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. **D**
18. **B**
19. **C**
20. **C**
21. **B**
22. **E**
23. **D**
24. **D**

Answer Explanations

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, which would have caused 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:

```
(2 + 3) * 12 / (7 - 4 + 8)
```

This becomes `5 * 12 / 11 = 60 / 11 = 5`.