Chapter 6: Working with Methods and Encapsulation

1. C. The `protected` modifier allows access by subclasses and members within the same package, while the package-private modifier allows access only to members in the same package. Therefore, the `protected` access modifier allows access to everything the package-private access modifier, plus subclasses, making Option C the correct answer. Options A, B, and D are incorrect because the first term is a more restrictive access modifier than the second term.
2. B. The `super()` statement is used to call a constructor in a parent class, while the `this()` statement is used to call a constructor in the same class, making Option B correct and Option A incorrect. Options C and D are incorrect because they are not constructors.
3. D. The `sell()` method does not compile because it does not return a value if both of the if-then statements' conditional expressions evaluate to `false`. While logically, it is true that `price` is either less than `10` or greater than or equal to `10`, the compiler does not know that. It just knows that if both if-then statements evaluate to `false`, then it does not have a return value, therefore it does not compile.
4. D. The three overloaded versions of `nested()` compile without issue, since each method takes a different set of input arguments, making Options B and C incorrect. The code does not compile, though, due to the first line of the `main()` method, making Option A incorrect. The no-argument version of the `nested()` method does not return a value, and trying to output a `void` return type in the `print()` method throws an exception at runtime.
5. B. Java uses pass-by-value to copy primitives and references of objects into a method. That means changes to the primitive value or reference in the method are not carried to the calling method. That said, the data within an object can change, just not the original reference itself. Therefore, Option B is the correct answer, and Options C and D are incorrect. Option A is not a real term.
6. C. Option A is incorrect because the getter should return a value. Option B is incorrect because the setter should take a value. Option D is incorrect because the setter should start with `set` and should not return a value. Option C is a correct setter declaration because it takes a value, uses the `void` return type, and uses the correct naming convention.
7. B. Options A, C, and D are true statements about calling `this()` inside a constructor. Option B is incorrect because a constructor can only call `this()` or `super()` on the first line of the constructor, but never both in the same constructor. If both constructors were allowed to be called, there would be two separate calls to `super()`, leading to duplicate initialization of parent constructors, since the other constructor referenced by `this()` would also call `super()` (or be chained to one that eventually calls `super()`).
8. B. Option A is incorrect because the `public` access modifier starts with a lowercase

letter. Options C and D are incorrect because the return types, `void` and `String`, are incompatible with the method body that returns an integer value of `10`. Option B is correct and has package-private access. It also uses a return type of `Long` that the integer value of `10` can be easily assigned to without an explicit cast.

9. C. The only variables always available to all instances of the class are those declared `static`; therefore, Option C is the correct answer. Option A may seem correct, but `public` variables are only available if a reference to the object is maintained among all instances. Option B is incorrect because there is no `local` keyword in Java. Option D is also incorrect because a `private` instance variable is only accessible within the instance that created it.

10. A. First off, all of the lines compile but they produce various different results. Remember that the default initialization of a `boolean` instance variable is `false`, making `outside false` at line `p1`. Therefore, `this(4)` will cause `rope` to be set to `5`, while `this(5)` will cause `rope` to be set to `6`. Since `5` is the number we are looking for, Option A is correct, and Option C is incorrect. Option B is incorrect. While the statement does create a new instance of `Jump`, with `rope` having a value of `5`, that instance is nested and the value of `rope` does not affect the surrounding `instance` of `Jump` that the constructor was called in. Option D is also incorrect. The value assigned to `rope` is `4`, not the target `5`.

11. B. Options A, C, and D are true statements. In particular, Option C allows us to write the `equals()` methods between two objects that compare `private` attributes of the class. Option D is true because `protected` access also provides package-private access. Option B is false. Package-private attributes are only visible if the two classes are in the same package, regardless of whether one extends the other.

12. D. The class data, `stuff`, is declared `public`, allowing any class to modify the `stuff` variable and making the implementation inherently unsafe for encapsulation. Therefore, there are no values that can be placed in the two blanks to ensure the class properly encapsulates its data, making Option D correct. Note that if `stuff` was declared `private`, Options A, B, and C would all be correct. Encapsulation does not require JavaBean syntax, just that the internal attributes are protected from outside access, which all of these sets of values do achieve.

13. C. Option A is incorrect because Java only inserts a no-argument constructor if there are no other constructors in the class. Option B is incorrect because the parent can have a default no-argument constructor, which is inserted by the compiler and accessible in the child class. Finally, Option D is incorrect. A class that contains two no-argument constructors will not compile because they would have the same signature. Finally, Option C is correct. If a class extends a parent class that does not include a no-argument constructor, the default no-argument constructor cannot be automatically inserted into the child class by the compiler. Instead, the developer must explicitly declare at least one constructor and explicitly define how the call to the parent constructor is made.

14. A. A method may contain at most one varargs parameter, and it must appear as the last argument in the list. For this reason, Option A is correct, and Options B, C, and D are incorrect.

15. C. To solve this problem, it helps to remember that Java is a pass-by-value language in which copies of primitives and object references are sent to methods. This also means that an object's data can be modified within a method and shared with the caller, but not the reference to the object. Any changes to the object's reference within the method are not carried over to the caller. In the `slalom()` method, the `Ski` object is updated with an `age` value of `18`. Although, the last line of the `slalom()` method changes the variable value to `null`, it does not affect the `mySkier` object or reference in the `main()` method. Therefore, the `mySkier` object is not `null` and the `age` variable is set to `18`, making Options A and D incorrect. Next, the `name` variable is reassigned to the `Wendy` object, but this does not change the reference in the `main()` method, so `myName` remains `Rosie`. Finally, the `speed` array is assigned a new object and updated. Since the array is updated after the reference is reassigned, it does not affect the `mySpeed` array in the `main()` method. The result is that `mySpeed` continues to have a single element with the default `int` value of `0`. For these reasons, Option B is incorrect, and Option C is correct.

16. B. Options A and D would not allow the class to compile because two methods in the class cannot have the same name and arguments, but a different return value. Option C would allow the class to compile, but it is not a valid overloaded form of our `findAverage()` method since it uses a different method name. Option B is a valid overloaded version of the `findAverage()` method, since the name is the same but the argument list differs.

17. D. Implementing encapsulation prevents internal attributes of a class from being modified directly, so Option C is a true statement. By preventing access to internal attributes, we can also maintain class data integrity between elements, making Option B a true statement. Option A is also a true statement about encapsulation, since well-encapsulated classes are often easier to use. Option D is an incorrect statement. Encapsulation makes no guarantees about performance and concurrency.

18. A. Option B is incorrect because `String` values are immutable and cannot be modified. Options C and D are also incorrect since variables are passed by value, not reference, in Java. Option A is the correct answer. The contents of an array can be modified when passed to a method, since a copy of the reference to the object is passed. For example, the method can change the first element of a non-empty array.

19. B. Option A is not a valid syntax in Java. Option C would be correct if there was a `static` import, but the question specifically says there are not any. Option D is almost correct, since it is a way to call the method, but the question asks for the best way to call the method. In that regard, Option B is the best way to call the method, since we are given that two classes are in the same package, therefore the package name would not be required.

20. D. Options A and B are incorrect because a method with a non-`void` return type requires that the method return a value using the `return` statement. Option C is also incorrect since a method with a `void` return type can still call the `return` command with no values and exit the method. Therefore, Option D is the correct answer.

21. C. The `finish()` method modifies two variables that are marked `final`, `score` and `result`. The `score` variable is modified by the post-increment `++` operator, while the `result` variable is modified by the compound addition `+=` operator. Removing both `final` modifiers allows the code to compile. For this reason, Option C is the correct answer.

22. D. The `super()` statement is used to call a constructor in the parent class, while `super` is used to reference a member of the parent class. The `this()` statement is used to call a constructor in the current class, while `this` is used to reference a member of the current class. For these reasons, Option D is the correct answer.

23. B. The method signature has package-private, or default, access; therefore, it is accessible to classes in the same package, making Option B the correct answer.

24. A. The access modifier of `strength` is `protected`, meaning subclasses and classes within the same package can modify it. Changing the value to `private` would improve encapsulation by making the `Protect` class the only one capable of directly modifying it. For these reasons, the first statement is correct. Alternatively, the second and third statements do not improve the encapsulation of the class. While having getters and setters for `private` variables is helpful, they are not required. Encapsulation is about protecting the data elements. With this in mind, it is clear the `material` variable is already protected. Therefore, Option A is the correct answer.

25. A. Option A is correct since method names may include the underscore _ character as well as the dollar `$` symbol. Note that there is no rule that requires a method start with a lowercase character; it is just a practice adopted by the community. Option B is incorrect because the hyphen - character may not be part of a method name. Option C is incorrect since `new` is a reserved word in Java. Finally, Option D is incorrect. A method name must start with a letter, the dollar `$` symbol, or an underscore _ character.

26. D. The code does not compile, regardless of what is inserted into the line because the method signature is invalid. The return type, `int`, should go before the method name and after any access, `final`, or `static` modifiers. Therefore, Option D is the correct answer. If the method was fixed, by swapping the order of `int` and `static` in the method declaration, then Option C would be the correct answer. Options A and B are still incorrect, though, since each uses a return type that cannot be implicitly converted to `int`.

27. B. Java uses pass-by-value, so changes made to primitive values and object references passed to a method are not reflected in the calling method. For this reason, Options A and C are incorrect statements. Option D is also an invalid statement because it is a

special case of Option A. Finally, Option B is the correct answer. Changes to the data within an object are visible to the calling method since the object that the copied reference points to is the same.

28. C. The code contains a compilation problem in regard to the `contents` instance variable. The `contents` instance variable is marked `final`, but there is a `setContents()` instance method that can change the value of the variable. Since these two are incompatible, the code does not compile, and Option C is correct. If the `final` modifier was removed from the `contents` variable declaration, then the expected output would be of the form shown in Option A.

29. A. JavaBean methods use the prefixes `get`, `set`, and `is` for `boolean` values, making Option A the correct choice.

30. C. Option A is incorrect because the keywords `static` and `import` are reversed. The `Closet` class uses the method `getClothes()` without a reference to the class name `Store`, therefore a `static` import is required. For this reason, Option B is incorrect since it is missing the `static` keyword. Option D is also incorrect since `static` imports are used with members of the class, not a class name. Finally, Option C is the correct answer since it properly imports the method into the class using a `static` import.

31. D. In Java, the lack of an access modifier indicates that the member is package-private, therefore Option D is correct. Note that the `default` keyword is used for interfaces and `switch` statements, and is not an access modifier.

32. B. The code does not compile, so Option A is incorrect. The class contains two constructors and one method. The first method, `Stars()`, looks a lot like a no-argument constructor, but since it has a return value of `void`, it is a method, not a constructor. Since only constructors can call `super()`, the code does not compile due to this line. The only constructor in this class, which takes an `int` value as input, performs a pointless assignment, assigning a variable to itself. While this assignment has no effect, it does not prevent the code from compiling. Finally, the `main()` method compiles without issue since we just inserted the full package name into the class constructor call. This is how a class that does not use an `import` statement could call the constructor. Since the method is in the same class, and therefore the same package, it is redundant to include the package name but not disallowed. Because only one line causes the class to fail to compile, Option B is correct.

33. A. An instance method or constructor has access to all `static` variables, making Option A correct. On the other hand, `static` methods and `static` initializers cannot reference instance variables since they are defined across all instances, making Options B and C incorrect. Note that they can access instance variables if they are passed a reference to a specific instance, but not in the general case. Finally, Option D is incorrect because `static final` variables must be set when they are declared or in a `static` initialization block.

34. B. The method `calculateDistance()` requires a return type that can be easily

converted to a `short` value. Options A, C, and D are incorrect because they each use a larger data type that requires an explicit cast. Option D also does not compile because the `Short` constructor requires an explicit cast to convert the value of `4`, which is assumed to be an `int`, to a `short`, as shown in `new Short((short)4)`. Option B is the correct answer since a `byte` value can be easily promoted to `short` and returned by the method.

35. C. Overloaded methods have the same name but a different list of parameters, making the first and third statements true. The second statement is false, since overloaded methods can have the same or different return types. Therefore, Option C is the correct answer.

36. C. The declaration of `monday` does not compile, because the value of a `static final` variable must be set when it is declared or in a `static` initialization block. The declaration of `tuesday` is fine and compiles without issue. The declaration of `wednesday` does not compile because there is no data type for the variable. Finally, the declaration of `thursday` does not compile because the `final` modifier cannot appear before the access modifier. For these reasons, Option C is the correct answer.

37. D. The `Puppy` class does not declare a constructor, so the default no-argument constructor is automatically inserted by the compiler. What looks like a constructor in the class is actually a method that has a return type of `void`. Therefore, the line in the `main()` method to create the `new Puppy(2)` object does not compile, since there is no constructor capable of taking an `int` value, making Option D the correct answer.

38. A. The `public` modifier allows access to members in the same class, package, subclass, or even classes in other packages, while the `private` modifier allows access only to members in the same class. Therefore, the `public` access modifier allows access to everything the `private` access modifier does, and more, making Option A the correct answer. Options B, C, and D are incorrect because the first term is a more restrictive access modifier than the second term.

39. A. The code compiles without issue, so Option D is incorrect. The key here is that Java uses pass by value to send object references to methods. Since the `Phone` reference `p` was reassigned in the first line of the `sendHome()` method, any changes to the `p` reference were made to a new object. In other words, no changes in the `sendHome()` method affected the object that was passed in. Therefore, the value of `size` was the same before and after the method call, making the output `3` and Option A the correct answer.

40. B. Options A and D are equivalent and would allow the code to compile. They both are proper ways to access a `static` method from within an instance method. Option B is the correct answer. The class would not compile because `this.Drink` has no meaning to the compiler. Finally, Option C would still allow the code to compile, even though it is considered a poor coding practice. While `static` members should be accessed in a `static` way, it is not required.

41. C. The method signature requires one `int` value, followed by exactly one `String`, followed by `String` varargs, which can be an array of `String` values or zero or more individual `String` values. Only Option C conforms to these requirements, making it the correct answer.

42. D. Option A is a statement about `final static` variables, not all `static` variables. Option B only applies to `static` variables marked `private`, not `final`. Option C is false because `static` imports can be used to reference both variables and methods. Option D is the correct answer because a `static` variable is accessible to all instances of the class.

43. A. Option A is the correct answer because the first line of a constructor could be `this()` or `super()`, making it an untrue statement. Option B is a true statement because the compiler will insert the default no-argument constructor if one is not defined. Option C is also a true statement, since zero or more arguments may be passed to the parent constructor, if the parent class defines such constructors. Option D is also true. The value of a `final` instance variable should be set when it is declared, in an initialization block, or in a constructor.

44. D. The last `static` initialization block accesses `height`, which is an instance variable, not a `static` variable. Therefore, the code will not compile no matter how many `final` modifiers are removed, making Option D the correct answer. Note that if the line `height = 4;` was removed, then no `final` modifiers would need to be removed to make the class compile.

45. D. Since a constructor call is not the first line of the `RainForest()` constructor, the compiler inserts the no-argument `super()` call. Since the parent class, `Forest`, does not define a no-argument `super()` constructor, the `RainForest()` constructor does not compile, and Option D is correct.

46. A. The code compiles without issue, so Option D is incorrect. In the `main()` method, the value `2` is first cast to a `byte`. It is then increased by one using the addition `+` operator. The addition `+` operator automatically promotes all `byte` and `short` values to `int`. Therefore, the value passed to the `choose()` in the `main()` method is an `int`. The `choose(int)` method is called, returning `5` and making Option A the correct answer. Note that without the addition operation in the `main()` method, `byte` would have been used as the parameter to the `choose()` method, causing the `choose(short)` to be selected as the next closest type and outputting `2`, making Option B the correct answer.

47. C. The variable `startTime` can be automatically converted to `Integer` by the compiler, but `Integer` is not a subclass of `Long`. Therefore, the code does not compile due the wrong variable type being passed to the `getScore()` method on line `m2`, and Option C is correct.

48. A. Java methods must start with a letter, the dollar `$` symbol, or underscore `_` character. For these reasons, Options B and D are incorrect, and Option A is correct. Option C is incorrect. The hashtag (`#`) symbol cannot be included in a method name.

49. B. The `protected` modifier allows access by any subclass or class that is in the same package, therefore Option B is the correct answer.

50. D. A `static` import is used to import `static` members of another class. In this case, the `withdrawal()` and `deposit()` methods in the `Bank` class are not marked `static`. They require an instance of `Bank` to be used and cannot be imported as `static` methods. Therefore, Option D is correct. If the two methods in the `Bank` class were marked `static`, then Option A would be the correct answer since wildcards can be used with `static` imports to import more than one method. Option B reverses the keywords `static` and `import`, while Option C incorrectly imports a class, which cannot be imported via a `static` import.