C
23. B
24. E
25. D
26. D

ANSWERS EXPLAINED

1. (B) When x is converted to an integer, as in segment I, information is lost. Java requires that an explicit cast to an `int` be made, as in segment II. Note that segment II will cause x to be truncated: The value stored in y is 14. By requiring the explicit cast, Java doesn't let you do this accidentally. In segment III y will contain the value `14.0`. No explicit cast to a `double` is required since no information is lost.

2. (E) The string argument contains two escape sequences: '\\', which means print a backslash (\), and '\n', which means go to a new line. Choice E is the only choice that does both of these.

3. (D) Short-circuit evaluation of the boolean expression will occur. The expression `(n != 0)` will evaluate to `false`, which makes the entire boolean expression `false`. Therefore the expression `(x / n > 100)` will not be evaluated. Hence no division by zero will occur, causing an `ArithmeticException` to be thrown. When the boolean expression has a value of `false`, only the `else` part of the statement, *statement2*, will be executed.

4. (E) For this choice, the integer division `13/5` will be evaluated to 2, which will then be cast to `2.0`. The output will be `13/5 = 2.0`. The compiler needs a way to recognize that real-valued division is required. All the other options provide a way.

5. (E) The operators *, /, and % have equal precedence, all higher than -, and must be performed first, from left to right.

```
   13 - 3 * 6 / 4 % 3
=  13 - 18 / 4 % 3
=  13 - 4 % 3
=  13 - 1
=  12
```

6. (C) The expression must be evaluated as if parenthesized like this:

13. (A) If `(c < a)` is `false`, `((c == a*b) && (c < a))` evaluates to `false` irrespective of the value of `c == a*b`. In this case, `!(c == a*b && c < a)` evaluates to `true`. Then `(a < b) || true` evaluates to `true` irrespective of the value of the test `(a < b)`. In all the other choices, the given expression *may* be `true`. There is not enough information given to guarantee this, however.

14. (C) If `a == b` is false, then `a != b` is true. Thus, the second piece of the compound test must be evaluated before the value of the whole test is known. Since `a == b` is false, `a - b` is not equal to zero. Thus, there is no division by zero, and no exception will be thrown. Also, since the relative values of `a`, `b`, and `n` are unknown, the value of the test `n / (a - b) > 90` is unknown, and there is insufficient information to determine whether the compound test is true or false. Thus, either */* statement 1 */* or */* statement 2 */* will be executed.

15. (A) If $n \geq 1$, both segments will print out the integers from 1 through `n`. If $n \leq 0$, both segments will fail the test immediately and do nothing.

16. (D) The `(value != SENTINEL)` test occurs before a value has been read from the list. This will cause 0 to be processed, which may cause an error. The code must be fixed by reading the first value before doing the test:

```
final int SENTINEL = -999;
int value = IO.readInt();
while (value != SENTINEL)
{
    //code to process value
    value = IO.readInt();
}
```

17. (A) Quick method: Convert each hex digit to binary.

$$\begin{aligned} & 3 \quad D_{hex} \\ &= 0011 \quad 1101 \quad \text{(where D equals 13 in base 10)} \\ &= 111101_{bin} \end{aligned}$$

Slow method: Convert $3D_{hex}$ to base 10.

$$\begin{aligned} 3D_{hex} &= (3)(16^1) + (D)(16^0) \\ &= 48 + 13 \\ &= 61_{dec} \end{aligned}$$

Now convert 61_{dec} to binary. Write 61 as a sum of descending powers of 2:

$$\begin{aligned} 61 &= 32 + 16 + 8 + 4 + 1 \\ &= 1(2^5) + 1(2^4) + 1(2^3) + 1(2^2) + 0(2^1) + 1(2^0) \\ &= 111101_{bin} \end{aligned}$$

18. (C) Start by converting each of the three numbers to hexadecimal:

$$\begin{aligned} 14 &= (0)(16^1) + (14)(16^0) &&= 0E \\ 20 &= (1)(16^1) + (4)(16^0) &&= 14 \\ 255 &= (15)(16^1) + (15)(16^0) &&= FF \end{aligned}$$

Therefore $(14, 20, 255) = \#0E14FF$.