**Answers to Assessment Test**

1. B. The code compiles successfully, so options D and E are incorrect. The value of a cannot be changed by the addToInt() method, no matter what the method does, because only a copy of the variable is passed into the parameter x. Therefore, a does not change, and the output on line 9 is 15. For more information, see [Chapter 7](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c07.xhtml).
2. C, D, E. The program contains three compiler errors. First, the method getTailLength() in the interface HasTail is implicitly to be public, since it is an abstract interface method. Therefore, line 3 does not compile since it is an invalid override, reducing the visibility of the method, making option C correct. Next, the class Cougar implements an overloaded version of getTailLength() with a different signature than the abstract interface method it inherits. For this reason, the declaration of Cougar is invalid, and option D is correct. Finally, option E is correct, since Puma is marked abstract and cannot be instantiated. For more information, see [Chapter 9](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c09.xhtml).
3. B. Initially, moon is assigned a value of 9, while star is assigned a value of 8. The multiplication operator (\*) has a higher order of precedence than the addition operator (+), so it gets evaluated first. Since star is not greater than 10, sun is assigned a value of 3, which is promoted to 3.0f as part of the assignment. The value of jupiter is (3.0f + 9) - 1.0, which is 11.0f. This value is implicitly promoted to double when it is assigned. In the last assignment, moon is predecremented from 9 to 8, with the value of the expression returned as 8. Since 8 is less than or equal to 8 is true, mars is set to a value of 2. The final output is 3.0-11.0-2, making option B the correct answer. Note that while Java outputs the decimal for both float and double values, it does not output the f for float values. For more information, see [Chapter 3](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c03.xhtml).
4. D. String literals are used from the string pool. This means that s1 and s2 refer to the same object and are equal. Therefore, the first two print statements print true. The concat() method forces a new String to be created making the third print statement print false. The intern() method reverts the String to the one from the string pool. Therefore, the fourth print statement prints true. The fifth statement print statement prints false because toString() uses a method to compute the value, and it is not from the string pool. The final print statement again prints true because equals() looks at the values of String objects. For more information, see [Chapter 5](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c05.xhtml).
5. E. The code does not compile because Java allows at most one public class in the same file. Either the FlightNumber class must not be declared public or it should be moved to its own source file named FlightNumber.java. The compiler error occurs on line 7, so the answer is option E. For more information, see [Chapter 1](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c01.xhtml).
6. G. This exam requires knowing how to run at the command line. The new -p option specifies the module path. The new -m option precedes the program to be run in the format moduleName/fullyQualifiedClassName. Option G is the only one that matches these requirements. For more information, see [Chapter 11](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c11.xhtml).
7. F. The code does not compile because switch statements require case statements before the colon (:). For example, case score1: would compile. For this reason, option F is the correct answer. If the six missing case statements were added throughout this snippet, then the default branch would be executed as 7 is not matched in any of the case statements, resulting in an output of great-good- and making option A correct. For more information, see [Chapter 4](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c04.xhtml).
8. A, C, F. The Predicate interface takes a single parameter and returns a boolean. Lambda expressions with one parameter are allowed to omit the parentheses around the parameter list, making options A and C equivalent and both correct. The return statement is optional when a single statement is in the body, making option F correct. Option B is incorrect because a return statement must be used if braces are included around the body. Options D and E are incorrect because the type is Integer in the predicate and int in the lambda. Autoboxing works for collections not inferring predicates. If these two were changed to Integer, they would be correct. For more information, see [Chapter 6](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c06.xhtml).
9. C. Option A is incorrect because var is only allowed as a type for local variables, not instance members. Options B and E are incorrect because new and case are reserved words and cannot be used as identifiers. Option C is correct, as var can be used as a method name. Option D is incorrect because a single underscore (\_) cannot be used as an identifier starting with Java 9. Finally, option F is incorrect because var cannot be specified as the return type of a method. For more information, see [Chapter 2](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c02.xhtml).
10. A, C, E. The for-each loop implicitly casts each Tadpole object to an Amphibian reference, which is permitted because Tadpole is a subclass of Amphibian. From there, any supertype of Amphibian is permitted without an explicit cast. This includes CanSwim, which Amphibian implements, and Object, which all classes extend from, making options A and E correct. Option C is also correct since the reference is being cast to the same type. Option B is incorrect, since Boolean is not a supertype of Amphibian. Option D is also incorrect. Even though the underlying object is a Tadpole instance, it requires an explicit cast on line 9 since the reference type is Amphibian. Option F is incorrect because there are options that allow the code to compile. For more information, see [Chapter 8](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c08.xhtml).
11. B, D. Option A does not compile, as the expression 3 + 2.0 is evaluated as a double, and a double requires an explicit cast to be assigned to an int. Option B compiles without issue, as a long value can be implicitly cast to a double. Option C does not compile because the ternary operator (? :) is missing a colon (:), followed by a second expression. Option D is correct. Even though the int value is larger than a short, it is implicitly cast to a short, which means the value will wrap around to fit in a short. Option E is incorrect, as you cannot use a decimal (.) with the long (L) postfix. Finally, option F is incorrect, as an underscore cannot be used next to a decimal point. For more information, see [Chapter 3](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c03.xhtml).
12. D. If this were a valid module-info.java file, it would need to be placed at the root directory of the module, which is option A. However, a module is not allowed to use the public access modifier. Option D is correct because the provided file does not compile regardless of placement in the project. For more information, see [Chapter 11](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c11.xhtml).
13. C. The code compiles and runs without issue; therefore, options E and F are incorrect. This type of problem is best examined one loop iteration at a time:
    * On the first iteration of the outer loop i is 0, so the loop continues.
    * On the first iteration of the inner loop, i is updated to 1 and x to 6. The if statement branch is not executed, and x is increased to 10 and j to 1.
    * On the second iteration of the inner loop (since j = 1 and 1 <= 2), i is updated to 2 and x to 11. At this point, the if branch will evaluate to true for the remainder of the program run, which causes the flow to break out of the inner loop each time it is reached.
    * On the second iteration of the outer loop (since i = 2), i is updated to 3 and x to 12. As before, the inner loop is broken since x is still greater than 10.
    * On the third iteration of the outer loop, the outer loop is broken, as i is already not less than 3. The most recent value of x, 12, is output, so the answer is option C.
    * For more information, see [Chapter 4](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c04.xhtml).
14. F. Line 5 does not compile as the FileNotFoundException thrown on line 12 is not handled or declared by the method. Line 7 does not compile because StringBuilder does not implement AutoCloseable and is therefore not compatible with a try-with-resource statement. Finally, line 10 does not compile as RuntimeException is a subclass of Exception in the multi-catch block, making it redundant. Since this method contains three compiler errors, option F is the correct answer. For more information, see [Chapter 10](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c10.xhtml).
15. A, C, F. The Java Development Kit (JDK) is used when creating Java programs. It contains a compiler since it is a development tool making option A correct and option B incorrect. The JDK contains a Java Virtual Machine (JVM) making option F correct and option E incorrect. The compiler creates bytecode making option C correct and option D incorrect. For more information, see [Chapter 1](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c01.xhtml).
16. C, E, F. The jump() method has default (package-private) access, which means it can be accessed only from the same package. Tadpole is not in the same package as Frog, causing lines 7 and 10 to give a compiler error, making options C and F correct. The ribbit() method has protected access, which means it can only be accessed from a subclass reference or in the same package. Line 6 is fine because Tadpole is a subclass. Line 9 does not compile because the variable reference is to a Frog, making option E correct. This is the trickiest question you can get on this topic on the exam. For more information, see [Chapter 7](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c07.xhtml).
17. C. The code compiles and runs without issue, so options G and H are incorrect. First, the Reindeer object is instantiated using the constructor that takes an int value. Since there is no explicit call to the parent constructor, the compiler inserts super()as the first line of the constructor on line 7. The parent constructor is called, and Deer is printed on line 2. The flow returns to the constructor on line 7, which prints Reindeer. Next, the method hasHorns() is called. The reference type is Deer, and the underlying object type is Reindeer. Since Reindeer correctly overrides the hasHorns() method, the version in Reindeer is called, printing true. For these reasons, option C is the correct answer. For more information, see [Chapter 8](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c08.xhtml).
18. B, E, F. The array is allowed to use an anonymous initializer because it is in the same line as the declaration. The ArrayList uses the diamond operator. This specifies the type matches the one on the left without having to retype it. After adding the two elements, list contains [6, 8]. We then replace the element at index 1 with 9, resulting in [6, 9]. Finally, we remove the element at index 0, leaving [9]and making option B correct. Option C is incorrect because arrays output something that looks like a reference rather than a nicely printed list of values.

Option E is correct because the compare() method returns 0 when the arrays are the same length and have the same elements. Option F is correct because the mismatch() method returns a -1 when the arrays are equivalent. For more information, see [Chapter 5](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c05.xhtml).

1. C, D. Immediately after line 8, only Grasshopper g1, created on line 6, is eligible for garbage collection since both one and two point to Grasshopper g2, making option C correct and option A incorrect. Immediately after line 9, we still only have Grasshopper g1 eligible for garbage collection, since one points to it. For this reason, option B is incorrect and option D is correct. Reference two now points to null. Immediately after line 10, both Grasshopper objects are eligible for garbage collection since both one and two point to null, making option E incorrect. The code does compile, so option F is incorrect. Although it is traditional to declare instance variables early in the class, you don’t have to. For more information, see [Chapter 2](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c02.xhtml).
2. B, C. Only checked exceptions are required to be handled or declared, making option B correct and option F incorrect. An Error is intended to be thrown by the JVM and never caught by the programmer, making option C correct and options A, D, and E incorrect. While a programmer could throw or catch an Error, this would be a horrible practice. For more information, see [Chapter 10](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c10.xhtml).
3. B, C, F. First, null and nonstatic are not valid method modifiers, making options A and D incorrect. Options B and F are correct, as abstract methods cannot be marked private or static, since they then would not be able to be overridden. Option C is also correct, as you cannot declare two access modifiers on the same method. Finally, options E and G are two sets of valid modifiers that can be used together in a method declaration. Using private with final is allowed, albeit redundant. For more information, see [Chapter 9](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c09.xhtml).
4. A. This is a great example to practice the process of elimination. The first thing to notice is that multiplier is not effectively final since it is reassigned. None of the lambdas will compile, making option A correct. The next step is to look at the lambda syntax. Options D and F are invalid because lambdas with more than one parameter must have parentheses. Options E and F are invalid because a return statement may not be used in a lambda without a block present. While option C at least compiles, the code fails at runtime because List.of() creates an immutable list. This is tricky as none of the lambdas will work successfully. Therefore, option A is the only correct answer. For more information, see [Chapter 6](file:///C:\Users\Denzo\AppData\Local\Temp\k0mecevx.2cy\OPS\c06.xhtml).