socrative

Answer Key
February 24, 2021

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

1. What is the result of this code?

```
public class HowMany {
public int add(int one, int two) {
return one + two;
}
public static void main(String[] args) {
HowMany h = new HowMany();
int result = h.add(2, 3);
System.out.println(result);
}
}
```

A 2

B 3

✓ C 5

D The code does not compile.

E An exception is thrown.

i This might seem like a really easy question—and it is. Many of the questions on the exam are easy. This exam gives tougher questions to get you ready. When you see an easy question, check carefully you aren't missing a trick and then answer. In this case, the instance method takes two parameters, adds them and returns the number.

2. Determine the output of the following code when executed with the command:
java HelloWorld hello world goodbye

```
1: public static class HelloWorld {
2: public static void main(String [] args) {
3: System.out.println(args[1] + " " + args[2]);
4: } }
```

A hello world

B world goodbye

C null null

D An ArrayIndexOutOfBoundsOccurs at runtime.

✓ E Does not compile

F Throws a different exception

i The class declaration on line 1 contains the static modifier, which is not a valid modifier for a top-level class. This causes a compiler error. If the static keyword were removed, the answer would be option B because arrays start with index 0, not 1.

3. 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, "max"))
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 The interface takes two int parameters. The code on line 10 passes an int parameter and a String parameter. This question tries to distract you with lambdas so that you don't notice the parameter types are wrong.

4. What is the result of the following code?

```
public class A {
private int i = 6;
private int j = i;
public A() {
i = 5;
}
public static void main(String[] args) {
A a = new A();
System.out.println(a.i + a.j);
} }
```

- A 55
- B 56
- C 66
- D 10
- ✓ E 11
- F 12
- G The code does not compile.

i The constructor runs after the instance initializers. *i* and *j* both get initialized to 6. Then the constructor runs and changes *i* to 5. Since both numbers are integers, Java adds them rather than concatenating them.

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

```
StringBuilder b = new StringBuilder("1");
StringBuilder c = b.append("2");
b.append("234");
c.deleteCharAt(1);
System.out.println("Equals? " + (b == c));
System.out.println("B=" + b);
System.out.println("C=" + c);
```

- ✓ A The output contains Equals? true
- B The output contains Equals? false.
- ✓ C The output contains B=1234.
- D The output contains B=234.
- ✓ E The output contains C=1234.
- F The output contains C=2.
- G The code does not compile.

i When calling a StringBuilder method such as append(), it returns the object itself. This means *b* and *c* point to the same object in memory. After both appends, that value is 12234. Since indexes are zero based, the first 2 is removed and we have 1234.

6. Which of the following are unchecked exceptions? (Choose all that apply)

- ✓ **A** ArithmeticException
- ✓ **B** ClassCastException
- **C** FileNotFoundException
- **D** NoClassDefFoundError
- **E** StackOverflowError

i Options A and B are unchecked exceptions because they extend RuntimeException. Option C is a checked exception, and options D and E are errors.

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

```
3: int x = 4;
4: if(x <= 5 ^ x == 4)
5: System.out.print("Low");
6: else if(x==4)
7: System.out.print("Match");
8: if(x>1) System.out.print("High");
```

- **A** Low
- **B** Match
- **C** High
- **D** LowHigh
- ✓ **E** MatchHigh
- **F** The code will not compile because of line 4.
- **G** The code will not compile because of line 8.

i This code will compile and run without issue, so options F and G are incorrect. On line 4, the expression on the left-hand side and right-hand side of the exclusive OR ^, both evaluate to true, and since (true ^ true) is false, the first branch will not be taken. On line 6, x does equal 4, so line 7 will be executed and "Match" will be printed. Finally, since the if-then statement on line 8 is not connected to the previous statements (despite the indentation), it will also be executed and "High" will be printed. The answer that matches the output is option E. Note that this “matching algorithm” is somewhat broken conceptually, because a value of x of 5 will output LowHigh. The key is to focus on what the algorithm does, not what it does it well.

8. What is the output of the following snippet?

```
12: int a = 123;
13: int b = 0;
14: try {
15: System.out.print(a / b);
16: System.out.print("1");
17: } catch (ArithmeticException e) {
18: System.out.print("2");
19: } catch (RuntimeException r) {
20: System.out.print("3");
21: } finally {
22: System.out.print("4");
23: }
```

A 14

B 2

✓ C 24

D 234

E The code does not compile.

F An uncaught exception is thrown.

i Line 15 attempts integer division by 0, which throws an ArithmeticException. Line 16 is skipped and the catch block on line 17 handles the exception and prints 2. Now that the exception is handled, the finally block on line 21 executes and prints 4.

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

```
3: String tiger = "Tiger";
4: String lion = "Lion";
5: final String statement = 250 > 338 ? lion : tiger = " is Bigger";
6: System.out.println(statement);
```

A Tiger

B Lion

C Tiger is Bigger

D Lion is Bigger

E is Bigger

✓ F The code will not compile because of line 5.

i The code does not compile, because the assignment operator has the highest order of precedence in this expression. It may be helpful to see what the compiler is doing by adding optional parentheses:
final String statement = (250 > 338 ? lion : tiger) = " is Bigger";
This expression is then reduced to:
final String statement = "tiger" = " is Bigger";
This expression is invalid, as the left-hand side of the second assignment operator is not a variable, so the answer is option F. Note that if the question had added explicit parentheses around the expression (tiger = " is Bigger"), option E would have the correct output.

10. What is the result of the following code assuming garbage collection runs on line 6?

```
1: public class Caterpillar {
2: public static void main(String[] args) {
3: Caterpillar c1 = new Caterpillar();
4: Caterpillar c2 = new Caterpillar();
5: c1 = c2; c2 = null;
6: // garbage collection runs
7: c1 = null;
8: }
9: protected void finalize() {
10: System.out.println("becomes a butterfly");
11: } }
```

A "becomes a butterfly" is never printed.

✓ B "becomes a butterfly" is printed exactly once.

C "becomes a butterfly" is printed exactly twice.

D The code fails compilation on line 3.

E The code fails compilation on line 9.

F The code fails compilation on another line.

i After line 5, *c1* points to the Caterpillar object instantiated on line 4 and *c2* is null. Garbage collection sees it can clean up the Caterpillar object instantiated on line 3 and calls its finalize() method. The code compiles fine. A default constructor is provided if no constructors are defined and nothing is wrong with the finalize() method.

11. Which of the following correctly overload this method? (Choose all that apply)

```
public void buzz() { }
```

✓ A private void buzz(String sound) { }

B public final void buzz() { }

C public static void buzz() { }

✓ D public static void buzz(int... time) { }

✓ E public void buzz(boolean softly) { }

F public void buzzLouder() { }

i Options B and C are incorrect because they have the same name and empty parameter list as the original. One of these must differ to be an overload. Option F is incorrect because the method name is different than the original. It is a perfectly fine method, but it isn't an overload. Options A, D, and E are correct overloads because the method name is the same and the method parameter list is different. Anything else is allowed to vary. Remember that Java is case sensitive.

12. What is the output of the following code?

```
LocalDate date = LocalDate.of(2018, Month.APRIL, 30);
date = date.plusDays(2);
date = date.plusYears(3);
System.out.println(date.getYear() + " " + date.getMonth() + " "
+ date.getDayOfMonth());
```

A 2018 APRIL 2

B 2018 APRIL 30

C 2018 MAY 2

D 2012 APRIL 2

E 2012 APRIL 30

✓ F 2021 MAY 2

i The date starts out as April 30, 2018. Adding two days takes us to May 2, 2018. Then adding three years brings us to May 2, 2021, making option F the correct answer.

13. What is the result of this code?

```
public class Cub {
private String name;
private double weight;
public Cub(double weight) {
this.weight = weight;
this("", weight);
}
public Cub(String name, double weight) {
weight = weight;
this.name = name;
}
public static void main(String[] args) {
Cub cub = new Cub(44);
System.out.println(cub.weight + "" + cub.name);
}
}
```

A 0

B null

C 44

D 44 null

E 44.0

F 44.0 null

✓ G The code does not compile.

i this() must be the first line of a constructor if present. If the order of the statements in the constructor were reversed, the answer would be option E.

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

- ✓ **A** For a String, capacity would be a redundant property.
- **B** For a StringBuilder, capacity would be a redundant property.
- ✓ **C** An empty String has a length of zero.
- **D** An empty String has a length of one.
- **E** A StringBuilder's length is sometimes greater than its capacity.
- ✓ **F** A StringBuilder's length is never greater than its capacity.

i In a String, the length and capacity are the same because a String is immutable. In a StringBuilder, the length can be less than or equal to the capacity, but never more. An empty String has a size of zero because it does not contain any characters.

15. Which of the following imports can be inserted to make the Robot class compile? (Choose all that apply)

```
package util;
public interface Ports {
int RIGHT_MOTOR = 1;
int LEFT_MOTOR = 2;
}
package robot;
// INSERT CODE HERE
public class Robot {
Robot(int motor1, int motor2) { }
public static void main(String[] args) {
Robot robbie = new Robot(Ports.RIGHT_MOTOR, LEFT_MOTOR);
}
}
```

- **A** `import static util.Ports.*;`
- ✓ **B** `import static util.Ports.*;`
  `import util.*;`
- **C** `import static util.Ports;`
- **D** `static import util.Ports.*;`
- **E** `static import util.Ports.*;`
  `import util.*;`
- **F** `static import util.Ports;`

i The Robot class refers to two things that need to be imported. One is the Ports interface because Ports.RIGHT_MOTOR appears in the code. The other is a static import of Ports.LEFT_MOTOR because this constant is referred to without the classname. Options D, E, and F are all incorrect because static imports are specified with import static. Option A is incorrect because it only imports the static members of the Ports interface. Java still doesn't know where to find Ports when it is referred to by name. Option C is incorrect because you cannot do a static import of a class or interface. Static imports are for static members only. Option B is correct. The first import lets us refer to LEFT_MOTOR without writing Ports first. The second import lets us reference Ports itself.

16. What is the result of the following?

```
6: String[] lizards = {"iguana", "gecko"};
7: List<String> list = Arrays.asList(lizards);
8: list.set(1, "");
9: System.out.println(list);
```

A [iguana]

✓ B [iguana,]

C [iguana, gecko]

D An exception is thrown on line 8.

E An exception is thrown on line 9.

F The code does not compile.

i When you're converting from an array to a List, the returned List is a fixed size because it is backed by the array. It is not immutable. Elements can be changed, but not removed or added.

17. What is the result of the following code?

```
public class Cardinal {
static int number;
Cardinal() { number++; }
public static void main(String[] args) {
Cardinal c1 = new Cardinal();
if (c1 == null) {
Cardinal c2 = new Cardinal();
} else {
Cardinal c2 = new Cardinal();
}
Cardinal c2 = new Cardinal();
System.out.println(c1.number);
} }
```

A 0

B 1

C 2

✓ D 3

E 4

F The code does not compile.

i The constructor is called three times. *c1* and *c3* should be obvious. *c2* is created inside the else block of the if statement. The code does compile. It is legal to use an instance variable to refer to a static variable. Class (static) variables are initialized to the default value for that type—zero in this case.

18. What is the result of the following code?

```
11: List differentTypes = new ArrayList();
12: differentTypes.add("goldfish");
13: differentTypes.add(0, false);
14: differentTypes.add(1);
15: differentTypes.remove(1);
16: boolean b1 = differentTypes.contains("goldfish");
17: boolean b2 = differentTypes.contains(new Boolean(false));
18: boolean b3 = differentTypes.contains(1);
19: System.out.println(b1 + " " + b2 + " " + b3);
```

- **A** false false true
- ✓ **B** false true true
- **C** true false false
- **D** true true false
- **E** An exception is thrown.
- **F** The code does not compile.

i Since this code doesn't use generics, any object can be added to differentTypes. The later two use autoboxing to convert from a primitive to an object. Line 12 inserts at index 0, giving us [false, goldfish, 1]. On line 15, the remove() method does not autobox the 1 to new Integer(1) because there is already a remove() method that takes an int—the index of the element to remove—which gives us [false, 1]. Line 16 returns false since this is the element we removed. Line 17 returns true. Since the primitive was autoboxed, we can refer to it using the wrapper type. Line 18 also returns true because 1 is still in the list.

19. Which of the following exceptions are always thrown programmatically? (Choose all that apply)

- **A** ArrayIndexOutOfBounds
- **B** ExceptionInInitializerError
- ✓ **C** java.io.IOException()
- **D** NullPointerxception()
- ✓ **E** NumberFormatException

i All of these exceptions can be thrown programmatically, but the question asks which ones always are. An IOException is thrown by many methods that read/write files and is always thrown programmatically. Similarly, NumberFormatException is thrown when converting numbers. The other three exceptions are thrown by the JVM.

20. Which lambda can replace the MySecret class? (Choose all that apply)

```
interface Secret {
String test(String a, String b);
}

class MySecret implements Secret {
public String test(String a, String b) {
return a + b;
}
}
```

- ✓ **A** caller((a, b) → a + b);
- **B** caller((String a, b) → a + b);
- ✓ **C** caller((String a, String b) → a + b);
- **D** caller((a, b) , a + b);
- **E** caller((String a, b) , a + b);
- **F** caller((String a, String b) , a + b);

i This interface specifies two String parameters. We can provide the parameter list with or without parameter types. However, it needs to be consistent, making option B incorrect. Options D, E, and F are incorrect because they do not use the arrow operator. This is also tricky because test() is the method name used in Predicate. This is not a predicate, so it can have a different number of parameters.

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

```
3: int x = 10;
4: switch(x % 4.) {
5: default: System.out.print("Not divisible by 4");
6: case 0: System.out.print("Divisible by 4");
7: }
```

- **A** Not divisible by 4
- **B** Divisible by 4
- **C** Not divisible by 4Divisible by 4
- **D** The code does not output any text.
- ✓ **E** The code will not compile because of line 4.

i For this problem you need to remember your rules about numeric promotion as well as what data types are allowed in a switch statement. The expression x % 4. automatically promotes the x to a double; since 4. is a double, the result is a double. Because double is not one of the types accepted by a switch statement, the code fails to compile due to line 4, so option E is the correct answer.

22. Which of the following complete this method to print out all the arguments? (Choose all that apply)

```
public void print(String... args) {
// INSERT CODE HERE
}
```

✓ **A** for (int i = 0; i < args.length; i++)
System.*out*.println(args[i]);

B for (int i = 0; i <= args.length; i++)
System.*out*.println(args[i]);

C for (int i = 1; i < args.length; i++)
System.*out*.println(args[i]);

D for (int i = 1; i <= args.length; i++)
System.*out*.println(args[i]);

✓ **E** for (String arg : args)
System.*out*.println(arg);

F None of the above; args is not an array so it cannot be accessed like one.

i This method uses varargs, which are accessed just like an array. Option A is the standard loop for an array and option E is the standard enhanced for loop. Options B and D are incorrect because args[args.length] is one past the end of the array. Option C is incorrect because it misses the first element, which has index 0.

23. What is the output of the following code?

```
1: class Animal {
2: public int getAge() {return 10;}
3: }
4: class Mammal extends Animal {
5: protected int getAge(int input) {return 7;}
6: }
7: public class Sloth extends Mammal {
8: public boolean hasFur() {return true;}
9: public static void main(String[] args) {
10: Mammal sloth = new Sloth();
11: System.out.print(sloth.getAge());
12: System.out.print(sloth.getAge(2));
13: System.out.print(sloth.hasFur());
14: }
15: }
```

A 7true10

B 10true7

C The code will not compile because of line 5.

D The code will not compile because of line 11.

E The code will not compile because of line 12.

✓ **F** The code will not compile because of line 13.

i The method getAge() is overloaded in Mammal, not overridden, so the access modifier change from

public to the more restrictive protected is permitted. Lines 11 and 12 also compile and run without issue, since both getAge() methods exist in the Mammal reference. The code will fail to compile on line 13, though, because the Mammal reference does not have access to the Sloth method hasFur(). Even though the object is a Sloth object, it is accessed via polymorphism via the Mammal reference; therefore, the compiler will throw an error on this code, and option F is the correct answer.

24. Which lambda can replace the MySecret class? (Choose all that apply)

```
interface Secret {
int number();
}

class MySecret implements Secret {
public int number() {
return 5;
}
}
```

- A caller(→ 5);
- B caller(x → 5);
- ✓ C caller(() → 5);
- D caller((x) → 5);
- E caller(() → {5});
- ✓ F caller(() → { return 5;});

i Since the interface doesn't take any parameters, we need to pass an empty parameter list. Option A is incorrect because it does not specify a parameter list. Options B and D are incorrect because they pass one parameter. Option E is incorrect because the return keyword is missing.

25. Which of the following lists is a correct sequence for catching exceptions? (Choose all that apply)

- ✓ A ArrayIndexOutOfBoundsException, IOException, RuntimeException
- B ArrayIndexOutOfBoundsException, Exception, IOException
- C Exception, IOException, ArrayIndexOutOfBoundsException
- D Exception, ArrayIndexOutOfBoundsException, IOException
- E IOException, RuntimeException, ArrayIndexOutOfBoundsException
- ✓ F IOException, ArrayIndexOutOfBoundsException, Exception

i Exception may not be before any other exception. RuntimeException cannot be before ArrayIndexOutOfBoundsException.

26. Which of the following lines compile? (Choose all that apply)

```
11: StringBuilder s1 = new StringBuilder();
12: StringBuilder s2 = new StringBuilder(5);
13: StringBuilder s3 = new StringBuilder(6.0);
14: StringBuilder s4 = new StringBuilder("b");
15: StringBuilder s5 = new StringBuilder(false);
```

- ✓ **A** Line 11
- ✓ **B** Line 12
- **C** Line 13
- ✓ **D** Line 14
- **E** Line 15

i Line 11 creates an empty StringBuilder with the default capacity. Line 12 creates an empty StringBuilder with a capacity of 5. Line 13 does not compile since the capacity must be an integer rather than a double. Line 14 creates a StringBuilder with a length of 1 and the default capacity. Line 15 does not compile because booleans cannot create a StringBuilder.

27. Which of the following can fill in the blank on line 5 to make the code compile? (Choose all that apply)

```
1: package x.y;
2: public class Movie {
3: private String title;
4: protected String rating;
5: int length;
6: public String description;
7: }
```

```
1: package x.y;
2: public class MovieTheater {
3: private Movie movie = new Movie();
4: public MovieTheater() {
5: _____
6: } }
```

- **A** movie.title = "Office Space";
- ✓ **B** movie.rating = "R";
- ✓ **C** movie.length = 89;
- ✓ **D** movie.description = "Comedy";
- **E** None of the above

i Public access is available from anywhere, making option D correct. Both classes are in the same package, which means default and protected access are allowed; thus, options B and C are correct. Since there are two classes, private access is not allowed.

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

```
3: for(int i=0; i<5; i++) {
4: System.out.print(1);
5: if(i<2) continue;
6: else break;
7: }
```

- A 1
- B 11
- ✓ C 111
- D 1111
- E 11111
- F The code produces no output.
- G The code will not compile because of line 5.

i The code compiles and runs without issue. During the first two iterations of the loop, 1 is output twice, and since the value of i is less than 2, continue is called, ending the iteration of the current loop. On the third iteration of the loop, 1 is output again, and i < 2 evaluates to false since i is 2. Therefore, the else branch is reached, which immediately ends the loop. The result is that 1 has been printed three times, so option C is the correct answer.

29. What is the output of the following program?

```
1: public class ExceptionTest {
2: public static void main(String[] args) {
3: String ref = null;
4: try {
5: System.out.print(ref.toString());
6: System.out.print("a");
7: } catch (NumberFormatException e) {
8: System.out.print("b");
9: } finally {
10: System.out.print("c");
11: }
12: System.out.print("d");
13: } }
```

- A acd
- B bcd
- C bc and the stack trace for a NullPointerException
- ✓ D c and the stack trace for a NullPointerException
- E cd and the stack trace for a NullPointerException
- F The code does not compile.

i Line 5 throws a NullPointerException, so line 6 skipped. The catch block does not handle this type of exception, so control proceeds to the finally on line 9. Since the exception was not caught, main() throws the exception and the program ends with a NullPointerException.

30. Which creates an object with the current hours, minutes, and seconds?

- **A** new LocalDate();
- **B** new LocalTime();
- **C** LocalDate.current();
- **D** LocalTime.current();
- **E** LocalDate.now();
- ✓ **F** LocalTime.now();

i Options A and B are incorrect because the date and time classes have private constructors. Options C and D are incorrect because there is no such method. Option E is incorrect because the question asks about time.