Chapter 10: OCA Practice Exam

1. E. The first time through the loop, we are calling `indexOf` on an empty `StringBuilder`. This returns `-1`. Since we cannot insert at index -1, the code throws a `StringIndexOutOfBoundsException`.
2. C, E. In Option A, the assignment operator `=` incorrectly comes after the addition `+` operator. In Option B, the addition operator `+` incorrectly comes after the division `/` operator. In Option D, the subtraction operator `-` incorrectly comes after the multiplication `*` operator. This leaves Options C and E as the correct answers. For these answers, it may help to remember that the modulus operator `%`, multiplication operator `*`, and division operator `/` have the same operator precedence.
3. B, C, F. Option A is incorrect because a getter should not take a value. Option D is incorrect because the prefix `is` should only be with `boolean` values. Option E is incorrect because `gimme` is not a valid JavaBean prefix. Options B, C, and F are each proper JavaBean method signatures.
4. A, E. Line 24 does not compile because arrays use `length`. It is `ArrayList` that uses `size()`. All of the other lines compile, making Option A correct. It is allowed to split up the braces in the 2D array declaration on line 20. The code is also allowed to use `crossword.length` as the loop condition on line 22, although this is not a good idea. The array starts out with all 200 of the cells initialized to the default value for an `int` of 0. Both loops iterate starting at 0 and stopping before 10, which causes only half of the array to be set to `'x'`. The other half still has the initial default value of 0, making Option E correct.
5. B, D. Options A and E are incorrect because they indicate states that the application can possibly recover from. An `Error` generally indicates an unrecoverable problem. While it is possible to catch an `Error`, it is strongly recommended that an application never do so, making Options B and D correct. Finally, Option C is incorrect because `Error` extends from `Throwable`, not `Exception`, and is unchecked.
6. A, C, D. The first `import` statement allows only the class `forest.Bird` to be available, making Option A correct and Options E and F incorrect. Option B is incorrect since the third `import` statement only allows access to classes within the `savana` package, not any sub-packages. Option C is correct because the second `import` statement allows any class in the `jungle.tree` package to be accessible. Finally, Option D is correct because `java.lang.*` is implicitly included in all Java classes.
7. C. *Mutable* means the object can change state. *Immutable* means the object cannot change state. An `ArrayList` stores a collection of objects. It mutates as the elements change. A `StringBuilder` is also mutable as it improves performance by not creating a new object each time it changes. A `String` is immutable. Methods that look like they change the value simply return a different `String` object. The date/time objects added in Java 8, such as `LocalDateTime`, are also immutable. Therefore, Option C is correct with `String` and `LocalDateTime` as the immutable object types.

8. C. On the first iteration through the loop, the first five characters are removed and `builder` becomes `s growing`. Since there are more than five characters left, the loop iterates again. This time, five more characters are removed and `builder` becomes `wing`. This matches Option C.

9. D. The code compiles without issue, so Option E is incorrect. The key here is that none of the variables are assigned the same object due to the use of the `new` keyword. Comparing any two variables with `==` will always result in an evaluation of `false`, making the first two values of the `print` statement be `false` and `false`. On the other hand, they all have an underlying `String` value equivalent to `up`, so calling `equals()` on any two variables will return `true`. Option D is the correct answer that matches what the application will print.

.0. C. Lines 4 and 5 both print `false` since a `String` should be compared with a method rather than `==`, especially when not comparing two values from the string pool. Line 6 also prints `false` because one value is uppercase and the other is lowercase. Line 7 prints `true` because both values are uppercase. Lines 8 and 9 print `true` because they don't look at the case. This makes Option C the answer.

l1. A, B, C. Let's look at each one in turn. Option A is correct because the labels are not referenced. Option B is correct because the outer `while` is broader than the inner `while`. Since there is no other code in the loop, it is not needed. Option C is also correct because a label is not used. Option D is incorrect because the inner loop is more specific than the outer loop. While the code still compiles, it prints one less chapter. Options E and F are incorrect because you cannot remove one half of a loop construct and have it compile.

.2. B, C. A `long` cannot contain a number with decimal points, preventing Options B and C from compiling. Options D and E show you can force a number to be a `long` by ending it with an upper- or lowercase `L`. This does not work if the number has a decimal point. Option F shows how to use underscores to break up a number.

.3. A. A `while` loop checks the condition before executing. Since the hour is not less than one, the loop never enters, and Option A is correct. This is good, because we'd have an infinite loop if the loop was entered since the result of `plusHours` is ignored.

.4. D. This question appears to ask you about involved array logic. Instead, it is checking to see if you remember that instance and class variables are initialized to `null`. Line 6 throws a `NullPointerException`. If the array was declared, the answer would be E because the code would throw an `ArrayStoreException` on line 8.

.5. C, E. The diamond operator is only allowed to be used when instantiating rather than declaring. In other words, it can't go on the left side of the equal sign. Therefore, Options B, D, and F are incorrect. The remaining three options compile. However, Option A produces a warning because generics are not used on the right side of the assignment operator. Therefore, Options C and E are correct. Option C is better than Option E since it uses the diamond operator rather than specifying a redundant type.

.6. B, D. At the end of the method, `shoe1` and `shoe3` both point to `"flip flop"`. `shoe2` points to `"croc"`. Since there are no references to `"sandal"`, it is eligible for garbage collection, making Option B correct. However, garage collection is not guaranteed to run, so Option D is also correct.

17. C. The code does not compile, so Options A and B are incorrect. The `getFish()` method is declared properly in the `Fish` class and successfully overridden in the `Clownfish` class. An overridden method must not declare any new or broader checked exceptions, but it is allowed to declare narrower exceptions or drop checked exceptions. The overridden method also uses a covariant return type. The use of `final` on the method and class declarations has no meaningful impact, since the methods and classes are not extended in this application. So where does the compilation error occur? In the `main()` method! Even though the `Clownfish` version of `getFish()` does not declare a checked exception, the call `f.getFish()` uses a `Fish` reference variable. Since the `Fish` reference variable is used and that version of the method declares a checked `Exception`, the compiler enforces that the checked exception must be handled by the `main()` method. Since this checked exception is not handled with a try-catch block nor by the `main()` method declaration, the code does not compile, and Option C is the correct answer.

.8. A. This is a correct example of using lambdas. The code creates an `ArrayList` with three elements. The `print()` method loops through and checks for negative numbers. Option A is correct.

.9. F. A `try` statement requires a `catch` or a `finally` block. It can also have both a `catch` and a `finally` block. Since no option matches these rules, Option F is the correct answer. Note that `finalize` is not a keyword but a method inherited from `java.lang.Object`. Lastly, the `throws` keyword can be applied to method declarations and is not used as part of a `try` statement.

:0. A. On line 12, `result` is first set to `8`. On line 13, the `boolean` condition is `true` because `8 > 7`. On line 13, `result` is incremented to `9`. Then the inner loop runs, decrementing `result` until it is no longer greater than `5`. On line 18, loop execution is completed because `result` is equal to `5`. The `break` statement says to skip to after the labeled loop, which is line `20`. Then `result` is printed as `5`, making Option A correct.

21. C. The code compiles and runs without exception, making Options E and F incorrect. The question is testing your knowledge of variable scope. The `teeth` variable is `static` in the `Alligator` class, meaning the same value is accessible from all instances of the class, including the `static main()` method. The `static` variable `teeth` is incremented each time the constructor is called. Since `teeth` is a local variable within the `snap()` method, the argument value is used, but changes to the local variable do not affect the `static` variable `teeth`. Since the local variable `teeth` is not used after it is decremented, the decrement operation has no meaningful effect on the program flow or the `static` variable `teeth`. Since the constructor is called twice, with `snap()` executed after each constructor call, the output printed is `1 2`, making Option C the

correct answer.

22. A. A `String` is immutable. Since the result of the `concat()` method call is ignored, the value of `witch` never changes. It stays as a single letter, and Option A is correct.
23. A, C, F. An interface method is exactly one of three types: `default`, `static`, or `abstract`. For this reason, Options A, C, and F are correct. An interface method cannot be `protected` nor `private` because the access modifier is always `public`, even when not specified, making Options B and D incorrect. Option E is also incorrect because `final` cannot be applied to `static` methods, since they cannot be overridden. It can also not be applied to `default` and `abstract` methods because they are always able to be overridden.
24. D. Look at the loop condition carefully. It tries to assign `null` to a `String` variable. This is not an expression that returns a `boolean`. Therefore, the code does not compile, and Option D is correct. If this was fixed by making the loop condition `tie == null`, then Option B would be correct.
25. B, F. A class may be defined without a `package` statement, making the class part of the default package. For this reason, Options A and D are incorrect. Every Java class implicitly imports exactly one package, `java.lang.*`, making Option B correct and Option C incorrect. Option E is incorrect because an `import` statement is not required. Finally, Option F is correct; any class that does not extend another class implicitly extends `java.lang.Object`.
26. D. A class cannot inherit two interfaces that declare the same `default` method, unless the class overrides them. In this case, the version of `grow()` in the `Tree` class is an overloaded method, not an overridden one. Therefore, the code does not compile due to the declaration of `Tree` on line `m1`, and Option D is the correct answer.
27. D. Variables are allowed to start with an underscore and are allowed to contain a $. Therefore, all the variable declarations compile, making Options A, B, and C incorrect. However, the `println()` refers to the uninitialized local `boolean`. Since local variables are not automatically initialized, the code does not compile, and Option D is correct.
28. A. Prefix operators, such as `--x` and `++x`, modify the variable and evaluate to the new value, while postfix operators, such as `x--` and `x++`, modify the variable but return the original value. Therefore, Option A is the correct answer.
29. B, C, E. The constructors declared by Options A, D, and F compile without issue. Option B does not compile. Since there is no call to a parent constructor or constructor in the same class, the compiler inserts a no-argument `super()` call as the first line of the constructor. Because `Big` does not have a no-argument constructor, the no-argument constructor `Trouble()`does not compile. Option C also does not compile because `super()` and `this()` cannot be called in the same constructor. Note that if the `super()` statement was removed, it would still not compile since this would be a recursive constructor call. Finally, Option E does not compile. There is no matching constructor that can take a `String` followed by a `long` value. If the input argument `deep`

was an `int` in this constructor, then it would match the constructor used in Option D and compile without issue.

30. E, F. A `static` method is not allowed to access instance variables without an instance of the class, making Options E and F correct. Notice that only `max` is initialized to `100` in Option E. Since `min` doesn't have a value specified, it gets the default value, which is `0`.

31. B, E. The ternary `? :` operator only evaluates one of the two right-hand expressions at runtime, so Option A is incorrect. A `switch` statement may contain at most one optional `default` statement, making Option B correct. A single if-then statement can have at most one `else` statement, so Option C is incorrect. Note that you can join if-then-else statements together, but each `else` requires an additional if-then statement. The disjunctive `|` operator will always evaluate both operands, while the disjunctive short-circuit `||` operator will only evaluate the right-hand side of the expression if the left-hand side evaluates to `false`. Therefore, they are not interchangeable, especially if the right-hand side of the expression modifies a variable. For this reason, Option D is incorrect. Finally, Option E is correct. The logical complement `!` operator may only be applied to `boolean` expressions, not numeric ones.

32. C. Line 3 creates an empty `StringBuilder`. Line 4 adds three characters to it. Line 5 removes the first character resulting in `ed`. Line 6 deletes the characters starting at position 1 and ending right before position 1. Since there are no indexes that meet that description, the line has no effect. Therefore, Option C is correct.

33. A, D. Java methods must start with a letter, the dollar `$` symbol, or the underscore `_` character. For this reason, Option B is incorrect, and Options A and D are correct. Despite how Option A looks, it is a valid method signature in Java. Options C, E, and F do not compile because the symbols `-`, `\`, and `#` are not allowed in method names, respectively.

34. B, C. First off, Option A is incorrect, since whether or not `static` or inherited methods are chosen is a matter of design and individual preference. Options B and C are true statements about inheritance and two of the most important reasons Java supports inheritance. Option D is incorrect because all Java classes extend `java.lang.Object`. Option E is incorrect. Whether or not inheritance simplifies or complicates a design is based on the skills of the developer creating the application.

35. E. All of the statements are true statements about Java, making Option E the correct answer. Java was built with object-oriented programming and polymorphism in mind. Also, Java supports functional programming using lambda expressions.

36. C. This array has three elements, making `listing.length` output `3`. It so happens that each element references an array of the same size. But the code checks the first element and sees it is an array of size two, making the answer Option C.

37. A, B, E. A `switch` statement supports the primitive types `byte`, `short`, `char`, and `int` and their associated wrapper classes `Character`, `Byte`, `Short`, and `Integer`. It also supports

the `String` class and enumerated types. Floating-point types like `float` and `double` are not supported, nor is the `Object` class. For these reasons, Options A, B, and E are correct.

8. D. The lambda syntax is incorrect. It should be `->`, not `=>`. Therefore, Option D is correct. If this was fixed, Option A would be correct.

9. B, C, E. The `/* */` syntax can have additional (and uneven) `*` characters in Java, making Options B and E correct. Option C is the standard way to comment a single line with two slashes `//`. Option A contains a `*/` in the middle of the expected comment, making the part after the comment `Insert **/` invalid. Option D is incorrect because a single slash `/` is not valid comment in Java. Finally, the `#` is not a comment character in Java, so Option F is incorrect.

0. A, F. A `static` import is used to import `static` members of another class. Option A is correct because the method `getGrass` and variable `seeds` are imported. Option F is also correct because a wildcard on the `Grass` class for all visible `static` members is allowed. Option B is incorrect because the wildcard must be on a class, not a package. Options C and E are incorrect since the keywords `import` and `static` are reversed. Option D is incorrect because the `static` keyword is missing.

1. D. When converting an array to a `List`, Java uses a fixed-sized backed list. This means that the list uses an array in the implementation. While changing elements to new values is allowed, adding and removing elements is not.

2. A, D. Variable names can begin with an underscore, making Option A correct. To use an underscore in a numeric literal, it must be between two digits, making Option D correct.

3. B. While no arguments are passed from the command line, this doesn't matter because the `main()` method redefines the `args` array. Remember that `String` values sort alphabetically rather than by number. Therefore, `01` sorts before `1`, and Option B is correct.

4. D. The `public` modifier allows access members in the same class, package, subclass, or even classes in other packages, while the `static` modifier allows access without an instance of the class. For these reasons, Option D is the correct answer. Option A is incorrect because `final` is not related to access, and package-private prevents access from classes outside the package. Option B is incorrect because `class` is not a modifier; it is a keyword. Option C is incorrect because `instance` is not a Java keyword or modifier, and `protected` prevents classes that are not subclasses and are outside the package from accessing the variable. Finally, Option E is incorrect. The `default` keyword is for interface methods and `switch` statements, not class variables.

5. A. Looping through the same list multiple times is allowed. Notice how there are not braces around the loops. This means that only the `print` statement is inside the loop. It executes four times. However, the `println()` only executes once at the end, making Option A the answer.

46. C, D. The `javac` command compiles a `.java` file into a `.class` bytecode file, making Option C a correct answer, while also making Options B, E, and F incorrect. The `javac` command compiles to a set of java instructions, or bytecode, not machine instructions, making Option A incorrect and Option D correct.

47. C. The `parseInt()` method returns an `int` primitive. Thanks to autoboxing, we can also assign it to an `Integer` wrapper class object reference. The `char` and `short` types are smaller than `int` so they cannot store the result. Therefore, lines 3 and 4 compile, and Option C is correct.

48. B, D, F. The compiler will broaden the data type on a numeric value until it finds a compatible signature. There are two versions of the `drive()` methods that return a value of `3`, one that takes a `short` and one that takes a `double`. Option A is incorrect because `boolean` cannot be converted to either of these types and trying to do so triggers a compiler error. Option B is correct because the data type `short` matches our message signature. Options C and E are incorrect. Remember that `int` and `long` are larger than `short` and will trigger different overloaded versions of `drive()` to be called, one that returns `5` and one that returns `2`, respectively. Option D is correct. The `byte` value can be implicitly converted to `short`, and there are no other matching method signatures that take a `byte` value. Finally, Option F is correct because `float` can be implicitly converted to `double`, and there is no other version of `drive()` that takes a `float` value.

49. A. Trick question. This appears to be about equality, but it is really about you recognizing that the `main()` method is missing the `static` keyword. Running this problem gives a runtime exception because the `main()` method is not properly declared. Therefore, Option A is the answer. If this was fixed, the answer would be Option C because the `int` and `String` comparisons return `true`.

50. D. The code compiles without issue, so Options E and F are incorrect. Note that line `p2` accesses a `static` method using an instance reference, which is discouraged but permitted in Java. First, a varargs `int` array of `[0,0]` is passed to the `swing()` method. The `try` block throws `ArrayIndexOutOfBoundsException`, since the third element is requested and the size of the array is two. For this reason, the `print()` statement in the `try` block is not executed. Next, since `ArrayIndexOutOfBoundsException` is a subclass of `RuntimeException`, the `RuntimeException catch` block is executed and `2` is printed. The rest of the `catch` blocks are skipped, since the first one was selected. The `finally` block then executes and prints `4`. Lastly, control is returned to the `main()` method without an exception being thrown, and `5` is printed. Since `245` is printed, Option D is the correct answer.

51. E. In the first iteration through the loop, `container` is 2 and `cup` is printed. Notice how the loop body subtracts 1 to account for indexes being zero based in Java. Then the update statement runs, setting `container` to 3. The condition is run and sees that 3 is in fact greater than 0. The loop body subtracts 1 and tries to get the element at index 2. There isn't one and the code throws an exception. This makes Option E correct. You

might be tempted to think this is an infinite loop. If the body did not throw an exception, it would be!

2. A, E, F. An entry point in a Java application consists of a `main()` method with a single `String[]` or vararg `String...` argument, return type of `void`, and modifiers `public` and `static`. Note that the name of the variable in the input argument does not matter and the `final` modifier is optional. Options A, E, and F match this description and are correct. Option B is incorrect because the argument is a single `String`. Option C is incorrect, since the access modifier is incorrectly marked `protected`. Finally, Option D is incorrect because it has two return types, `int` and `void`.

3. C, D, E. For this question, it helps to remember that the value of a `case` statement must be a literal expression or a `final` constant variable, and have a compatible data type. For these reasons, Lines 10 and 12 do not compile, making Options C and E correct answers. Line 10 uses a constant value, but `long` is not compatible with `switch` statements, while Line 12 uses a variable that is not marked `final`. Next, a `switch` statement may only have one `default` block. Therefore, Line 11 or 14 must be removed. Since Line 14 is not in the list of options, Option D becomes the last correct answer. The rest of the lines are fine since removing Lines 10, 11, and 12 allows the code to compile.

4. A, B, C. All of the compilation issues with this code involve access modifiers. First, all interface methods are implicitly `public`, and explicitly setting an interface method to `protected` causes a compilation error on line `h1`, making Option A correct. Next, lines `h2` and `h3` both override the interface method with the package-private access modifier. Since this reduces the implied visibility of `public`, the overrides are invalid and neither line compiles. Therefore, Options B and C are also correct. Note that the `RuntimeException` is allowed in an overridden method even though it is not in the parent method signature because only new checked exceptions in overridden methods cause compilation errors. Line `h4` is valid. An object can be implicitly cast to a superclass or inherited interface. Finally, lines `h5` and `h6` will compile without issue but independently throw a `ClassCastException` and a `NullPointerException` at runtime, respectively. Since the question only asks about compilation problems, neither of these are correct answers.

5. B, E, F. Unchecked exceptions inherit the `RuntimeException` class and are not required to be caught in the methods where they are declared. Since `ArithmeticException` and `IllegalArgumentException` extend `RuntimeException`, they are included as unchecked exceptions, making Options B, E, and F correct. The rest are checked exceptions, which inherit `Exception` but not `RuntimeException`.

6. F. The code compiles without issue, making Options D and E incorrect. Applying the ternary `? :` operator, the variable `ship` is assigned a value of `10.0`. The expression in the first if-then statement evaluates to `true`, so `Goodbye` is printed. Note that there is no `else` statement between the first and second if-then statements, therefore the second if-then statement is also executed. The expression in the second if-then

statement evaluates to `false`, so the `else` statement is called and `See you again` is also printed. Therefore, Option F is the correct answer, with two statements being printed.

57. B, C, D. The `clock` variable is accessed by a class in the same package; therefore, it requires package-private or less restrictive access (`protected` and `public`). The `getTime()` method is accessed by a subclass in a different package; therefore, it requires `protected` or less restrictive access (`public`). Options B, C, and D conform to these rules, making them the correct answer. Options A and F cause the `Snooze` class to fail to compile because the `getTime()` method is not accessible outside the package, even though `Snooze` is a subclass of `Alarm`. Option E causes the `Coffee` class to fail to compile because the `clock` variable is only visible within the `Alarm` class.

58. B. This problem appears to be to be about overriding a method, but in fact, it is much simpler. The class `CarbonStructure` is not declared `abstract`, yet it includes an `abstract` method. To fix it, the definition of `CarbonStructure` would have to be changed to be an `abstract` class, or the `abstract` modifier would need to be removed from `getCount()` in `CarbonStructure` and a method body added. Since the only answer choice available is to change the `getCount()` method on line `q1`, Option B is the correct answer. Note that the rest of the application, including the override on line `q2`, is correct and compiles without issue. The return types `Long` and `Number` are covariant since `Number` is a superclass of `Long`. Likewise, the exception thrown in the subclass method is narrower, so no compilation error occurs on `q2`.

59. C. Line 5 does not declare a `main()` method that can be the entry point to the program. It does correctly declare a regular instance method and does compile. Line 6 does not compile because `LocalDate` needs to use a static method rather than a constructor. Line 7 is incorrect because `Period` methods should not be chained. However, it does compile, returning a period of 1 day. Line 8 does not compile because the correct class name is `DateTimeFormatter`. Line 9 is correct. Option C is correct because lines 6 and 8 do not compile.

60. A, E. A `try` block can have zero or more `catch` blocks, and zero or one `finally` blocks, but must be accompanied by at least one of these blocks. For these reasons, Options B, D, and F are incorrect, and Option E is correct. A `finally` block must appear after the last `catch` block, if there are any, making Option C incorrect, and Option A correct.

61. B. The code compiles without issue, so Option E is incorrect. For this problem, it helps to remember that `+` and `*` have a higher precedence than the ternary `? :` operator. In the first expression, `1 + 2 * 5` is evaluated first, resulting in a reduction to `11>=2 ? 4 : 2`, and then `fish` being assigned a value of `4`. In the second expression, the first ternary expression evaluates to false resulting in a reduction to the second right-hand expression `5>=5 ? 9 : 7`, which then assigns a value of `9` to `mammals`. In the `print()` statement, the first `+` operator is an addition operator, since the operands are numbers, resulting in the value of `4 + 9`, `13`. The second `+` operator is a concatenation since one of the two operands is a `String`. The result `13` is printed, making Option B the correct answer.

2. A, C, E. An object can be cast to a superclass or inherited interface type without an explicit cast. Furthermore, casting an object to a reference variable does not modify the object in any way; it just may change what methods and variables are immediately accessible. For these reasons, Options A, C, and E are correct. Option B is incorrect; since the compiler can try to block or warn about invalid casts, it cannot prevent them. For example, any object can be implicitly cast to `java.lang.Object`, then explicitly cast to any other object, leading to a `ClassCastException` at runtime. Option D is also incorrect because assigning an object to a subclass reference variable requires an explicit cast. Finally, Option F is incorrect. An object can always be cast to one of its inherited types, superclass or interface, without a `ClassCastException` being thrown.

3. F. The array is not sorted. It does not meet the pre-condition for a binary search. Therefore, the output is not guaranteed and the answer is Option F.

4. B. While `shoe3` goes out of scope after the `shopping()` method, the `croc` object is referenced by `shoe1` and therefore cannot be garbage collected. Similarly, the `sandal` object is now referenced by `shoe2`. No variables reference the `flip flop` object, so it is eligible to be garbage collected, and Option B is correct.

5. E. The `throws` keyword is used in method declarations, while the `throw` keyword is used to throw an exception to the surrounding process, and the `finally` keyword is used to add a statement that is guaranteed to execute even if an exception is thrown. For these reasons, Option E is the correct answer.

6. B, E. The first two iterations through the loop complete successfully, making Option B correct. However, the two arrays are not the same size and the `for` loop only checks the size of the first one. The third iteration throws an `ArrayIndexOutOfBoundsException`, making Option E correct.

7. E. For this question, it helps to try all answers out. Most of them do not make any sense. For example, overloading a method is not a facet of inheritance. Likewise, concrete and abstract methods can both be overridden, not just one. The only answer that is valid is Option E. Without virtual methods, overriding a method would not be possible, and Java would not truly support polymorphism.

8. E. The code does compile. Line `s1` is a bit tricky because `length` is used for an array and `length()` is used for a `String`. Line `s1` stores the length of the `Fall` in a variable, which is 4. Line `s2` throws an `ArrayIndexOutOfBoundsException` because 4 is not a valid index for an array with four elements. Remember that indices start counting with zero. Therefore, Option E is correct.

9. D. The code definitely does not compile, so Option A is incorrect. The first problem with this code is that the `Drum` class is missing a constructor causing the class declaration on line 8 to fail to compile. The default no-argument constructor cannot be inserted if the superclass, `Instrument`, does not define a no-argument constructor. The second problem with the code is that line 11 does not compile, since it calls `super.play(5)`, but the version of `play()` in the parent class does not take any

arguments. Finally, line 15 does not compile. While `mn` may be a reference variable that points to a `Drum()` object, the `concert()` method cannot be called unless it is explicitly cast back to a `Drum` reference. For these three reasons, the code does not compile, and Option D is the correct answer.

70. B. The application compiles and runs without issue, so Options E and F are incorrect. Java uses pass-by-value, so even though the change to `length` in the first line of the `adjustPropellers()` method does not change the value in the `main()` method, the value is later returned by the method and used to reassign the `length` value. The result is that `length` is assigned a value of `6`, due to it being returned by the method. For the second parameter, while the `String[]` reference cannot be modified to impact the reference in the calling method, the data in it can be. Therefore, the value of the first element is set to `LONG`, resulting in an output of `6,LONG`, making Option B the correct answer.

71. D. The first compilation problem with the code is that the second `catch` block in `openDrawbridge()` is unreachable since `CableSnapException` is a subclass of `OpenDoorException`. The `catch` blocks should be ordered with the more narrow exception classes before the broader ones. Next, the variable `ex` is declared twice within the same scope since it appears in the second `catch` block as well as the embedded try-catch block. Finally, the `openDrawbridge()` method declares the checked `Exception` class, but it is not handled in the `main()` method with a try-catch block, nor in the `main()` method declaration. For these three reasons, Option D is correct.

72. D. Object orientation is the property of structuring an object with its related data and methods. Encapsulation is the property of removing direct access to the underlying data from processes outside the class. The two go hand and hand to improve class design, making Option D the correct choice.

73. E. In Java, `String` is a class and not a primitive. This means it needs to begin with an uppercase letter in the declaration. The code does not compile, making Option E correct. If this was fixed, the answer would be Option B.

74. A. This class is called with three command-line arguments. First the array is sorted, which meets the pre-condition for binary search. At this point, the array contains `[flower, plant, seed]`. The key is to notice the value of `args[0]` is now `flower` rather than `seed`. Calling binary search to find the position of `flower` returns `0`, which is the index matching that value. Therefore, the answer is Option A.

75. B, C, D. A `for`-each loop is a specialized loop that just iterates through an array or list. It can be rewritten using explicit indexing code in any of the other three loop types. Therefore, Options B, C, and D are correct. Option A is incorrect because a `do-while` loop is guaranteed to execute at least once. Option E is incorrect because the traditional `for` loop can loop backwards or by skipping indexes. Option F is incorrect because non-index-related `boolean` conditions are allowed to be used in a `while` loop.

76. E. The `LocalDate` class is only for day/month/year values. It does not support time, so

`getHour()` and `plusHours()` do not compile, making Option E the answer.

'7. C. All arrays are objects regardless of whether they point to primitives or classes. That means both `balls` and `scores` are objects. Both are set to `null` so they are eligible for garbage collection. The `balls` array is initialized to all `null` references. There are no objects inside. The `scores` array is initialized to all 0 values. Therefore, only two objects exist to be eligible for garbage collection, and Option C is correct.

'8. B. Since there are not brackets around the `while` loop, only line 17 is in the loop body. Line 18 gets executed once after the loop completes. This means that `count` will be 1 assuming the loop completes. Subtracting a month from `JANUARY` results in `DECEMBER`. Since the loop completes E is incorrect and Option B is the answer. Note that if the brackets were added as the indentation suggests, Option D would be the answer since we are counting months backwards.

'9. D. Line 10 does not compile because the override reduces the visibility of an inherited method, with the package-private modifier being more restrictive than the `protected` modifier. Line 11 does also not compile, since the left-hand side of a compound assignment operator must be used with a variable, not a method. Finally, Line 12 does not compile because `super.grunt()` is inherited as an `abstract` method in the `PolarBear` class, meaning the parent class has no implementation. For these three reasons, Option D is the correct answer.

'0. B, E. Package-private, or default, access is denoted by the absence of an access modifier, making Option A incorrect. Option B is correct, since a `switch` statement can contain a `default` execution path. Options C and F are incorrect because keywords in Java cannot be used as method or variable names. Finally, interfaces can contain `default` interface methods but they must be concrete with a method body. For this reason, Option E is correct and Option D is incorrect.