C. void is a return type. Only the access modifier or optional specifiers are allowed before the return type. Option C is correct, creating a method with private access. Option B is correct, creating a method with default access and the optional specifier final. Since default access does not require a modifier, we get to jump right to final.

http://freepdf-books.com

Chapter 4: Methods and Encapsulation 343

bapp01.indd 1½ 4/2014 Page 343

Option A is incorrect because default access omits the access modifier rather than specifying default. Option D is incorrect because Java is case sensitive. It would have been correct if public were the choice. Option E is incorrect because the method already has a void return type. Option F is incorrect because labels are not allowed for methods. 2.

A, D. Options A and D are correct because the optional specifiers are allowed in any order. Options B and C are incorrect because they each have two return types. Options E and F are incorrect because the return type is before the optional specifier and access modifier, respectively.

3. A, C, D. Options A and C are correct because a void method is allowed to have a return statement as long as it doesn’t try to return a value. Options B and G do not compile because null requires a reference object as the return type. void is not a reference object since it is a marker for no return type. int is not a reference object since it is a primitive. Option D is correct because it returns an int value. Option E does not compile because it tries to return a double when the return type is int. Since a double cannot be assigned to an int, it cannot be returned as one either. Option F does not compile because no value is actually returned.

4. A, B, G. Options A and B are correct because the single vararg parameter is the last parameter declared. Option G is correct because it doesn’t use any vararg parameters at all. Options C and F are incorrect because the vararg parameter is not last. Option D is incorrect because two vararg parameters are not allowed in the same method. Option E is incorrect because the ... for a vararg must be after the type, not before it.

5. D, G. Option D passes the initial parameter plus two more to turn into a vararg array of size 2. Option G passes the initial parameter plus an array of size 2. Option A does not compile because it does not pass the initial parameter. Options E and F do not compile because they do not declare an array properly. It should be new boolean[] {true}. Option B creates a vararg array of size 0 and option C creates a vararg array of size 1.

6. D. Option D is correct. This is the common implementation for encapsulation by setting all fields to be private and all methods to be public. Option A is incorrect because protected access allows everything that package private access allows and additionally allows subclasses access. Option B is incorrect because the class is public. This means that other classes can see the class. However, they cannot call any of the methods or read any of the fields. It is essentially a useless class. Option C is incorrect because package private access applies to the whole package. Option E is incorrect because Java has no such capability.

7. B, C, D, F. The two classes are in different packages, which means private access and default (package private) access will not compile. Additionally, protected access will not compile since School does not inherit from Classroom. Therefore, only line 8 will compile because it uses public access.

8. B, C, E. Encapsulation requires using methods to get and set instance variables so other classes are not directly using them. Instance variables must be private for this to work. Immutability takes this a step further, allowing only getters, so the instance variables do not change state.

http://freepdf-books.com

344 Appendix A ■ Answers to Review Questions

bapp01.indd 1½ 4/2014 Page 344

9. C, E. Option A is incorrect because the property is of type boolean and getters must begin with is for booleans. Options B and D are incorrect because they don’t follow the naming convention of beginning with get/is/set. Options C and E follow normal getter and setter conventions.

10. B. Rope runs line 3, setting LENGTH to 5, then immediately after runs the static initializer, which sets it to 10. Line 5 calls the static method normally and prints swing. Line 6 also calls the static method. Java allows calling a static method through an instance variable. Line 7 uses the static import on line 2 to reference LENGTH.

11. B, E. Line 10 does not compile because static methods are not allowed to call instance methods. Even though we are calling play() as if it were an instance method and an instance exists, Java knows play() is really a static method and treats it as such. If line 10 is removed, the code works. It does not throw a NullPointerException on line 16 because play() is a static method. Java looks at the type of the reference for rope2 and translates the call to Rope.play().

12. D. There are two details to notice in this code. First, note that RopeSwing has an instance initializer and not a static initializer. Since RopeSwing is never constructed, the instance initializer does not run. The other detail is that length is static. Changes from one object update this common static variable.

13. E. static final variables must be set exactly once, and it must be in the declaration line or in a static initialization block. Line 4 doesn’t compile because bench is not set in either of these locations. Line 15 doesn’t compile because final variables are not allowed to be set after that point. Line 11 doesn’t compile because name is set twice: once in the declaration and again in the static block. Line 12 doesn’t compile because rightRope is set twice as well. Both are in static initialization blocks.

14. B. The two valid ways to do this are import static java.util.Collections.\*; and import static java.util.Collections.sort;. Option A is incorrect because you can only do a static import on static members. Classes such as Collections require a regular import. Option C is nonsense as method parameters have no business in an import. Options D, E, and F try to trick you into reversing the syntax of import static.

15. E. The argument on line 17 is a short. It can be promoted to an int, so print() on line 5 is invoked. The argument on line 18 is a boolean. It can be autoboxed to a boolean, so print() on line 11 is invoked. The argument on line 19 is a double. It can be autoboxed to a double, so print() on line 11 is invoked. Therefore, the output is intObjectObject and the correct answer is option E.

16. B. Since Java is pass-by-value and the variable on line 8 never gets reassigned, it stays as 9. In the method square, x starts as 9. y becomes 81 and then x gets set to –1. Line 9 does set result to 81. However, we are printing out value and that is still 9.

17. B, D, E. Since Java is pass-by-reference, assigning a new object to a does not change the caller. Calling append() does affect the caller because both the method parameter and

http://freepdf-books.com

Chapter 4: Methods and Encapsulation 345

bapp01.indd 1½ 4/2014 Page 345

caller have a reference to the same object. Finally, returning a value does pass the reference to the caller for assignment to s3 .

18. C, G. Since the main() method is in the same class, it can call private methods in the class. this() may only be called as the first line of a constructor. this.variableName can be called from any instance method to refer to an instance variable. It cannot be called from a static method because there is no instance of the class to refer to. Option F is tricky. The default constructor is only written by the compiler if no user-defined constructors were provided. this() can only be called from a constructor in the same class. Since there can be no user-defined constructors in the class if a default constructor was created, it is impossible for option F to be true.

19. A, G. Options B and C don’t compile because the constructor name must match the classname. Since Java is case sensitive, these don’t match. Options D, E, and F all compile and provide one user-defined constructor. Since a constructor is coded, a default constructor isn’t supplied. Option G defines a method, but not a constructor. Option A does not define a constructor, either. Since no constructor is coded, a default constructor is provided for options A and G.

20. E. Options A and B will not compile because constructors cannot be called without new. Options C and D will compile but will create a new object rather than setting the fields in this one. Option F will not compile because this() must be the first line of a constructor. Option E is correct.

21. C. Within the constructor numSpots refers to the constructor parameter. The instance variable is hidden because they have the same name. this.numSpots tells Java to use the instance variable. In the main() method, numSpots refers to the instance variable. Option A sets the constructor parameter to itself, leaving the instance variable as 0. Option B sets the constructor parameter to the value of the instance variable, making them both 0. Option C is correct, setting the instance variable to the value of the constructor parameter. Options D and E do not compile.

22. E. On line 3 of OrderDriver, we refer to Order for the first time. At this point the statics in Order get initialized. In this case, the statics are the static declaration of result

and the static initializer. result is u at this point. On line 4, result is the same because the static initialization is only run once. On line 5, we create a new Order, which triggers the instance initializers in the order they appear in the file. Now result

is ucr. Line 6 creates another Order, triggering another set of initializers. Now result

is ucrcr. Notice how the static is on a different line than the initialization code in lines 4–5 of Order. The exam may try to trick you by formatting the code like this to confuse you.

23. A. Line 4 instantiates an Order. Java runs the declarations and instance initializers first in the order they appear. This sets value to tacf. Line 5 creates another Order and initializes value to tacb. The object on line 5 is stored in the same variable line 4 used. This makes the object created on line 4 unreachable. When value is printed, it is the instance variable in the object created on line 5.

http://freepdf-books.com

346 Appendix A ■ Answers to Review Questions

bapp01.indd 1½ 4/2014 Page 346

24. B, C, E. value1 is a final instance variable. It can only be set once: in the variable declaration, an instance initializer, or a constructor. Option A does not compile because the final variable was already set in the declaration. value2 is a static variable. Both instance and static initializers are able to access static variables, making options B and E correct. value3 is an instance variable. Options D and F do not compile because a static initializer does not have access to instance variables.

25. A, E. The 100 parameter is an int and so calls the matching int constructor. When this constructor is removed, Java looks for the next most specific constructor. Java prefers autoboxing to varargs, and so chooses the Integer constructor. The 100L parameter is a long. Since it can’t be converted into a smaller type, it is autoboxed into a Long and then the constructor for Object is called.

26. A. This code is correct. Line 8 creates a lambda expression that checks if the age is less than 5. Since there is only one parameter and it does not specify a type, the parentheses around the type parameter are optional. Line 10 uses the Predicate interface, which declares a test() method. 27. C. The interface takes two int parameters. The code on line 7 attempts to use them as if one is a StringBuilder. It is tricky to use types in a lambda when they are implicitly specified. Remember to check the interface for the real type. 28. A, D, F. removeIf() expects a Predicate, which takes a parameter list of one parameter using the specified type. Options B and C are incorrect because they do not use the return keyword. It is required inside braces for lambda bodies. Option E is incorrect because it is missing the parentheses around the parameter list. This is only optional for a single parameter with an inferred type. 29. A, F. Option B is incorrect because it does not use the return keyword. Options C, D, and E are incorrect because the variable e is already in use from the lambda and cannot be redefined. Additionally, option C is missing the return keyword and option E is missing the semicolon.

Chapter 5: Class Design