Chapter 2: Working with Java Data Types

1. A. Option A does not compile because Java does not allow declaring different types as part of the same declaration. The other three options show various legal combinations of combining multiple variables in the same declarations with optional default values.
2. D. The `table` variable is initialized to "`metal`". However, `chair` is not initialized. In Java, initialization is per variable and not for all the variables in a single declaration. Therefore, the second line tries to reference an uninitialized local variable and does not compile, which makes Option D correct.
3. B. Instance variables have a default value based on the type. For any non-primitive, including `String`, that type is a reference to `null`. Therefore Option B is correct. If the variable was a local variable, Option C would be correct.
4. B. An identifier name must begin with a letter, `$`, or `_`. Numbers are only permitted for subsequent characters. Therefore, Option B is not a valid variable name.
5. B. In Java, class names begin with an uppercase letter by convention. Then they use lowercase with the exception of new words. Option B follows this convention and is correct. Option A follows the convention for variable names. Option C follows the convention for constants. Option D doesn't follow any Java conventions.
6. C. Objects have instance methods while primitives do not. Since `int` is a primitive, you cannot call instance methods on it. `Integer` and `String` are both objects and have instance methods. Therefore, Option C is correct.
7. C. Underscores are allowed between any two digits in a numeric literal. Underscores are not allowed at the beginning or end of the literal, making Option C the correct answer.
8. C. Option A is incorrect because `int` is a primitive. Option B is incorrect because it is not the name of a class in Java. While Option D is a class in Java, it is not a wrapper class because it does not map to a primitive. Therefore, Option C is correct.
9. C. There is no class named integer. There is a primitive `int` and a class `Integer`. Therefore, the code does not compile, and Option C is correct. If the type was changed to `Integer`, Option B would be correct.
.0. C. The `new` keyword is used to call the constructor for a class and instantiate an instance of the class. A primitive cannot be created using the `new` keyword. Dealing with references happens after the object created by `new` is returned.
ι1. D. Java uses the suffix `f` to indicate a number is a `float`. Java automatically widens a type, allowing a float to be assigned to either a `float` or a `double`. This makes both lines `p1` and `p3` compile. Line `p2` does compile without a suffix. Line `p4` does not compile without a suffix and therefore is the answer.
.2. A. A `byte` is smaller than a `char`, making Option C incorrect. `bigint` is not a primitive,

making Option D incorrect. A `double` uses twice as much memory as a `float` variable, therefore Option A is correct.

.3. D. The instance variables, constructor, and method names can appear in any order within a class declaration.

.4. B. Java does not allow multiple Java data types to be declared in the same declaration, making Option B the correct answer. If `double` was removed, both `hot` and `cold` would be the same type. Then the compiler error would be on `x3` because of a reference to an uninitialized variable.

.5. C. Lines 2 and 7 illustrate instance initializers. Line 6 is a static initializer. Lines 3–5 are a constructor.

.6. A. Since `defaultValue` is a local variable, it is not automatically initialized. That means the code will not compile with any type. Therefore, Option A is correct. If this was an instance variable, Option C would be correct as `int` and `short` would be initialized to `0` while `double` would be initialized to `0.0`.

l7. A. The `finalize()` method may not be called, such as if your program crashes. However, it is guaranteed to be called no more than once.

.8. D. `String` is a class, but it is not a wrapper class. In order to be a wrapper class, the class must have a one-to-one mapping with a primitive.

.9. C. Lines 15–17 create the three objects. Lines 18–19 change the references so `link2` and `link3` point to each other. The lines 20–21 wipe out two of the original references. This means the object with `name` as x is inaccessible.

:0. C. Options A and D are incorrect because `byte` and `short` do not store values with decimal points. Option B is tempting. However, `3.14` is automatically a `double`. It requires casting to `float` or writing `3.14f` in order to be assigned to a `float`. Therefore, Option C is correct.

21. B. `Integer` is the name of a class in Java. While it is bad practice to use the name of a class as your local variable name, this is legal. Therefore, `k1` does compile. It is not legal to use a reserved word as a variable name. All of the primitives including `int` are reserved words. Therefore, `k2` does not compile, and Option B is the answer. Line `k4` doesn't compile either, but the question asks about the first line to not compile.

:2. B. Dot notation is used for both reading and writing instance variables, assuming they are in scope. It cannot be used for referencing local variables, making Option B the correct answer.

:3. C. Class names follow the same requirements as other identifiers. Underscores and dollar signs are allowed. Numbers are allowed, but not as the first character of an identifier. Therefore, Option C is correct. Note that class names begin with an uppercase letter by convention, but this is not a requirement.

:4. D. This question is tricky as it appears to be about primitive vs. wrapper classes.

Looking closely, there is an underscore right before the decimal point. This is illegal as the underscore in a numeric literal can only appear between two digits.

25. C. Local variables do not have a default initialization value. If they are referenced before being set to a value, the code does not compile. Therefore, Option C is correct. If the variable was an instance variable, Option B would be correct. Option D is tricky. A local variable will compile without an initialization if it isn't referenced anywhere or it is assigned a value before it is referenced.

26. C. Since `defaultValue` is an instance variable, it is automatically initialized to the corresponding value for that type. For `double`, that value is `0.0`. By contrast, it is `0` for `int`, `long`, and `short`. Therefore Option C is correct.

27. B. Option B is an example of autoboxing. Java will automatically convert from primitive to wrapper class types and vice versa. Option A is incorrect because you can only call methods on an object. Option C is incorrect because this method is used for converting to a wrapper class from a `String`. Option D is incorrect because autoboxing will convert the primitive to an object before adding it to the `ArrayList`.

28. C. Java does not allow calling a method on a primitive. While autoboxing does allow the assignment of an `Integer` to an `int`, it does not allow calling an instance method on a primitive. Therefore, the last line does not compile.

29. D. In order to call a constructor, you must use the `new` keyword. It cannot be called as if it was a normal method. This rules out Options A and B. Further, Option C is incorrect because the parentheses are required.

30. A. Option A (I) correctly assigns the value to both variables. II does not compile as `dog` does not have a type. Notice the semicolon in that line, which starts a new statement. III compiles but only assigns the value to `dog` since a declaration only assigns to one variable rather than everything in the declaration. IV does not compile because the type should only be specified once per declaration.

31. C. The wrapper class for `int` is `Integer` and the wrapper class for `char` is `Character`. All other primitives have the same name. For example, the wrapper class for `boolean` is `Boolean`.

32. A. Assuming the variables are not primitives, they allow a `null` assignment. The other statements are false.

33. A. An example of a primitive type is `int`. All the primitive types are lowercase, making Option A correct. Unlike object reference variables, primitives cannot reference `null`. `String` is not a primitive as evidenced by the uppercase letter in the name and the fact that we can call methods on it. You can create your own classes, but not primitives.

34. D. While you can suggest to the JVM that it might want to run a garbage collection cycle, the JVM is free to ignore your suggestion. Option B is how to make this suggestion. Since garbage collection is not guaranteed to run, Option D is correct.

35. C. All three references point to the `String` apple. This makes the other two `String` objects eligible for garbage collection and Option C correct.

36. B. A constructor can only be called with a class name rather than a primitive, making Options A and C incorrect. The newly constructed `Double` object can be assigned to either a `double` or `Double` thanks to autoboxing. Therefore, Option B is correct.

37. B. First line 2 runs and sets the variable using the declaration. Then the instance initializer on line 6 runs. Finally, the constructor runs. Since the constructor is the last to run of the three, that is the value that is set when we print the result, so Option B is correct.

38. C. Objects are allowed to have a `null` reference while primitives cannot. `int` is a primitive, so assigning `null` to it does not compile. `Integer` and `String` are both objects and can therefore be assigned a `null` reference. Therefore, Option C is correct.

39. C. An instance variable can only be referenced from instance methods in the class. A `static` variable can be referenced from any method. Therefore, Option C is correct.

40. B. Underscores are allowed between any two digits in a numeric literal. Underscores are not allowed adjacent to a decimal point, making Option B the correct answer.

41. A. These four types represent nondecimal values. While you don't need to know the exact sizes, you do need to be able to order them from largest to smallest. A `byte` is smallest. A `short` comes next, followed by `int` and then `long`. Therefore, Option A is correct.

42. A. Java uses dot notation to reference instance variables in a class, making Option A correct.

43. B. If there was a `finalize()` method, this would be a different story. However, the method here is `finalizer`. Tricky! That's just a normal method that doesn't get called automatically. Therefore `clean` is never output.

44. A. Options B and C do not compile. In Java, braces are for arrays rather than instance variables. Option A is the correct answer. It uses dot notation to access the instance variable. It also shows that a private variable is accessible in the same class and that a narrower type is allowed to be assigned to a wider type.

45. B. The `parseInt()` methods return a primitive. The `valueOf()` methods return a wrapper class object. In real code, autoboxing would let you assign the return value to either a primitive or wrapper class. In terms of what gets returned directly, Option B is correct.

46. B. On line 9, all three objects have references. The `elena` and `zoe` objects have a direct reference. The `diana` object is referenced through the `elena` object. On line 10, the reference to the `diana` object is replaced by a reference to the `zoe` object. Therefore, the `diana` object is eligible to be garbage collected, and Option B is correct.

47. C. Options A and B are `static` methods rather than constructors. Option D is a method

that happens to have the same name as the class. It is not a constructor because constructors don’t have return types.

.8. A. Remember that garbage collection is not guaranteed to run on demand. If it doesn’t run at all, Option B would be output. If it runs at the requested point, Option C would be output. If it runs right at the end of the `main()` method, Option D would be output. Option A is the correct answer because `play` is definitely called twice. Note that you are unlikely to see all these scenarios if you run this code because we have not used enough memory for garbage collection to be worth running. However, you still need to be able to answer what could happen regardless of it being unlikely.

ı9. B. Each wrapper class has a constructor that takes the primitive equivalent. The methods mentioned in Options A, C, and D do not exist.

;0. C. The `main()` method calls the constructor which outputs `a`. Then the main method calls the `run()` method. The `run()` method calls the constructor again, which outputs `a` again. Then the `run()` method calls the `Sand()` method, which happens to have the same name as the constructor. This outputs `b`. Therefore, Option C is correct.