socrative

Answer Key
February 24, 2021

OCA Bonus Test 1 (pt 1 of 2) ('Copy')

1. What is the result of the following code?

```
4: String numbers = "2468";
5: int total = 0;
6: total += numbers.indexOf("6");
7: total += numbers.indexOf("9");
8: char ch = numbers.charAt(3);
9: System.out.println(total + " " + ch);
```

A 1 6

✓ **B** 1 8

C 2 6

D 2 8

E An exception is thrown.

F The code does not compile.

i Line 6 returns the number 2 since indexes are zero based. Line 7 returns –1 because indexOf() could not find a match. This makes *total* equal to 1. Line 8 returns the character at index 3, which is 8.

2. Given the following class definition, which method signature could appear in a subclass of Otter? (Choose all that apply)

```
1: public abstract class Otter {
2: protected abstract void eatFish(int count);
3: }
```

✓ **A** protected void eatFish(int count)

✓ **B** protected void eatFish()

C protected int eatFish(int count)

D void eatFish(int fishCount)

✓ **E** public void eatFish(int numberOfFish)

i Option A is correct because it is a straight override with the same access modifier. Option B is an overload, not an override, but that means it can still appear in a subclass of Otter. Option C is incorrect because the return type has changed; void and int are not covariant return types. Option D can be eliminated because the assumed default access modifier is more restrictive than the inherited protected modifier. Finally, option E is acceptable because the public modifier is less restrictive than the inherited protected modifier, so it is allowed for an override.

3. Which of the following are included as a result of this code? (Choose all that apply)

```
3: String s = "Hello";
4: String t = new String(s);
5: if ("Hello".equal(s)) System.out.println("one");
6: if (t == s) System.out.println("two");
7: if (t.equal(s)) System.out.println("three");
8: if ("Hello" == s) System.out.println("four");
9: if ("Hello" == t) System.out.println("five");
```

A one

B two

C three

D four

E five

✓ F The code does not compile.

equal() is not the method used to compare two String objects. It should be equals(). Tricky, isn't it? The exam creators like to appear to be testing you on one thing when really asking something else. This question tries to mislead you into thinking it is about String logical equality versus reference equality. If equals() were in this code, the answer would be options A, C, and D.

4. What is the output of the following program?

```
1: public class Pet {
2: public void eat() throws NullPointerException {
3: throw new NullPointerException();
4: }
5: public static void main(String[] args) {
6: Pet pet = new Pet();
7: pet.eat();
8: System.out.println("Pet just ate");
9: } }
```

A Pet just ate

B Line 2 generates a compiler error.

C Line 3 generates a compiler error.

D Line 7 generates a compiler error.

✓ E NullPointerException is thrown at runtime.

The code compiles fine. NullPointerException is a RuntimeException, which means it can, but does not have to be, handled. Invoking pet.eat() on line 7 causes an uncaught NullPointerException to be thrown at runtime, so line 8 does not execute.

5. What is the output of the following code?

```
1: interface WaterFowl {}
2: class Bird {}
3: public class Duck extends Bird implements WaterFowl {
4: public void quack() { System.out.println("quack!"); }
5: public static void main(String[] args) {
6: Object object = (Object)new Bird();
7: Duck duck = (Duck)object;
8: duck.quack();
9: }
10: }
```

- **A** quack!
- **B** No output
- **C** The code will not compile because of line 4.
- **D** The code will not compile because of line 6.
- **E** The code will not compile because of line 7.
- ✓ **F** The code compiles but throws an exception at runtime.

The code compiles without issue but throws an exception at runtime. The first thing to notice is that the cast to Object on line 6, although harmless, is not actually necessary since we are casting to a super type of the class. Alternatively, the cast on line 7 is required to go from Object to a subclass of type Duck, and the compiler allows this without issue. The code fails at runtime, though, because the object is an instance of Bird and cannot be cast to a subclass of type Duck, so option F is correct.

6. What is the result of the following code?

```
3: StringBuilder b = new StringBuilder();
4: b.append("maybe").insert(2, "sh").insert(5, "xy");
5: System.out.println(b.toString());
```

- ✓ **A** mashyxybe
- **B** mashybexy
- **C** mayshbxye
- **D** mayshxybe
- **E** An exception is thrown.
- **F** The code does not compile.

This example uses method chaining. After the call to append(), b contains "maybe". That result is passed to the first insert() call, which inserts at index 2. At this point b contains "mashybe". That result is passed to the final insert(), which inserts xy at index 4, resulting in "mashyxybe".

7. Which of the following identifiers are valid Java identifiers? (Choose all that apply)

✓ **A** _x

B if

C static

D com.my.company

✓ **E** Abstract

F 2nd_place

i Option A is valid because you can use the underscore character in identifiers. Options B and C are Java keywords and are not valid identifiers. Option D is not valid because the dot (.) is not allowed in identifiers. Option E is valid because Java is case sensitive, so Abstract is not a reserved word and therefore a valid identifier. Option F is not valid because the first character is not a letter.

8. What is the result of the following code snippet?

```
3: int x = 10, y = 3;
4: if(x % y == 2)
5: System.out.print("two");
6: System.out.print(x%y);
7: if(x % y == 1) System.out.print("one");
```

A two

B two1

C two2

D one

✓ **E** 1one

F The code will not compile because of line 6.

i The code compiles and runs without issue, so option F is incorrect. The expression 10 % 3 is 1, so the if-then boolean expression on line 4 is false, and the branch is not taken. Line 6 outputs "1" regardless of the evaluation of the if-then statement on line 4, since the if-then branch is not part of it (the indentation on line 6 here is a red herring). Finally, the if-then boolean expression on line 7 is true, so "one" is output. Option E is the only answer that matches this output.

9. Which methods will compile if inserted into Joey? (Choose all that apply)

```
class Kangaroo {
public void hop() { }
}
class Joey extends Kangaroo {
// INSERT CODE HERE
}
```

A public void hop() {}

B public void hop() throws Exception {}

C public void hop() throws IOException {}

[✓ D] public void hop() throws RuntimeException {}

E None of these

ℹ Since Kangaroo’s method does not declare any checked exceptions, Joey’s method is not allowed to either. IOException and Exception are both checked exceptions.

10. What is the output when new BlackRhino() is called?

```
public class Rhinoceros {
public Rhinoceros() { System.out.print("1"); }
}
public class BlackRhino extends Rhinoceros {
public BlackRhino(int age) { System.out.print("2"); }
public BlackRhino() {
this(5);
System.out.print("3");
}
}
```

[✓ A] 123

B 132

C 12

D 13

E This code does not compile.

ℹ The code compiles and runs without issue, so option E is incorrect. The main() method calls the no-argument constructor of BlackRhino, which first invokes the other constructor in BlackRhino, which takes an int value. The first line of this constructor is automatically added by the compiler as super(). The parent constructor is then called, resulting in the first output value of 1. The process then unwinds from the bottom up with 2 and 3 outputted as the constructors are visited, so option A is correct.

11. Suppose we have the following classes named InventoryItem and Order. Which one of the following statements inserted at line 2 of the Order class will make the Order class compile successfully?

```
1: package com.abc.products;
2: public class InventoryItem { }

1: package com.abc.orders;
2: // INSERT CODE HERE
3: public class Order {
4: InventoryItem [] items; }
```

A import com.abc.products;

✓ B

import com.abc.products.*;

C import static com.abc.products;

D import com.abc.products.InventoryItem.*;

E Nothing; the class compiles as is.

> i Option E is incorrect. Without any imports, the Order class will not compile because line 6 of Order refers to InventoryItem, a class in a different package. Options A, C, and D are not valid statements because they do not import a class. Option B is the only valid statement and is also sufficient for making the Order class compile successfully, so the answer is option B.

12. Which of the following are true? (Choose all that apply)

✓ A All String literals are automatically instantiated into a String object.

✓ B The StringBuilder and StringBuffer classes define the same public methods.

C A StringBuilder object is immutable.

D A StringBuilder object cannot change its length once it is instantiated.

✓ E A String object cannot change its length once it is instantiated.

> i String literals such as "a" are automatically used as String objects. Although you don’t need to know StringBuffer, you should remember that it has the same methods as StringBuilder. A String is immutable, but StringBuilder and StringBuffer are not. Therefore, a String cannot change its length.

13. What is the result of the following code?

```
public class Deer {
static int count;
static { count = 0; }

Deer() {
count++;
}

public static void main(String[] args) {
count++;
Deer mother = new Deer();
Deer father = new Deer();
Deer fawn = new Deer();
System.out.println(father.count);
}
}
```

A 0

B 1

C 2

D 3

✓ E 4

F 5

G The code does not compile.

i Since *count* is static, there is only one variable across all the instances of Deer. It is incremented once in the static block and again in each of the three constructors. It would be good practice to name this variable *COUNT*, but the exam doesn't rely on good practice in an attempt to confuse you.

14. Which of the following are true statements? (Choose all that apply)

A Java allows extending from multiple classes directly.

✓ B Java allows implementing multiple interfaces directly.

C Java code compiled on Unix must be recompiled to run on Windows.

✓ D Java supports method overloading.

E Java supports operator overloading.

i Option A is correct because Java only allows single inheritance of classes. Option B is incorrect because multiple interfaces can be implemented. Option C is incorrect because Java bytecode is platform independent. Option D is correct because methods can be overloaded to have different signatures. Option E is incorrect because this is a C++ feature that Java intentionally chose to omit.

15. What is the result of this code?

```
1: public class C {
2: String seq = "c";
3: { seq += "g"; }
4: public C() {
5: this("abc");
6: seq += "y";
7: }
8: public C(String s) {
9: seq += "e";
10: }
11: { seq += "z"; }
12: public static void main(String[] args) {
13: C c = new C();
14: System.out.println(c.seq);
15: } }
```

- **A** ceygz
- **B** cgeyz
- **C** cgyez
- ✓ **D** cgzey
- **E** cgzye
- **F** cyegz
- **G** The code does not compile.

i Line 13 kicks off initialization. First comes the instance declaration on line 2. Next are any instance initializers in the order they appear. At this point, *seq* is cgz. Next is the constructor. The constructor on line 4 calls the one on line 7, which adds e to *seq*. Finally, the no-argument constructor completes and adds y to *seq*.

16. Which of the following is a reference variable (and not a primitive)? (Choose all that apply)

- ✓ **A** int[] ints;
- ✓ **B** long[] longs;
- ✓ **C** String[] strings;
- ✓ **D** Object[] objects;
- **E** None of the above

i All array types are reference variables. Although int is a primitive type, int[] is still a reference type.

17. What is the result of the following code snippet?

```
3: int x = 5;
4: while(x >= 0) {
5: int y = 3;
6: while (y > 0) {
7: if(x<2) continue;
8: x--; y--;
9: System.out.print(x*y+" ");
10: }
11: }
```

A 42 31 20 12

B 6 4 2 3

C 8 3 0 2

D The code will not compile because of line 6.

E The code will not compile because of line 7.

✓ F The code contains an infinite loop and does not terminate.

i The thing to notice about this example is that soon as x reaches 1, the outer loop will run infinitely. Notice that x and y are both modified by the inner loop if x is not less than 2. Therefore, it takes only four iterations of the inner loop for x to reach 1. When the program executes, the inner loop will run exactly three times, as y goes from 3 to 0 and x from 5 to 2. The control will then be returned to the outer loop, and the inner loop will *attempt* to run three more times, as the value of y has been reset. During the new first iteration of the inner loop, though, x will be set to 1. The inner loop will then be called again and the if statement and continue will activate. The value of y, as well as the value of x, are no longer modified, so the inner loop will run infinitely, and the answer is option F. Even if the inner loop did terminate, nothing is modifying x in the outer loop boolean expression, so that loop will run infinitely as well. Before reaching the infinite state, the code will output 8 3 0 2.

18. What is the result of the following class?

```
1: import java.util.function.*;
2:
3: public class Panda {
4: int age;
5: public static void main(String[] args) {
6: Panda p1 = new Panda();
7: p1.age = 1;
8: check(p1, Panda p → p.age < 5);
9: }
10: private static void check(Panda panda, Predicate<Panda> pred) {
11: String result = pred.test(panda) ? "match" : "not match";
12: System.out.print(result);
13: } }
```

- **A** match
- **B** not match
- ✓ **C** Compiler error on line 8
- **D** Compiler error on line 10
- **E** Compile error on line 11
- **F** A runtime exception is thrown.

Line 8 is missing parentheses around the parameter list. The parentheses are allowed to be omitted only when there is one parameter and its type is not declared.

19. Which exception will the following throw?

```
int[] nums = new int[] { 1, 4, 6};
Object p = nums;
String[] two = (String[]) p;
System.out.println(two[two.length]);
```

- **A** ArrayIndexOutOfBoundsException
- ✓ **B** ClassCastException
- **C** IllegalArgumentException
- **D** NumberFormatException
- **E** None of the above

The cast fails because int[] and String[] are not compatible.

20. What is the output of the following code?

```
1: class Car {
2: public int velocity = 10;
3: }

1: public class TestDrive {
2: public static void go(Car c) {
3: c.velocity += 10;
4: }
5: public static void main(String[] args) {
6: Car porsche = new Car();
7: go(porsche);
8:
9: Car stolen = porsche;
10: go(stolen);
11:
12: System.out.println(porsche.velocity);
13: }
14: }
```

A 0

B 10

C 20

✓ D 30

E 40

F

The code does not compile.

i The code compiles, so option F is incorrect. The Car object on line 6 has an initial *velocity* of 10 from the Car object instantiation. The call to go() on line 7 changes its *velocity* to 20. The stolen reference on line 9 points to the same Car object, so calling go() changes the *velocity* to 30.

21. What is the result of the following code?

```
public class Letters {
{ System.out.print("a"); }
public Letters() {
{ System.out.print("b"); }
}
{ System.out.print("c"); }
public static void main(String[] args) {
Letters a = new Letters();
{ System.out.print("d"); }
} }
```

- **A** abcd
- ✓ **B** acbd
- **C** bacd
- **D** bdac
- **E** dbad
- **F** The code does not compile.

> **i** Instance initializers run in the order they appear in the class and before the constructor. In the main() method, we have a block of code that prints a value. Though the braces are unnecessary, they are allowed and have no impact on the order of execution.

22. What is the output of the following code?

```
public class HowMany {
public static int count(int a) {
if (a != 3) { int b = 1;
} else {
int b = 2;
}
return a++ + b;
}
public static void main(String[] args) {
System.out.print(count(3));
System.out.print(count(9));
}}
```

- **A** 39
- **B** 49
- **C** 410
- **D** 511
- ✓ **E** The code fails to compile.

> **i** *b* is a local variable that is only in scope within the if statement (or else statement). By the time the return statement is reached, the compiler no longer knows about *b*.

23. Which are true of the following code? (Choose all that apply)

```
package aquarium;
public class Water {
public String toString() { return ""; } }

package aquarium;
public class Shark {
static int numFins;
static Water water;
public static void Main(String[] args) {
String s1 = water.toString();
String s2 = numFins.toString(); } }
```

A Shark.java needs an additional import to compile.

B water.toString() does not compile.

✓ **C** numFins.toString() does not compile.

D Water.java does not compile.

E java Shark will run the program and produce no output.

F The code compiles as is.

> **i** toString() is a method and methods are only allowed to be called on objects (reference types.) numFins.toString() does not compile because *numFins* is a primitive. Since Shark and Water are in the same package, no imports are needed. Although Shark does compile, it does not have a main() method and will therefore result in an error if you try to run it. Remember that Java is case sensitive, which means main() and Main() are different methods.

24. What does the following code output when run as java Duck Duck Goose?

```
public class Duck {
public void main(String[] args) {
for (int i = 1; i <= args.length; i++)
System.out.println(args[i]); } }
```

A Duck Goose

B Duck ArrayIndexOutOfBoundsException

C Goose

D Goose ArrayIndexOutOfBoundsException

✓ **E** None of the above

> **i** The main() method isn't static; it is a method that happens to be named, but it's not an application entry point. When the program is run, it gives the error Error: Main method is not static in class Duck, please define the main method as: public static void main(String[] args). If the method were static, the answer would be option D. Arrays are zero based, so the loop ignores the first element and throws an exception when accessing the element after the last one.

25. What is the result of the following code?

```
1: interface Climb {
2: boolean isTooHigh(int height, int limit);
3: }
4:
5: public class Climber {
6: public static void main(String[] args) {
7: check((h, l) → h > l, 5);
8: }
9: private static void check(Climb climb, int height) {
10: if (climb.isTooHigh(height, 10))
11: System.out.println("too high");
12: else
13: System.out.println("ok");
14: }
15: }
```

- ✓ **A** ok
- **B** too high
- **C** Compiler error on line 7
- **D** Compiler error on line 10
- **E** Compiler error on a different line
- **F** A runtime exception is thrown.

i Since the interface takes two parameters, the lambda expression needs to as well. It correctly specifies the parameters and the code functions as if we defined the lambda in an interface. Line 7 passes a *height* of 5 and the lambda expression is for deferred execution. check() calls the code and actually executes it.

26. What is the result of the following code snippet?

```
3: String year = "Senior";
4: switch(year) {
5: case "Freshman" : default: case "Sophomore" :
6: case "Junior" : System.out.print("See you next year"); break;
7: case "Senior" : System.out.print("Congratulations");
8: }
```

- **A** See you next year
- ✓ **B** Congratulations
- **C** See you next yearCongratulations
- **D** The code does not output any text.
- **E** The code will not compile because of line 4.
- **F** The code will not compile because of line 5.

i The code compiles and runs without issue, so options E and F are incorrect. The year variable matches the last case statement only. The "Congratulations" branch statement is output, so option B is the correct answer.

27. What is the result of the following code?

```
1: public class BytePrinter {
2: public void print(byte b) {
3: System.out.print("byte");
4: }
5: public void print(Byte b) {
6: System.out.print("Byte");
7: }
8: public void print(int i) {
9: System.out.print("int");
10: }
11: public static void main(String[] args) {
12: BytePrinter printer = new BytePrinter();
13: short x = 10;
14: byte y = 12;
15: printer.print(x);
16: printer.print(y);
17: }
18: }
```

A byteint

[✓ B] intbyte

C Compiler error on line 5

D Compiler error on line 15

E Compiler error on line 16

The code compiles fine, so options C, D, and E are incorrect. The call to print() on line 15 invokes the overloaded print() method on line 8 because short can be widened to int. The call to print() on line 16 invokes the print() method on line 2 because it is an exact match. Therefore, the output is intbyte and the answer is option B.

28. What is the output of the following snippet?

```
int a = 123;
int b = 0;
try {
System.out.print(a / b);
System.out.print("1");
} catch (RuntimeException e) {
System.out.print("2");
} catch (ArithmeticException e) {
System.out.print("3");
} finally {
System.out.print("4");
}
```

A 14

B 2

C 24

D 3

E 34

✓ F The code does not compile.

G An uncaught exception is thrown.

Since ArithmeticException extends RuntimeException, the ArithmeticException catch block is unreachable code. These two blocks need to be reversed to compile.

29. Which of the following fill in the blank to print out "c"?

```
String letters = "abcde";
String answer = ____________________
System.out.println(answer);
```

A letters.charAt(2);

B letters.charAt(3);

C letters.substring(2, 2);

✓ D letters.substring(2, 3);

E letters.substring(3, 3);

F letters.substring(3, 4);

The substring() method takes the index of the character to start with as the first parameter. Since indexes are zero based, this is 2. The second parameter is the index after you want to stop capturing characters. Options C and E return an empty String. Option F returns d. Options A and B do not compile because they return a character, not a String.

30. Which of braces can be removed without causing a compiler error? (Choose all that apply)

```
public static void main(String[] args) {
int num1 = 8;
int num2 = 8;
for (int i = 0; i < 3; i++) {
if (num1 == num2) {
try {
System.out.println("t");
} catch (Exception e) {
System.out.println("c");
}
}
}
}
```

- **A** The braces for the method
- ✓ **B** The braces for the loop
- ✓ **C** The braces for the if statement
- **D** The braces for the try
- **E** The braces for the catch
- **F** None of the above; the code does not compile as is.

> **i** The {} are required around methods, try blocks, and catch blocks even if there is only one statement inside. They are not required for loops and if statements.