socrative

Answer Key
February 24, 2021

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

1. Which statements are true? (Choose all that apply)

A If a try block contains System.exit(0), the finally block will run.

✓ B If a try block contains System.exit(0), the finally block will not run.

C If a catch block contains System.exit(0), the finally block will run.

✓ D If a catch block contains System.exit(0), the finally block will not run.

E A try block will not compile containing System.exit(0).

i System.exit(0) ends the program immediately. No code runs after that line.

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

```
StringBuilder s1 = new StringBuilder("meow");
StringBuilder s2 = new StringBuilder("meow");
if (s1 == s2) System.out.println("one");
if (s1.equals(s2)) System.out.println("two");
if (s1 == "meow") System.out.println("three");
if (s1.toString() == "meow") System.out.println("four");
```

A The code compiles.

✓ B The code does not compile.

C If statements that do not compile are removed, the output contains one.

D If statements that do not compile are removed, the output contains two.

E If statements that do not compile are removed, the output contains three.

F If statements that do not compile are removed, the output contains four.

i The two StringBuilder objects are different objects and so are not equal using reference equality. StringBuilder does not implement the equals method, so you cannot compare their values using equals() either. The third if statement doesn't even compile. The compiler is smart enough to know String and StringBuilder are of different types and couldn't possibly point to the same object. The fourth one is tempting. However, only one value is a literal from the string pool.

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

```
3: boolean x = false;
4: int z = 0;
5: do {
6: if(z>5) {z++; x = true;}
7: else z += 3;
8: } while (!(x = true));
9: System.out.print(z);
```

- **A** 8
- **B** 0
- **C** 7
- ✓ **D** 3
- **E** 6
- **F** The code will not compile because of line 8.
- **G** The code contains an infinite loop and does not terminate.

i The code compiles and runs without issue, so options F and G are incorrect. The looks a little complicated, until you realize that the loop is only ever executed once. The assignment operator = was used instead of the equality operator in the boolean expression of the do-while statement; therefore x is assigned a value of true the first time it runs. Since the complement of true is false, the loop will execute once and then terminate. On the single iteration of the loop, the value z is not greater than 5, so it is updated from 0 to 3. The output after the loop is 3, so the answer is option D. If the loop had used the equality operator instead of the assignment operator, it would have executed and stopped after three iterations, outputting 7, so option C would have been correct.

4. What is the output when new Hippopotamus() is called?

```
1: class Mammal {
2: public Mammal(int age) { System.out.println("1"); }
3: }
4: public class Hippopotamus extends Mammal {
5: public Hippopotamus() { System.out.println("2"); }
6: }
```

- **A** 12
- **B** 1
- **C** 2
- **D** The code will not compile because of line 2.
- ✓ **E** The code will not compile because of line 5.
- **F** The code will not compile because of line 5.

i The subclass Hippopotamus defines a constructor that does not make an explicit call to a parent constructor. The compiler attempts to compensate for this by insert a call to the default no-argument constructor super(), but there is no such constructor defined within Mammal; therefore, the code will not compile and option E is correct. The rest of answers are incorrect for this reason.

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

```
3: String[] values = {"one","two","three"};
4: for(int index = 0; index < values.length; index++)
5: System.out.print(values[index]);
6: System.out.print(index);
```

- A onetwothree
- B onetwothree2
- C one0two1three2
- D The code will not compile because of line 4.
- ✓ E The code will not compile because of line 6.

i The variable index is defined in the initiation block of the for loop and is therefore unavailable after the loop has terminated. The code will not compile, so option E is the correct answer.

6.

What is the result of this code?

```
public class Chicken {
private void layEggs(int... eggs) {
System.out.print("many " + eggs[0] + " ");
}
private void layEggs(int eggs) {
System.out.print("one " + eggs + " ");
}
public static void main(String[] args) {
Chicken c = new Chicken();
c.layEggs(1, 2);
c.layEggs(3);
c.layEggs(new Integer(2));
}
}
```

- ✓ A many 1 one 3 one 2
- B many 2 one 3 one 2
- C many 1 many 3 one 2
- D many 2 one 3 many 2
- E The code does not compile.
- F An exception is thrown.

i Java tries to use more specific method signatures first. The first call to layEggs() cannot use the int signature since it has two parameters. Java uses zero-based indexes so it outputs many 1. The other two calls can use the int signature. The last one uses autoboxing.

7. Which of the following are supported by Java? (Choose all that apply)

- ✓ **A** Encapsulation
- **B** Indentation changing the meaning of code
- ✓ **C** Inheritance
- **D** Operator overloading
- ✓ **E** Platform independence
- **F** Pointers

> **i** Encapsulation, inheritance, and platform independence are all key features are Java. Operator overloading and pointers are from C++ and are not included in Java. Other languages, such as Python, use indentation to convey meaning; Java does not.

8. Which of the following fill in the blank to remove the blank space in this String? (Choose all that apply)

```
String g = "Guinea Pig";
int i = g.indexOf(" ");
String answer = _____________________
System.out.println(answer);
```

- **A** g.trim();
- **B** g.substring(0, i) + g.substring(i);
- **C** g.substring(0, i - 1) + g.substring(i);
- ✓ **D** g.substring(0, i) + g.substring(i + 1);
- **E** g.substring(0, i - 1) + g.substring(i + 1);
- **F** None of the above

> **i** Writing down the String and index positions should help you. *i* is 6 because indexes are zero based. We want to make the calls substring(0, 6) and substring(7) to get rid of the space. Remember that the second parameter is the index after we want to stop. Option A is a trick: trim() only removes spaces at the beginning or end of a String.

9. 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(Object obj) {
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 The finalize() method in Object does not take any parameters. Since the method defined in Caterpillar does have a parameter, it does not override Object's and is not called during garbage collection. If the parameter was not in Caterpillar's finalize() method, the answer would be option B.

10. What gets printed when running the following snippet?

```
public static void main(String[] args) {
int i = 0;
try {
i += 1;
e();
i += 2;
} finally {
i += 8;
}catch (Exception e) {
i += 4;
}
System.out.print(i);
}
private static void e() {
throw new IllegalArgumentException();
}
```

A 1

B 9

C 11

D 13

✓ **E** The code does not compile.

F An exception is thrown.

i A finally block must be coded after a catch block in a try statement. If the order were reversed, the answer would be option D.

11. Which of the following statements can be inserted in the blank so that the code will compile successfully? (Choose all that apply)

```
public interface WalksOn4Legs {}
public abstract class Mammal {
public int numberOfOffspring;
}
public class Antelope extends Mammal implements WalksOn4Legs {}
public class ParkRanger {
public void noteNewOffspring(Mammal mammal) { mammal.numberOfOffspring++; }
public static void main(String[] args) {
new ParkRanger().noteNewOffspring(_____);
}
}
```

- **A** new Mammal()
- ✓ **B** new Antelope()
- **C** new WalksOn4Legs()
- ✓ **D** (Mammal)new Object()
- **E** (Mammal)new String()
- ✓ **F** null

i Option A is incorrect, since Mammal is declared abstract and cannot be instantiated. Likewise, WalkOn4Legs is an interface, which is inherently abstract; therefore, option C is incorrect. Option B is correct as Antelope is a subclass of Mammal and as a polymorphic parameter can be passed without issue to a method accepting a reference of type Mammal. Option D is correct, since Object is a superclass of Mammal and the compiler allows the cast, even though this would produce a ClassCastException at runtime. Option E is incorrect, since String is not a superclass of Mammal and the compiler detects this and throws a compilation error. Finally, option F is correct—a null value can always be passed as a reference. Although this will compile without issue, it will produce a NullPointerException at runtime.

12. What is the result of this code?

```
class Toy{
private boolean containsIce = false;
public boolean containsIce() {
return containsIce;
}
public void removeIce() {
this.containsIce = true;
}
}
public class Otter {
private static void play(Toy toy) {
toy.removeIce();
}
public static void main(String[] args) {
Toy toy = new Toy();
Otter.play(toy);
System.out.println(toy.containsIce());
}
}
```

A false

✓ B true

C There is one compiler error.

D There are two compiler errors.

E There are three compiler errors.

F An exception is thrown.

i Changes made by method calls to method parameters are reflected in the caller because they both have a reference to the same object.

13. Given the following class definitions, which method signature could appear in a subclass of Albatross? (Choose all that apply)

```
public abstract class SeaBird {
public abstract void fly(int height);
}
public abstract class Albatross {
abstract Long fly();
}
```

A private abstract void fly(int height)

✓ B protected void fly(int height)

C public Number fly()

✓ D Long fly()

✓ E protected Long fly()

i Albatross is not a subclass of SeaBird, so you can ignore the first declaration void fly(int height) in answering the question. Option A is an overloaded method of fly(), but it's an invalid definition since a

method cannot be marked both abstract and private—no class would be able to implement it. Option B is correct—it is an overloaded method, not an overridden one—so the access modifier change from public to protected is allowed. Option C is incorrect because Number is not a subclass of Long—it is a superclass; therefore, they are not covariant return types. Option D is a correct override of the fly() method, since the access modifiers are the same between the parent and child class. Option E is correct since protected is considered a less restrictive override compared to the default access modifier.

14. What is the result of the following code?

```
1: public class Counts {
2: private boolean b;
3: public static void main(String[] args) {
4: Counts c = new Counts();
5: int one, two = 0;
6: if (c.b) {
7: System.out.println(one); } } }
```

- **A** Compiler error on line 5
- **B** Compiler error on line 6
- ✓ **C** Compiler error on line 7
- **D** Compiles successfully and prints 0
- **E** Compiles successfully and prints null
- **F** Compiles successfully with no output

i Since *b* is an instance variable, it is initialized to the default value of false. *two* is explicitly initialized to 0. *one* is a trick. If both variables were set to 0, the code would say int one = 0, two = 0;. Line 7 refers to a variable that was not initialized and fails to compile.

15. What is the result of the following program?

```
1: public class Egret {
2: private String color;
3: public void Egret() {
4: Egret("white");
5: }
6: public void Egret(String color) {
7: color = color;
8: }
9: public static void main(String[] args) {
10: Egret e = new Egret();
11: System.out.println("Color:" + e.color);
12: }}
```

- **A** Color:
- ✓ **B** Color:null
- **C** Color:White
- **D** Compiler error on line 4
- **E** Compiler error on line 10
- **F** Compiler error another line

There are no user-defined constructors in this example. There are two methods that happen to have the same name as the class. You can tell these are methods because they have a return type. Line 4 compiles because it calls one of these methods. Since there are no user-defined constructors, an empty default constructor is generated. Line 10 compiles because it calls this default constructor. The instance variable *color* keeps its default value of null, making B correct.

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

```
3: int x = 9;
4: long y = x * (long) (++x);
5: System.out.println(y);
```

A –1

B 9

C 81

✓ D 90

E The code will not compile because of line 4.

The code compiles and runs without issue, so option E is incorrect. The pre-increment operator is applied to x, resulting in a value of 9 * (long)10. The upcast to long is unnecessary here—the result will be upcast to long after the multiplication. The result is a value of 90, and so option D is the correct answer.

17. What is the result of the following?

```
3: int[] times [] = new int[3][3];
4: for (int i = 0; i < times.length; i++)
5: for (int j = 0; j < times.length; j++)
6: times[i][j] = i*j;
7: System.out.println(times[2][3]);
```

A 1

B 4

C 1 printed 4 times

D 4 printed 3 times

✓ E An exception is thrown

F The code fails to compile because of line 3.

G The code fails to compile for another reason.

This is a 3´3 array. The maximum valid index is 2. Line 7 throws an ArrayIndexOutOfBoundsException because 3 is not a valid index.

18. What is the result of compiling the following code?

```
1: interface CanSwim {
2: public static int SPEED;
3: public void swim();
4: }
5: public class MantaRay implements CanSwim {
6: public void swim() { System.out.println("MantaRay is swimming: "+SPEED); }
7: }
```

- A The code compiles without issue.
- ✓ B The code will not compile because of line 2.
- C The code will not compile because of line 3.
- D The code will not compile because of line 5.
- E The code will not compile because of line 6.
- F The code compiles but throws an exception at runtime.

i The code does not compile properly, so options A and F are incorrect. The compilation error is in line 2; SPEED is an interface variable and all interface variables are implicitly assumed public static final. Since SPEED is assumed final, it must declare a value when it is initialized. Because there is no such default value, the code will not compile and option B is the correct answer.

19. 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 c3 = new Cardinal();
System.out.println(c1.number);
}
package bird;
}
```

- A 0
- B 1
- C 2
- D 3
- E 4
- ✓ F The code does not compile.

i If a package statement is present, it must be the first noncommented line of code in the file. If the package statement were omitted, the answer would be option D.

20. How many of the following lines give a compiler error?

```
1: public class _C {
2: String s = null;
3: static int 123;
4: void char = 'a';
5: byte b1 = 2;
6: int c1$ = b1;
7: short s1 = c1$;
8: long e_2 = c1$; }
```

- **A** 1
- **B** 2
- ✓ **C** 3
- **D** 4
- **E** 5

i Line 3 does not compile because variable names are not allowed to begin with a number. 123 is not a valid variable name. Line 4 does not compile because void is not allowed in a variable declaration. Also, there is no variable name on line 4. Line 7 does not compile because the short type is smaller than the int type and setting to a smaller type requires a cast. Three lines don't compile, so option C is correct.

21. What is the result of the following code?

```
public class Lion {
Lion l = new Lion();
static void public main(String[] args) {
new Lion();
}
public void roar() {
Lion l = new Lion();
if ( l == l) {
System.out.println("roar!");
} } }
```

- **A** Outputs nothing
- **B** Outputs roar!
- **C** The program never terminates.
- ✓ **D** The code does not compile.
- **E** The code throws an exception.

i The main() method does not compile because public is listed after the return type. It should be public static void main. If the code did compile, the answer would be option C because there is an instance initializer that calls the constructor. And the constructor calls the instance initializer and so forth until the program runs out of memory.

22. What is the output when new Buffalo() is called?

```
1: class Animal {
2: public Animal(int age) { System.out.println("1"); this(); }
3: public Animal() { System.out.println("2"); }
4: }
5: public class Buffalo extends Animal {
6: public Buffalo() {
7: System.out.println("3");
8: }
9: }
```

- **A** 123
- **B** 213
- **C** 3
- **D** 23
- ✓ **E** The code will not compile because of line 2.
- **F** The code will not compile because of line 7.
- **G** It compiles but throws an exception at runtime.

i The code may look complicated, but the answer is quite simple. On line 2, this() is used as the second statement of the constructor, which is not allowed. When this() or super() is used, it must be used as the first line of the constructor; therefore, the code does not compile and option E is correct. If we reverse the order of the statements in the constructor on line 2, then the code would compile with output 23, since the default no-argument constructor would be inserted before line 7.

23. When does the String object instantiated on line 3 become eligible for garbage collection?

```
1: public class ReturnDemo {
2: public static String getName() {
3: String temp = new String("Jane Doe");
4: return temp;
5: }
6: public static void main(String [] args) {
7: String result;
8: result = getName();
9: System.out.println(result);
10: result = null;
11: System.gc();
12: }
13: }
```

- **A** Immediately after line 4
- **B** Immediately after line 5
- **C** Immediately after line 9
- ✓ **D** Immediately after line 10
- **E** Immediately after line 11
- **F** Immediately after line 12
- **G** The code does not compile.

i The object on line 4 is referred to by the temp reference, which goes out of scope after line 5. However, the result reference gets a copy of temp, so it refers to the "Jane Doe" object until line 10 when result is set to null. At that point "Jane Doe" is no longer reachable and becomes immediately eligible for garbage collection. Calling System.gc() does not affect eligibility for garbage collection.

24. What is the output of the following code?

```
1: interface CanClimb { int maxHeight(); }
2: interface HasClaws { boolean isSharp(); }
3: public class Koala implements CanClimb, HasClaws {
4: public boolean isSharp() { return true; }
5: public int maxHeight() { return 15; }
6: public static void main(String[] args) {
7: Object koala = new Koala();
8: CanClimb canClimb = (CanClimb)koala;
9: HasClaws hasClaws = (HasClaws)canClimb;
10: System.out.print(canClimb.maxHeight());
11: System.out.print(hasClaws.isSharp());
12: }
13: }
```

- **A** true15
- ✓ **B** 15true
- **C** The code will not compile because of line 7.
- **D** The code will not compile because of line 8.
- **E** The code will not compile because of line 9.
- **F** The code compiles but throws an exception at runtime.

i The key here is that there is only one object created, but it is being accessed by multiple references of varying types. Line 7 compiles correctly since all objects can be implicitly cast to java.lang.Object, so option C is incorrect. Line 8 compiles without issue since CanClimb is a subclass of Object, so option D is incorrect. Line 9 also compiles correctly since they are related types, so option E is incorrect. At runtime, all of the casts succeed since Koala implements both CanClimb and HasClaws, so option F is incorrect. The code compiles and runs without issue, with 15true as the output.

25. What is the result of the following code?

```
1: public class DoubleLoopSample {
2: public static void main(String[] args) {
3: char c = 'a';
4: for(int i=1; i<= 3; i++)
5: for(int j=0; j<= 2; j++) System.out.print(c++);
6: } }
```

- ✓ **A** abcdefghi
- **B** bcdefghij
- **C** abcdef
- **D** abcabcabc
- **E** The code will not compile.

i The code compiles and runs without issue, so option E is incorrect. The outer loop executes three times and the inner loop executes three times, so you should first look for an answer in which 9 characters are printed. Because the post-increment operator was used on c, the first value printed is 'a', followed by 'b', and so on, up to 'i'. The output is abcdefghi, so the answer is option A.

26. What is the output of the following code?

```
LocalDate d = LocalDate.of(2015, 5, 7);
Period p = Period.of(1, 2, 3);
d = d.minus(p);
DateTimeFormatter f = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT);
System.out.println(f.format(d));
```

- A 3/7/14 11:22 AM
- B 6/7/14 11:22 AM
- C 5/10/15 11:22 AM
- D 6/10/15 11:22 AM
- E The code does not compile.
- ✓ F A runtime exception is thrown.

i Formatting a date object as a date and time is not allowed because there is no time. Therefore, a runtime exception is thrown.

27. What is the result of the following?

```
3: int[] times [] = new int[3][3];
4: for (int i = 0; i < times.length; i++)
5: for (int j = 0; j < times.length; j++)
6: times[i, j] = i*j;
7: System.out.println(times[2, 2]);
```

- A 1
- B 4
- C 1 printed 4 times
- D 4 printed 3 times
- E An exception is thrown.
- F The code fails to compile because of line 3.
- ✓ G The code fails to compile for another reason.

i The array indexes are incorrect. They should be times[i][j] and times[2][2]. If this were fixed, the answer would be option B. Note that there are no braces around the for loop. This means the print statement is not actual part of the loop and is only run once at the end.

28. Which of the following exist in Java? (Choose all that apply)

- A NullPointerError
- ✓ B NullPointerException
- C IOError
- ✓ D IOException
- ✓ E StackOverflowError
- F StackOverflowException

i NullPointerException is a runtime exception, IOException is a checked exception, and StackOverflowError is an error. You do need to be able to identify which categories these belong to.

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

```
1: public class Giraffe {
2: private int height;
3: private int age;
4: boolean tired;
5: public boolean isTired() { return tired; }
6: public int getHeight() { return height; }
7: public void setHeight(int h) { this.height = h; } }
```

A The data in this class is encapsulated.

B This class currently is immutable.

✓ C Making line 4 private would exhibit data encapsulation.

D Making line 4 private would exhibit immutability.

✓ E The method names follow JavaBean conventions.

i Data encapsulation and immutability both require private instance variables. Immutability does not allow setters. It is okay that no methods refer to *age*. Neither encapsulation nor immutability requires every instance variable be exposed. The method names are correct because they use is for the boolean getter, get for the other getter, and set for the setter.

30. What is the output of the following snippet?

```
31: Integer x = null;
32: int y = 0;
33: if(x == y) {
34: System.out.print("1");
35: } else {
36: System.out.print("2");
37: } finally {
38: System.out.print("3");
39: }
```

A 13

B 23

C 2 and the stack trace for a NullPointerException

D 23 and the stack trace for a NullPointerException

E 234

✓ F The code does not compile.

i A finally block can only appear in a try statement. It is not allowed at the end of an if statement. Therefore, this code does not compile.