Operators

For more details on SUN Certifications, visit JavaScjpDumps

Q: 01 Given:
11. public class Test {
12. public static void main(String [] args) {
13. int x = 5;
14. boolean b1 = true;
15. boolean b2 = false;
16.
17. if ((x == 4) && !b2)
18. System.out.print("1 ");
19. System.out.print("2 ");
20. if ((b2 = true) && b1)
21. System.out.print("3 ");
22. }
23. }
What is the result?
A. 2
B. 3
C. 1 2
D. 2 3
E. 1 2 3
F. Compilation fails.
G. An exception is thrown at runtime.
Answer: D

Q: 02 Given the command line java Pass2 and:
15. public class Pass2 {
16. public void main(String [] args) {
17. int x = 6;
18. Pass2 p = new Pass2();
19. p.doStuff(x);
20. System.out.print(" main x = " + x);
21. }
22.
23. void doStuff(int x) {
24. System.out.print(" doStuff x = " + x++);
25. }
26. }
What is the result?

A. Compilation fails.
B. An exception is thrown at runtime.
C. doStuff x = 6 main x = 6
D. doStuff x = 6 main x = 7
E. doStuff x = 7 main x = 6
F. doStuff x = 7 main x = 7

Answer: B

Q: 03 Given:

```
13. public class Pass {
14. public static void main(String [] args) {
15. int x = 5;
16. Pass p = new Pass();
17. p.doStuff(x);
18. System.out.print(" main x = " + x);
19. }
20.
21. void doStuff(int x) {
22. System.out.print(" doStuff x = " + x++);
23. }
24. }
```

What is the result?

A. Compilation fails.
B. An exception is thrown at runtime.
C. doStuff x = 6 main x = 6
D. doStuff x = 5 main x = 5
E. doStuff x = 5 main x = 6
F. doStuff x = 6 main x = 5

Answer: D

Question: 04

Given:

```
42. public class ClassA {
43. public int getValue() {
44.int value=0;
45. boolean setting = true;
46. String title=”Hello”;
47. if (value || (setting && title == “Hello”)) { return 1; }
48. if (value == 1 & title.equals(”Hello”)) { return 2; }
49. }
50. }
```

And:

```
70. ClassA a = new ClassA();
```

71. a.getValue();
What is the result?
A. 1
B. 2
C. Compilation fails.
D. The code runs with no output.
E. An exception is thrown at runtime.
Answer: C

5. Given:
class Hexy {
public static void main(String[] args) {
Integer i = 42;
String s = (i<40)?"life":(i>50)?"universe":"everything";
System.out.println(s);
} }
What is the result?
A. null
B. life
C. universe
D. everything
E. Compilation fails.
F. An exception is thrown at runtime.
Answer:
-> **D** is correct. This is a ternary nested in a ternary with a little unboxing thrown in. Both of the ternary expressions are false.
-> A, B, C, E, and **F** are incorrect based on the above.

6. Given:
1. class Example {
2. public static void main(String[] args) {
3. Short s = 15;
4. Boolean b;
5. // insert code here
6. }
7. }
Which, inserted independently at line 5, will compile? (Choose all that apply.)
A. b = (Number instanceof s);
B. b = (s instanceof Short);
C. b = s.instanceof(Short);
D. b = (s instanceof Number);
E. b = s.instanceof(Object);
F. b = (s instanceof String);

Answer:
-> **B** and **D** correctly use boxing and instanceof together.
-> **A** is incorrect because the operands are reversed. **C** and **E** use incorrect instance of syntax. **F** is wrong because Short isn't in the same inheritance tree as String.

7. Given:

```
1. class Comp2 {
2. public static void main(String[] args) {
3. float f1 = 2.3f;
4. float[][] f2 = {{42.0f}, {1.7f, 2.3f}, {2.6f, 2.7f}};
5. float[] f3 = {2.7f};
6. Long x = 42L;
7. // insert code here
8. System.out.println("true");
9. }
10. }
```

And the following five code fragments:

```
F1. if(f1 == f2)
F2. if(f1 == f2[2][1])
F3. if(x == f2[0][0])
F4. if(f1 == f2[1,1])
F5. if(f3 == f2[2])
```

What is true?
A. One of them will compile, only one will be true.
B. Two of them will compile, only one will be true.
C. Two of them will compile, two will be true.
D. Three of them will compile, only one will be true.
E. Three of them will compile, exactly two will be true.
F. Three of them will compile, exactly three will be true.
Answer:
-> **D** is correct. Fragments F2, F3, and F5 will compile, and only F3 is true.
-> A, B, C, E, and **F** are incorrect. F1 is incorrect because you can't compare a primitive to an array. F4 is incorrect syntax to access an element of a two-dimensional array.

8. Given:

```
class Fork {
public static void main(String[] args) {
if(args.length == 1 | args[1].equals("test")) {
System.out.println("test case");
} else {
System.out.println("production " + args[0]);
} } }
```

And the command-line invocation:

java Fork live2
What is the result?
A. test case
B. production
C. test case live2
D. Compilation fails.
E. An exception is thrown at runtime.
Answer:
-> **E** is correct. Because the short circuit (||) is not used, both operands are evaluated. Since args[1] is past the args array bounds, an ArrayIndexOutOfBoundsException is thrown.
-> **A, B, C,** and **D** are incorrect based on the above.

9. Given:

```
class Foozit {
public static void main(String[] args) {
Integer x = 0;
Integer y = 0;
for(Short z = 0; z < 5; z++)
if((++x > 2) || (++y > 2))
x++;
System.out.println(x + " " + y);
} }
```

What is the result?
A. 5 1 B. 5 2 C. 5 3 D. 8 1
E. 8 2 F. 8 3 G. 10 2 H. 10 3
Answer:
-> **E** is correct. The first two times the if test runs, both x and y are incremented once (the x++ is not reached until the third iteration). Starting with the third iteration of the loop, y is never touched again, because of the short-circuit operator.
-> **A, B, C, D, F, G,** and **H** are incorrect based on the above.

10. Given:

```
class Titanic {
public static void main(String[] args) {
Boolean b1 = true;
boolean b2 = false;
boolean b3 = true;
if((b1 & b2) | (b2 & b3) & b3)
System.out.print("alpha ");
if((b1 = false) | (b1 & b3) | (b1 | b2))
System.out.print("beta ");
} }
```

What is the result?

A. beta
B. alpha
C. alpha beta
D. Compilation fails.
E. No output is produced.
F. An exception is thrown at runtime.
Answer:
-> **E** is correct. In the second if test, the leftmost expression is an assignment, not a comparison. Once b1 has been set to false, the remaining tests are all false.
-> **A, B, C, D,** and **F** are incorrect based on the above.

11. Given:

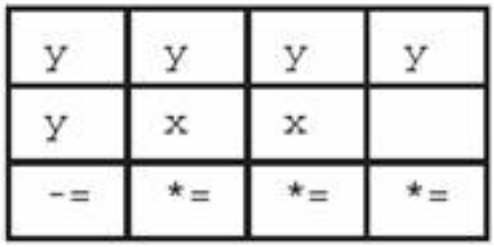
```
class Feline {
public static void main(String[] args) {
Long x = 42L;
Long y = 44L;
System.out.print(" " + 7 + 2 + " ");
System.out.print(foo() + x + 5 + " ");
System.out.println(x + y + foo());
}
static String foo() { return "foo"; }
}
```

What is the result?
A. 9 foo47 86foo
B. 9 foo47 4244foo
C. 9 foo425 86foo
D. 9 foo425 4244foo
E. 72 foo47 86foo
F. 72 foo47 4244foo
G. 72 foo425 86foo
H. 72 foo425 4244foo
I. Compilation fails.
Answer:
-> **G** is correct. Concatenation runs from left to right, and if either operand is a String, the operands are concatenated. If both operands are numbers they are added together. Unboxing works in conjunction with concatenation.
-> **A, B, C, D, E, F, H,** and **I** are incorrect based on the above.

12. Place the fragments into the code to produce the output 33. Note, you must use each fragment exactly once.
CODE:

```
class Incr {
public static void main(String[] args) {
```

Integer x = 7;
int y = 2;
x ___ ___;
___ ___ ___;
___ ___ ___;
___ ___ ___;
System.out.println(x);
}
}
FRAGMENTS:



Answer:
class Incr {
public static void main(String[] args) {
Integer x = 7;
int y = 2;
x *= x;
y *= y;
y *= y;
x -= y;
System.out.println(x);
}
}
Yeah, we know it's kind of puzzle-y, but you might encounter something like it on the real exam.

13. Given:
1. class Maybe {
2. public static void main(String[] args) {
3. boolean b1 = true;
4. boolean b2 = false;
5. System.out.print(!false ^ false);
6. System.out.print(" " + (!b1 & (b2 = true)));
7. System.out.println(" " + (b2 ^ b1));
8. }
9. }
Which are true?

A. Line 5 produces true.
B. Line 5 produces false.
C. Line 6 produces true.
D. Line 6 produces false.
E. Line 7 produces true.
F. Line 7 produces false.
Answer:
-> **A , D,** and **F** is correct. The ^ (xor) returns true if exactly one operand is true. The ! inverts the operand's boolean value. On line 6 b2 = true is an assignment not a comparison, and it's evaluated because & does not short-circuit it.
-> **B, C,** and **E** are incorrect based on the above.

14. Given:

```
class Sixties {
public static void main(String[] args) {
int x = 5;
int y = 7;
System.out.print(((y * 2) % x));
System.out.print(" " + (y % x));
}
}
```

What is the result?
A. 1 1
B. 1 2
C. 2 1
D. 2 2
E. 4 1
F. 4 2
G. Compilation fails.
H. An exception is thrown at runtime.
Answer:
->**F** is correct. The % (remainder a.k.a. modulus) operator returns the remainder of a division operation.
->**A, B, C, D, E, G,** and **H** are incorrect based on the above.

15. Given:

```
class Scoop {
static int thrower() throws Exception { return 42; }
public static void main(String [] args) {
try {
int x = thrower();
} catch (Exception e) {
x++;
```

} finally {
System.out.println("x = " + ++x);
} } }
What is the result?
A. x = 42
B. x = 43
C. x = 44
D. Compilation fails.
E. The code runs with no output.
Answer:
->**D** is correct, the variable x is only in scope within the try code block, it's not in scope inthe catch or finally blocks.
->**A, B, C,** and **E** is are incorrect based on the above.

16. Given:
class Alien {
String invade(short ships) { return "a few"; }
String invade(short... ships) { return "many"; }
}
class Defender {
public static void main(String [] args) {
System.out.println(new Alien().invade(7));
} }
What is the result?
A. many
B. a few
C. Compilation fails.
D. The output is not predictable.
E. An exception is thrown at runtime.
Answer:
->**C** is correct, compilation fails. The var-args declaration is fine, but invade takes a short,so the argument 7 needs to be cast to a short. With the cast, the answer is **B**, 'a few'.
->**A, B, D,** and **E** are incorrect based on the above. (Objective 1.3)

17. Given:
1. class Dims {
2. public static void main(String[] args) {
3. int[][] a = {{1,2,}, {3,4}};
4. int[] b = (int[]) a[1];
5. Object o1 = a;
6. int[][] a2 = (int[][]) o1;
7. int[] b2 = (int[]) o1;
8. System.out.println(b[1]);

9. } }
What is the result?
A. 2
B. 4
C. An exception is thrown at runtime
D. Compilation fails due to an error on line 4.
E. Compilation fails due to an error on line 5.
F. Compilation fails due to an error on line 6.
G. Compilation fails due to an error on line 7.
Answer:
->**C** is correct. A ClassCastException is thrown at line 7 because o1 refers to an int[][]not an int[]. If line 7 was removed, the output would be 4.
->**A, B, D, E, F,** and **G** are incorrect based on the above. (Objective 1.3)

18. Given:
```
class Eggs {
int doX(Long x, Long y) { return 1; }
int doX(long... x) { return 2; }
int doX(Integer x, Integer y) { return 3; }
int doX(Number n, Number m) { return 4; }
public static void main(String[] args) {
new Eggs().go();
}
void go() {
short s = 7;
System.out.print(doX(s,s) + " ");
System.out.println(doX(7,7));
} }
```
What is the result?
A. 1 1
B. 2 1
C. 3 1
D. 4 1
E. 2 3
F. 3 3
G. 4 3
Answer:
-> **G** is correct. Two rules apply to the first invocation of doX(). You can't widen and then box in one step, and var-args are always chosen last. Therefore you can't widen shorts to either ints or longs, and then box them to Integers or Longs. But you can box shorts to Shorts and then widen them to Numbers, and this takes priority over using a var-args method. The
second invocation uses a simple box from int to Integer.
-> **A, B, C, D, E,** and **F** are incorrect based on the above. (Objective 3.1)

19. Given:

```
class Mixer {
Mixer() { }
Mixer(Mixer m) { m1 = m; }
Mixer m1;
public static void main(String[] args) {
Mixer m2 = new Mixer();
Mixer m3 = new Mixer(m2); m3.go();
Mixer m4 = m3.m1; m4.go();
Mixer m5 = m2.m1; m5.go();
}
void go() { System.out.print("hi "); }
}
```

What is the result?

A. hi
B. hi hi
C. hi hi hi
D. Compilation fails
E. hi, followed by an exception
F. hi hi, followed by an exception

Answer:

-> **F** is correct. The m2 object's m1 instance variable is never initialized, so when m5 tries to use it a NullPointerException is thrown.
-> **A, B, C, D,** and **E** are incorrect based on the above. (Objective 7.3)

20. Given:

```
1. class Zippy {
2. String[] x;
3. int[] a [] = {{1,2}, {1}};
4. Object c = new long[4];
5. Object[] d = x;
6. }
```

What is the result?

A. Compilation succeeds.
B. Compilation fails due only to an error on line 3.
C. Compilation fails due only to an error on line 4.
D. Compilation fails due only to an error on line 5.
E. Compilation fails due to errors on lines 3 and 5.
F. Compilation fails due to errors on lines 3, 4, and 5.

Answer:

-> **A** is correct, all of these array declarations are legal. Lines 4 and 5 demonstrate that arrays can be cast.
->**B, C, D, E,** and **F** are incorrect because this code compiles. (Objective 1.3)

21. Given:

```
class Fizz {
int x = 5;
public static void main(String[] args) {
final Fizz f1 = new Fizz();
Fizz f2 = new Fizz();
Fizz f3 = FizzSwitch(f1,f2);
System.out.println((f1 == f3) + " " + (f1.x == f3.x));
}
static Fizz FizzSwitch(Fizz x, Fizz y) {
final Fizz z = x;
z.x = 6;
return z;
} }
```

What is the result?

A. true true B. false true C. true false
D. false false E. Compilation fails. F. An exception is thrown at runtime.

Answer:

->**A** is correct. The references f1, z, and f3 all refer to the same instance of Fizz. The final modifier assures that a reference variable cannot be referred to a different object, but final doesn't keep the object's state from changing.
->**B, C, D, E,** and **F** are incorrect based on the above. (Objective 7.3)

22. Given:

```
class Knowing {
static final long tooth = 343L;
static long dolt(long tooth) {
System.out.print(++tooth + " ");
return ++tooth;
}
public static void main(String[] args) {
System.out.print(tooth + " ");
final long tooth = 340L;
new Knowing().dolt(tooth);
System.out.println(tooth);
} }
```

What is the result?

A. 343 340 340
B. 343 340 342 C. 343 341 342
D. 343 341 340 E. 343 341 343 F. Compilation fails.
G. An exception is thrown at runtime.

Answer:

-> **D** is correct. There are three different long variables named tooth. Remember that you can apply the final modifier to local variables, but in this case the 2 versions of tooth marked final are not changed. The only tooth whose value changes is the one not marked final. This program demonstrates a bad practice known as shadowing.
->**A, B, C, E, F,** and **G** are incorrect based on the above. (Objective 7.3)

23. Given:
1. class Bigger {
2. public static void main(String[] args) {
3. // insert code here
4. }
5. }
6. class Better {
7. enum Faster {Higher, Longer};
8. }
Which, inserted independently at line 3, will compile? (Choose all that apply.)
A. Faster f = Faster.Higher;
B. Faster f = Better.Faster.Higher;
C. Better.Faster f = Better.Faster.Higher;
D. Bigger.Faster f = Bigger.Faster.Higher;
E. Better.Faster f2; f2 = Better.Faster.Longer;
F. Better b; b.Faster = f3; f3 = Better.Faster.Longer;
Answer:
-> **C** and **E** are correct syntax for accessing an enum from another class.
->**A, B, D,** and **F** are incorrect syntax.