# Types and Assignments – Answers

Question 1

e Compile-time error at line 3.

Although the referenced object is indeed an array of type int, an explicit cast is necessary to cast the obj reference to an int array.

Question 2

**a d e h**

class A extends Object. The compiler will insert a default constructor implicitly. The default constructor has no throws clause. The default constructor invokes the no-parameter constructor of the superclass. Field i1 and method m1 both have package access. When no constructor is declared explicitly the compiler will insert one implicitly. The implicitly declared default constructor will have the same access privileges as the class. In this case, the class is public, so the default constructor is also public. The default constructor accepts no parameters and throws no exceptions.

Question 3

**d transient**

A *transient* field is not part of the persistent state of an object, so it is not serialized. A *static* field is also not part of the persistent state of an object, and also is not serialized.

Question 4

a d

A value cannot be assigned to a final field more than once. A compile-time error is thrown if a blank final instance variable is not assigned a value before the end of each constructor. A field cannot be declared both final and volatile. Static and non-static field variables may be declared final. All final fields must be definitely assigned a value once and only once. If the declaration of a final variable does not include an initializer then the variable is called a blank final. All blank, final, static variables must be assigned in a static initializer. All blank final non-static variables must be assigned by the end of the instance construction process.

Question 5

**b Prints: 0,0,0.0,null**

Each array contains the default value for its type. The default value of a primitive byte or a primitive long is printed as 0. The default value of a float primitive is printed as 0.0. The default value of an Object is null and is printed as null.

Question 6

**d char d = -1;**

The assignment of -1 to char d generates a compile-time error, because the primitive char type is unsigned. A negative int cannot be assigned to a char without an explicit cast. If the literal value -1 were cast to type char then the result would be \uffff.

Question 7

a boolean b1 = true;

There are two primitive boolean values: true and false. Both must be written with lower case letters. Although the C programming language accepts zero as a boolean value, the Java programming language does not.

Question 8

d 4

The literal 1.0 is a double and cannot be used to initialize a float without an explicit cast.

Question 9

**b c d e final private protected public**

A field is a class member. A static field is sometimes called a class variable. A non-static field is sometimes called an instance variable. A variable declaration that is immediately contained by a block such as a method body is called a local variable. The access modifiers, private, protected and public, can be applied to a field. A final field cannot have its value assigned more than once. The abstract modifier may be applied to methods but not to fields.

Question 10

**a 1**

The escape sequences are as follows: '\b' (backspace), '\f' (formfeed), '\n' (newline), '\r' (carriage return), '\t' (horizontal tab), '\\' (backslash), '\"' (double quote), '\'' (single quote). Yes, you must memorize the escape sequences! Just remember "big farms need red tractors".

Question 11

**c 3**

The length member of the array type is an attribute. A compile-time error is generated as a result of the attempt to access length as though it were a method.

Question 12

**a b c d e doubleValue floatValue intValue longValue parseDouble**

Question 13

b Prints: ffF

Question 14

f Prints: BbB

The Boolean.valueOf method returns a Boolean instance. The Boolean.booleanValue method returns a primitive. The Boolean.TRUE field is a reference to an instance of type Boolean that wraps the primitive value true.

Question 15

g Prints: true,true,false

Integer.equals overrides Object.equals. The Integer.equals method compares the data values contained in the Integer instances. If the argument is of type Integer and if the value contained in the argument is the same as the value contained in the instance on which the method is invoked, then the result is true. The equality operator, ==, does not compare data values. Instead, the equality operator compares references. Distinct instances of any two objects cannot have the same reference value; so the expression new Integer(i1) == new Integer(i1) is false.

Question 16

b d e Long.parseLong("1L") Long.parseLong("0x10") Long.parseLong("1.0")

Long.parseLong is overloaded: one version accepts a String argument that represents an integral value; the other accepts both a String argument and an argument of type int. The int argument represents the radix (i.e. base) of the String argument. The Long.parseLong method is not able to determine the type of the String value by examining a suffix such as L. Any such suffix results in a run-time error. The Long.parseLong method is not able to determine the radix of the String value by examing a prefix such as 0 or 0x. The 0 prefix used to identify octal values is accepted, but the String is parsed as a decimal value. The prefix 0x generates a run-time error. The Long.parseLong method generates a run-time error if the String argument is not formatted as a decimal integer. A floating-point format results in a run-time error.

Question 17

e Prints: true,false,false

Integer.equals overrides Object.equals. The Integer.equals method compares the data values contained in the Integer instances. If the argument is of type Integer and if the value contained in the argument is the same as the value contained in the instance on which the method is invoked, then the result is true. If the argument is not of type Integer then the result is false.

Question 18

e f parseDouble valueOf

Question 19

c C

All three references, i1, i2 and i3, refer to object named C; so C is not eligible for garbage collection when method m2 begins to execute. The objects named A and B have references to each other, but no other objects refer to A and B. The objects A and B form an island of isolated objects and are eligible for garbage collection.

Question 20

c Prints: XXYY

The program will not print XXYY. Please note that the question asks which could NOT be a result of attempting to compile and run the program. The finalize method of each instance can only run once; so X or Y can never be printed more than once. The instances referenced by x1 and y1 become eligible for garbage collection when method m returns; so both could be finalized at that time, but there is no guarantee that they will be. Even though System.gc is invoked in the main method, there is no guarantee that the garbage collector will run at that time. If the garbage collector does run before the program terminates, then the name of each object could be printed at most one time. The order in which the names are printed depends on the order in which the objects are finalized. If the garbage collector does not run, then nothing will be printed.