LANGUAGE FUNDAMENTALS

1. Answer & Explanation

Answer: Option **B**

Explanation:

(1), (3), (4), (5) are the correct statements.

(2) is wrong because the default value for a String (and any other object reference) isnull, with no quotes.

(6) is wrong because the default value for boolean elements is false.

2. Answer & Explanation

Answer: Option **B**

Explanation:

All the words in option B are among the 49 Java keywords. Although goto reserved as a keyword in Java, goto is not used and has no function.

Option A is wrong because the keyword for the primitive int starts with a lowercase i.

Option C is wrong because "virtual" is a keyword in C++, but not Java.

Option D is wrong because "constant" is not a keyword. Constants in Java are markedstatic and final.

Option E is wrong because "include" is a keyword in C, but not in Java.

3. Answer & Explanation

Answer: Option **D**

Explanation:

The only legal array declaration and assignment statement is Option D

Option A is wrong because it initializes an int array with String literals.

Option B is wrong because it use something other than curly braces for the initialization.

Option C is wrong because it provides initial values for only one dimension, although the declared array is a two-dimensional array.

4. Answer & Explanation

Answer: Option **B**

Explanation:

The word "native" is a valid keyword, used to modify a method declaration.

Option A, D and E are not keywords. Option C is wrong because the keyword for subclassing in Java is extends, not 'subclasses'.

5. Answer & Explanation

Answer: Option **A**

Explanation:

interface is a valid keyword.

Option B is wrong because although "String" is a class type in Java, "string" is not a keyword.

Option C is wrong because "Float" is a class type. The keyword for the Java primitive isfloat.

Option D is wrong because "unsigned" is a keyword in C/C++ but not in Java.

6. Answer & Explanation

Answer: Option **A**

Explanation:

(1), (2), and (4) are legal array declarations. With an array declaration, you can place the brackets to the right or left of the identifier. Option A looks strange, but it's perfectly legal to split the brackets in a multidimensional array, and place them on both sides of the identifier. Although coding this way would only annoy your fellow programmers, for the exam, you need to know it's legal.

(3) and (5) are wrong because you can't declare an array with a size. The size is only needed when the array is actually instantiated (and the JVM needs to know how much space to allocate for the array, based on the type of array and the size).

7. Answer & Explanation

Answer: Option **A**

Explanation:

(1), (2) and (3) are correct. Interfaces can have constants, which are always implicitlypublic, static, and final. Interface constant declarations of public, static, andfinal are optional in any combination.

8. Answer & Explanation

Answer: Option **B**

Explanation:

Option B is the legal way to declare and initialize an array with five elements.

Option A is wrong because it shows an example of instantiating a class named Array, passing the integer value 5 to the object's constructor. If you don't see the brackets, you can be certain there is no actual array object! In other words, an Array object (instance of class Array) is not the same as an array object.

Option C is wrong because it shows a legal array declaration, but with no initialization.

Option D is wrong (and will not compile) because it declares an array with a size. Arrays must never be given a size when declared.

9. Answer & Explanation

Answer: Option **B**

Explanation:

(1), (3), and (6) are correct. char c1 = 064770; is an octal representation of the integer value 27128, which is legal because it fits into an unsigned 16-bit integer. char c3 = 0xbeef; is a

hexadecimal representation of the integer value 48879, which fits into an unsigned 16-bit integer. char c6 = '\uface'; is a Unicode representation of a character.

char c2 = 'face'; is wrong because you can't put more than one character in a char literal. The only other acceptable char literal that can go between single quotes is a Unicode value, and Unicode literals must always start with a '\u'.

char c4 = \u0022; is wrong because the single quotes are missing.

char c5 = '\iface'; is wrong because it appears to be a Unicode representation (notice the backslash), but starts with '\i' rather than '\u'.

10. Answer & Explanation

Answer: Option **A**

Explanation:

Option A is correct. A public access modifier is acceptable. The method prototypes in an interface are all abstract by virtue of their declaration, and should not be declaredabstract.

Option B is wrong. The final modifier means that this method cannot be constructed in a subclass. A final method cannot be abstract.

Option C is wrong. static is concerned with the class and not an instance.

Option D is wrong. protected is not permitted when declaring a method of an interface. See information below.

Member declarations in an interface disallow the use of some declaration modifiers; you cannot use transient, volatile, or synchronized in a member declaration in an interface. Also, you may not use the private and protected specifiers when declaring members of an interface.

11. Answer & Explanation

Answer: Option **C**

Explanation:

A boolean can only be assigned the literal true or false.

12. Answer & Explanation

Answer: Option **C**

Explanation:

(1) and (3) are integer literals (32 bits), and integers can be legally assigned to floats (also 32 bits). (6) is correct because (F) is appended to the literal, declaring it as a floatrather than a double (the default for floating point literals).

(2), (4),and (5) are all doubles.

13. Answer & Explanation

Answer: Option **A**

Explanation:

Option A sets the String reference to null.

Option B is wrong because null cannot be in single quotes.

Option C is wrong because there are multiple characters between the single quotes ('abc').

Option D is wrong because you can't cast a char (primitive) to a String (object).

14. Answer & Explanation

Answer: Option **D**

Explanation:

A char is really a 16-bit integer behind the scenes, so it supports 2^{16} (from 0 to 65535) values.

INNER CLASSES

1. Answer & Explanation

Answer: Option **C**

Explanation:

Option C is correct because the syntax of an anonymous inner class allows for only one named type after the new, and that type must be either a single interface (in which case the anonymous class implements that one interface) or a single class (in which case the anonymous class extends that one class).

Option A, B, D, and E are all incorrect because they don't follow the syntax rules described in the response for answer Option C.

2. Answer & Explanation

Answer: Option **B**

Explanation:

Option B is correct because anonymous inner classes are no different from any other class when it comes to polymorphism. That means you are always allowed to declare a reference variable of the superclass type and have that reference variable refer to an instance of a subclass type, which in this case is an anonymous subclass of Bar. Since Bar is a subclass of Boo, it all works.

Option A is incorrect because it passes an int to the Boo constructor, and there is no matching constructor in the Boo class.

Option C is incorrect because it violates the rules of polymorphismâ€"you cannot refer to a superclass type using a reference variable declared as the subclass type. The superclass is not guaranteed to have everything the subclass has.

Option D uses incorrect syntax.

3. Answer & Explanation

Answer: Option **B**

Explanation:

Option B is correct because a method-local inner class can be abstract, although it means a subclass of the inner class must be created if the abstract class is to be used (so an abstract method-local inner class is probably not useful).

Option A is incorrect because a method-local inner class does not have to be declaredfinal (although it is legal to do so).

C and D are incorrect because a method-local inner class cannot be made public(remember-you cannot mark any local variables as public), or static.

4. Answer & Explanation

Answer: Option **B**

Explanation:

Option B is correct because a static nested class is not tied to an instance of the enclosing class, and thus can't access the nonstatic members of the class (just as a static method can't access nonstatic members of a class).

Option A is incorrect because static nested classes do not need (and can't use) a reference to an instance of the enclosing class.

Option C is incorrect because static nested classes can declare and define nonstatic members.

Option D is wrong because it just is. There's no rule that says an inner or nested class has to extend anything.

5. Answer & Explanation

Answer: Option **D**

Explanation:

D is correct. It defines an anonymous inner class instance, which also means it creates an instance of that new anonymous class at the same time. The anonymous class is an implementer of the Runnable interface, so it must override the run() method ofRunnable.

A is incorrect because it doesn't override the run() method, so it violates the rules of interface implementation.

B and C use incorrect syntax.

6. Answer & Explanation

Answer: Option **B**

Explanation:

Option B is correct because the syntax is correct-using both names (the enclosing class and the inner class) in the reference declaration, then using a reference to the enclosing class to invoke new on the inner class.

Option A, C and D all use incorrect syntax. A is incorrect because it doesn't use a reference to the enclosing class, and also because it includes both names in the new.

C is incorrect because it doesn't use the enclosing class name in the reference variable declaration, and because the new syntax is wrong.

D is incorrect because it doesn't use the enclosing class name in the reference variable declaration.

7. Answer & Explanation

Answer: Option **A**

Explanation:

MyInner is a static nested class, so it must be instantiated using the fully-scoped name of MyOuter.MyInner.

Option B is incorrect because it doesn't use the enclosing name in the new.

Option C is incorrect because it uses incorrect syntax. When you instantiate a nested class by invoking new on an instance of the enclosing class, you do not use the enclosing name. The difference between Option A and C is that Option C is calling new on an instance of the enclosing class rather than just new by itself.

Option D is incorrect because it doesn't use the enclosing class name in the variable declaration.

DECLARATIONS AND ACCESS CONTROL

1. Answer & Explanation

Answer: Option **C**

Explanation:

Access modifiers dictate which classes, not which instances, may access features.

Methods and variables are collectively known as members. Method and variable members are given access control in exactly the same way.

private makes a member accessible only from within its own class

protected makes a member accessible only to classes in the same package or subclass of the class

default access is very similar to protected (make sure you spot the difference) default access makes a member accessible only to classes in the same package.

public means that all other classes regardless of the package that they belong to, can access the member (assuming the class itself is visible)

final makes it impossible to extend a class, when applied to a method it prevents a method from being overridden in a subclass, when applied to a variable it makes it impossible to reinitialise a variable once it has been initialised

abstract declares a method that has not been implemented.

transient indicates that a variable is not part of the persistent state of an object.

volatile indicates that a thread must reconcile its working copy of the field with the master copy every time it accesses the variable.

After examining the above it should be obvious that the access modifier that provides the most restrictions for methods to be accessed from the subclasses of the class from another package is C - protected. A is also a contender but C is more restrictive, B would be the answer if the constraint was the "same package" instead of "any package" in other words the subclasses clause in the question eliminates default.

2. Answer & Explanation

Answer: Option **A**

Explanation:

Option A compiles without problem.

Option B gives error - non-static variable cannot be referenced from a static context.

Option C package ot does not exist.

Option D gives error - non-static variable cannot be referenced from a static context.

3. Answer & Explanation

Answer: Option **C**

Explanation:

(3) is correct because an abstract class doesn't have to implement any or all of its interface's methods. (4) is correct because the method is correctly implemented ((7 > 4) is a boolean).

(1) is incorrect because interfaces don't implement anything. (2) is incorrect because classes don't extend interfaces. (5) is incorrect because interface methods are implicitlypublic, so the methods being implemented must be public.

4. Answer & Explanation

Answer: Option **C**

Explanation:

(1), (2) and (6) are valid array declarations.

Option (3) is not a correct array declaration. The compiler complains with: illegal start of type. The brackets are in the wrong place. The following would work: public int[ ] a

Option (4) is not a correct array declaration. The compiler complains with: ']' expected. A closing bracket is expected in place of the 3. The following works: private int a []

Option (5) is not a correct array declaration. The compiler complains with 2 errors:

']' expected. A closing bracket is expected in place of the 3 and

<identifier> expected A variable name is expected after a[ ] .

5. Answer & Explanation

Answer: Option **C**

Explanation:

Option A and B are wrong because they use the default access modifier and the access modifier for the class is public (remember, the default constructor has the same access modifier as the class).

Option D is wrong. The void makes the compiler think that this is a method specification - in fact if it were a method specification the compiler would spit it out.

6. Answer & Explanation

Answer: Option **E**

Explanation:

default access is the "package oriented" access modifier.

Option A and C are wrong because public and protected are less restrictive. Option B and D are wrong because abstract and synchronized are not access modifiers.

7. Answer & Explanation

Answer: Option **D**

8. Answer & Explanation

Answer: Option **B**

Explanation:

Option B generates a compiler error: <identifier> expected. The compiler thinks you are trying to create two arrays because there are two array initialisers to the right of the equals, whereas your intention was to create one 3 x 3 two-dimensional array.

To correct the problem and make option B compile you need to add an extra pair of curly brackets:

int [ ] [ ] scores = { {2,7,6}, {9,3,45} };

9. Answer & Explanation

Answer: Option **B**

Explanation:

(2), (3), and (5). These are all valid interface method signatures.

(1), is incorrect because an interface method must be public; if it is not explicitly declared public it will be made public implicitly. (4) is incorrect because interface methods cannot be static.

10. Answer & Explanation

Answer: Option **D**

Explanation:

The only two real contenders are C and D. Protected access Option C makes a member accessible only to classes in the same package or subclass of the class. While default access Option D makes a member accessible only to classes in the same package.

11. Answer & Explanation

Answer: Option **D**

Explanation:

However A, B and C are all wrong. Each of these would result in a narrowing conversion. Whereas we want a widening conversion, therefore the only correct answer is D. Don't be put off by the long cast, this applies only to the variable x and not the rest of the expression. It is the variable y (of type double) that forces the widening conversion todouble.

Java's widening conversions are:

- From a byte to a short, an int, a long, a float, or a double.

- From a short, an int, a long, a float, or a double.

- From a char to an int, a long, a float, or a double.

- From an int to a long, a float, or a double.

- From a long to a float, or a double.

- From a float to a double.

12. Answer & Explanation

Answer: Option **A**

Explanation:

Option A is correct - because the class that extends A is just simply overriding method1.

Option B is wrong - because it can't override as there are less access privileges in the subclass method1.

Option C is wrong - because to override it, the return type needs to be an integer. The different return type means that the method is not overriding but the same argument list means that the method is not overloading. Conflict - compile time error.

Option D is wrong - because you can't override a method and make it a class method i.e. using static.

13. Answer & Explanation

Answer: Option **A**

Explanation:

Option A is correct. It uses correct array declaration and correct array construction.

Option B is incorrect. It generates a compiler error: incompatible types because the array variable declaration is not correct. The array construction expects a reference type, but it is supplied with a primitive type in the declaration.

Option C is incorrect. It generates a compiler error: incompatible types because a string literal is not assignable to a character type variable.

Option D is wrong, it generates a compiler error <identifier> expected. The compiler thinks that you are trying to create two arrays because there are two array initialisers to the right of the equals, whereas your intention was to create a 3 x 3 two-dimensional array.

14. Answer & Explanation

Answer: Option **C**

Explanation:

(3), (6). Both are legal class declarations.

(1) is wrong because a class cannot be abstract and finalâ€”there would be no way to use such a class. (2) is wrong because interfaces and classes cannot be marked asstatic. (4) and (5) are wrong because classes and interfaces cannot be marked asprotected.

15. Answer & Explanation

Answer: Option **C**

Explanation:

Option C will not compile; the synchronized modifier applies only to methods.

Option A and B will compile because protected and transient are legal variable modifiers. Option D will compile because volatile is a proper variable modifier.

16. Answer & Explanation

Answer: Option **D**

Explanation:

(1) causes two compiler errors ('[' expected and illegal start of expression) because the wrong type of bracket is used, ( ) instead of [ ]. The following is the correct syntax:float[ ] f = new float[3];

(2) causes a compiler error ('{' expected) because the array constructor does not specify the number of elements in the array. The following is the correct syntax: float f2[ ] = new float[3];

(3), (4), and (5) compile without error.

17. Answer & Explanation

Answer: Option **C**

Explanation:

The private access modifier limits access to members of the same class.

Option A, B, D, and E are wrong because protected are the wrong access modifiers, and final, static, and volatile are modifiers but not access modifiers.

18. Answer & Explanation

Answer: Option **A**

Explanation:

(A) is valid interface declarations.

(B) and (C) are incorrect because interface variables cannot be either protected ortransient. (D) is incorrect because interface methods cannot be final or static.

www.v

EXCEPTIONS

1. Answer & Explanation

Answer: Option **A**

Explanation:

If you put a finally block after a try and its associated catch blocks, then once execution enters the try block, the code in that finally block will definitely be executed except in the following circumstances:

1. An exception arising in the finally block itself.
2. The death of the thread.
3. The use of System.exit()
4. Turning off the power to the CPU.

I suppose the last three could be classified as VM shutdown.

2. Answer & Explanation

Answer: Option **C**

Explanation:

Compilation fails because ArithmeticException has already been caught.ArithmeticException is a subclass of java.lang.Exception, by time theArithmeticException has been specified it has already been caught by theException class.

If ArithmeticException appears before Exception, then the file will compile. When catching exceptions the more specific exceptions must be listed before the more general (the subclasses must be caught before the superclasses).

3. Answer & Explanation

Answer: Option **C**

Explanation:

Error is thrown but not recognised line(22) because the only catch attempts to catch an Exception and Exception is not a superclass of Error. Therefore only the code in

the finally statement can be run before exiting with a runtime error (Exception in thread "main" java.lang.Error).

4. Answer & Explanation

Answer: Option **C**

Explanation:

A Run time exception is thrown and caught in the catch statement on line 10. All the code after the finally statement is run because the exception has been caught.

5. Answer & Explanation

Answer: Option **D**

Explanation:

The main() method properly catches and handles the RuntimeException in the catch block, finally runs (as it always does), and then the code returns to normal.

A, B and C are incorrect based on the program logic described above. Remember that properly handled exceptions do not cause the program to stop executing.

6. Answer & Explanation

Answer: Option **C**

Explanation:

This is what happens:

(1) The execution of the try block (line 5) completes abruptly because of the throwstatement (line 7).

(2) The exception cannot be assigned to the parameter of any catch clause of the trystatement therefore the finally block is executed (line 9) and "finally" is output (line 11).

(3) The finally block completes normally, and then the try statement completes abruptly because of the throw statement (line 7).

(4) The exception is propagated up the call stack and is caught by the catch in the main method (line 20). This prints "exception".

(5) Lastly program execution continues, because the exception has been caught, and "finished" is output (line 24).

7. Answer & Explanation

Answer: Option **C**

Explanation:

There is no exception thrown, so all the code with the exception of the catch statement block is run.

8. Answer & Explanation

Answer: Option **D**

Explanation:

(1) A RuntimeException is thrown, this is a subclass of exception.

(2) The exception causes the try to complete abruptly (line 7) therefore line 8 is never executed.

(3) The exception is caught (line 10) and "B" is output (line 12)

(4) The finally block (line 14) is always executed and "C" is output (line 16).

(5) The exception was caught, so the program continues with line 18 and outputs "D".

9. Answer & Explanation

Answer: Option **D**

Explanation:

Finally clauses are always executed. The program will first execute the try block, printing Hello world, and will then execute the finally block, printing Finally executing.

Option A, B, and C are incorrect based on the program logic described above. Remember that either a catch or a finally statement must follow a try. Since the finally is present, the catch is not required.

10. Answer & Explanation

Answer: Option **A**

Explanation:

An exception Exc1 is thrown and is caught by the catch statement on line 11. The code is executed in this block. There is no finally block of code to execute.

www.vtucs.co

THREADS

1. Answer & Explanation

Answer: Option **B**

Explanation:

Option B is Correct. The start() method causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.

Option A is wrong. There is no init() method in the Thread class.

Option C is wrong. The run() method of a thread is like the main() method to an application. Starting the thread causes the object's run method to be called in that separately executing thread.

Option D is wrong. The resume() method is deprecated. It resumes a suspended thread.

2. Answer & Explanation

Answer: Option **C**

Explanation:

(1) and (2) are both valid constructors for Thread.

(3), (4), and (5) are not legal Thread constructors, although (4) is close. If you reverse the arguments in (4), you'd have a valid constructor.

3. Answer & Explanation

Answer: Option **C**

Explanation:

(1), (2), and (6) are correct. They are all related to the list of threads waiting on the specified object.

(3), (5), (7), and (8) are incorrect answers. The methods isInterrupted() andinterrupt() are instance methods of Thread.

The methods sleep() and yield() are static methods of Thread.

D is incorrect because synchronized is a keyword and the synchronized()construct is part of the Java language.

4. Answer & Explanation

Answer: Option **C**

Explanation:

Option C is suitable to start a thread.

5. Answer & Explanation

Answer: Option **C**

Explanation:

Option C is correct. notify() - wakes up a single thread that is waiting on this object's monitor.

6. Answer & Explanation

Answer: Option **A**

Explanation:

(1) and (4). Only start() and run() are defined by the Thread class.

(2) and (3) are incorrect because they are methods of the Object class. (5) is incorrect because there's no such method in any thread-related class.

7. Answer & Explanation

Answer: Option **B**

Explanation:

(2) is correct because wait() always causes the current thread to go into the object's wait pool.

(5) is correct because sleep() will always pause the currently running thread for at least the duration specified in the sleep argument (unless an interrupted exception is thrown).

(6) is correct because, assuming that the thread you're calling join() on is alive, the thread calling join() will immediately block until the thread you're calling join() on is no longer alive.

(1) is wrong, but tempting. The yield() method is not guaranteed to cause a thread to leave the running state, although if there are runnable threads of the same priority as the currently running thread, then the current thread will probably leave the running state.

(3) and (4) are incorrect because they don't cause the thread invoking them to leave the running state.

(7) is wrong because there's no such method.

8. Answer & Explanation

Answer: Option **A**

Explanation:

Option A is correct. wait() causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

Option B is wrong. notify() - wakes up a single thread that is waiting on this object's monitor.

Option C is wrong. notifyAll() - wakes up all threads that are waiting on this object's monitor.

Option D is wrong. Typically, releasing a lock means the thread holding the lock (in other words, the thread currently in the synchronized method) exits the synchronized method. At that point, the lock is free until some other thread enters a synchronized method on that object. Does entering/exiting synchronized code mean that the thread execution stops? Not necessarily because the thread can still run code that is not synchronized. I think the word directly in the question gives us a clue. Exiting synchronized code does not directly stop the execution of a thread.

9. Answer & Explanation

Answer: Option **B**

Explanation:

Option B is correct because in an interface all methods are abstract by default therefore they must be overridden by the implementing class. The Runnable interface only contains 1 method, the void run() method therefore it must be implemented.

Option A and D are incorrect because they are narrowing the access privileges i.e. package(default) access is narrower than public access.

Option C is not method in the Runnable interface therefore it is incorrect.

10. Answer & Explanation

Answer: Option **A**

Explanation:

Option A is Correct. The run() method to a thread is like the main() method to an application. Starting the thread causes the object's run method to be called in that separately executing thread.

Option B is wrong. The start() method causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.

Option C is wrong. The stop() method is deprecated. It forces the thread to stop executing.

Option D is wrong. Is the main entry point for an application.

11. Answer & Explanation

Answer: Option **C**

Explanation:

Option C is correct. The start() method causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.

Option A is wrong. The run() method of a thread is like the main() method to an application. Starting the thread causes the object's run method to be called in that separately executing thread.

Option B is wrong. There is no construct() method in the Thread class.

Option D is wrong. There is no register() method in the Thread class.

12. Answer & Explanation

Answer: Option **A**

Explanation:

Option A. Either of the two events (notification or wait time expiration) will make the thread become a candidate for running again.

Option B is incorrect because a waiting thread will not return to runnable when the lock is released, unless a notification occurs.

Option C is incorrect because the thread will become a candidate immediately after notification, not two seconds afterwards.

Option D is also incorrect because a thread will not come out of a waiting pool just because a lock has been released.

13. Answer & Explanation

Answer: Option **A**

Explanation:

Option A is correct. notify() - wakes up a single thread that is waiting on this object's monitor.

Option B is wrong. wait() causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

Option C is wrong. Methods of the InputStream class block until input data is available, the end of the stream is detected, or an exception is thrown. Blocking means that a thread may stop until certain conditions are met.

Option D is wrong. sleep() - Causes the currently executing thread to sleep (temporarily cease execution) for a specified number of milliseconds. The thread does not lose ownership of any monitors.

14. Answer & Explanation

Answer: Option **A**

Explanation:

The Object class defines these thread-specific methods.

Option B, C, and D are incorrect because they do not define these methods. And yes, the Java API does define a class called Class, though you do not need to know it for the exam.

15. Answer & Explanation

Answer: Option **C**

Explanation:

Because the class implements Runnable, an instance of it has to be passed to theThread constructor, and then the instance of the Thread has to be started.

A is incorrect. There is no constructor like this for Runnable because Runnable is an interface, and it is illegal to pass a class or interface name to any constructor.

B is incorrect for the same reason; you can't pass a class or interface name to any constructor.

D is incorrect because MyRunnable doesn't have a start() method, and the onlystart() method that can start a thread of execution is the start() in the Threadclass.

www.vtu

OPERATORS AND ASSIGNMENTS

1. Answer & Explanation

Answer: Option **B**

Explanation:

Output: 15 15

The reference variables a1 and a3 refer to the same long array object. When the [1]element is updated in the fix() method, it is updating the array referred to by a1. The reference variable a2 refers to the same array object.

So Output: 3+7+5+" "3+7+5

Output: 15 15 Because Numeric values will be added

2. Answer & Explanation

Answer: Option **B**

Explanation:

The boolean b1 in the fix() method is a different boolean than the b1 in the start()method. The b1 in the start() method is not updated by the fix() method.

3. Answer & Explanation

Answer: Option **D**

Explanation:

When the fix() method is first entered, start()'s s1 and fix()'s s1 reference variables both refer to the same String object (with a value of "slip"). Fix()'s s1 is reassigned to a new object that is created when the concatenation occurs (this second String object has a value of "slipstream"). When the program returns to start(), another String object is created, referred to by s2 and with a value of "stream".

4. Answer & Explanation

Answer: Option **A**

Explanation:

Option A is correct. The >>> operator moves all bits to the right, zero filling the left bits. The bit transformation looks like this:

Before: 1000 0000 0000 0000 0000 0000 0000 0000

After: 0000 0000 0000 0000 0000 0000 0000 0001

Option C is incorrect because the >>> operator zero fills the left bits, which in this case changes the sign of x, as shown.

Option B is incorrect because the output method print() always displays integers in base 10.

Option D is incorrect because this is the reverse order of the two output numbers.

5. Answer & Explanation

Answer: Option **C**

Explanation:

The code will not compile because in line 7, the line will work only if we use (x==y) in the line. The == operator compares values to produce a boolean, whereas the = operator assigns a value to variables.

Option A, B, and D are incorrect because the code does not get as far as compiling. If we corrected this code, the output would be false.

6. Answer & Explanation

Answer: Option **B**

Explanation:

This is an example of a nested ternary operator. The second evaluation (x < 22) istrue, so the "tiny" value is assigned to sup.

7. Answer & Explanation

Answer: Option **C**

Explanation:

In the first two iterations x is incremented once and y is not because of the short circuit&& operator. In the third and forth iterations x and y are each incremented, and in the fifth iteration x is doubly incremented and y is incremented.

8. Answer & Explanation

Answer: Option **B**

Explanation:

The first two iterations of the for loop both x and y are incremented. On the third iteration x is incremented, and for the first time becomes greater than 2. The short circuit or operator || keeps y from ever being incremented again and x is incremented twice on each of the last three iterations.

9. Answer & Explanation

Answer: Option **D**

Explanation:

The & operator produces a 1 bit when both bits are 1. The result of the & operation is 9.
The ^ operator produces a 1 bit when exactly one bit is 1; the result of this operation is 10.
The | operator produces a 1 bit when at least one bit is 1; the result of this operation is 14.

10. Answer & Explanation

Answer: Option **B**

Explanation:

The & operator has a higher precedence than the | operator so that on line 8 b1 and b2are evaluated together as are b2 & b3. The final b1 in line 10 is what causes that if test to be true. Hence it prints "dokey".

11. Answer & Explanation

Answer: Option **D**

Explanation:

Because all of these expressions use the + operator, there is no precedence to worry about and all of the expressions will be evaluated from left to right. If either operand being evaluated is a String, the + operator will concatenate the two operands; if both operands are numeric, the + operator will add the two operands.

12. Answer & Explanation

Answer: Option **B**

Explanation:

The int x in the twice() method is not the same int x as in the start() method.Start()'s x is not affected by the twice() method. The instance variable s is updated by twice()'s x, which is 14.

13. Answer & Explanation

Answer: Option **C**

Explanation:

In the fix() method, the reference variable tt refers to the same object (class Two) as the t reference variable. Updating tt.x in the fix() method updates t.x (they are one in the same object). Remember also that the instance variable x in the Two class is initialized to 0.

14. Answer & Explanation

Answer: Option **C**

Explanation:

The reference variables b and x both refer to the same boolean array. count is incremented for each call to the set() method, and once again when the first if test istrue. Because of the && short circuit operator, count is not incremented during the second if test.

15. Answer & Explanation

Answer: Option **B**

Explanation:

Java only ever passes arguments to a method by value (i.e. a copy of the variable) and never by reference. Therefore the value of the variable i remains unchanged in themain method.

If you are clever you will spot that 16 is 4 multiplied by 2 twice, (4 * 2 * 2) = 16. If you had 16 left shifted by three bits then 16 * 2 * 2 * 2 = 128. If you had 128 right shifted by 2 bits then 128 / 2 / 2 = 32. Keeping these points in mind, you don't have to go converting to binary to do the left and right bit shifts.

www.vtucs.co

OBJECTS AND COLLECTIONS

1. Answer & Explanation

Answer: Option **C**

Explanation:

The iteration order of a Collection is the order in which an iterator moves through the elements of the Collection. The iteration order of a LinkedHashMap is determined by the order in which elements are inserted.

When a new LinkedHashMap is created by passing a reference to an existing Collection to the constructor of a LinkedHashMap the Collection.addAll method will ultimately be invoked.

The addAll method uses an iterator to the existing Collection to iterate through the elements of the existing Collection and add each to the instance of the newLinkedHashMap.

Since the iteration order of the LinkedHashMap is determined by the order of insertion, the iteration order of the new LinkedHashMap must be the same as the interation order of the old Collection.

2. Answer & Explanation

Answer: Option **C**

Explanation:

java.lang.StringBuffer is the only class in the list that uses the default methods provided by class Object.

3. Answer & Explanation

Answer: Option **D**

Explanation:

All of the collection classes allow you to grow or shrink the size of your collection.ArrayList provides an index to its elements. The newer collection classes tend not to have synchronized methods. Vector is an older implementation of ArrayListfunctionality and has synchronized methods; it is slower than ArrayList.

4. Answer & Explanation

Answer: Option **B**

Explanation:

Option B is correct. A set is a collection that contains no duplicate elements. The iterator returns the elements in no particular order (unless this set is an instance of some class that provides a guarantee). A map cannot contain duplicate keys but it may contain duplicate values. List and Collection allow duplicate elements.

Option A is wrong. A map is an object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value. The Map interface provides three collection views, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings. The order of a map is defined as the order in which the iterators on the map's collection views return their elements. Some map implementations, like the TreeMap class, make specific guarantees as to their order (ascending key order); others, like the HashMap class, do not (does not guarantee that the order will remain constant over time).

Option C is wrong. A list is an ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list. Unlike sets, lists typically allow duplicate elements.

Option D is wrong. A collection is also known as a sequence. The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list. Unlike sets, lists typically allow duplicate elements.

5. Answer & Explanation

Answer: Option **A**

Explanation:

Hash table based implementation of the Map interface.

6. Answer & Explanation

Answer: Option **A**

Explanation:

An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.

7. Answer & Explanation

Answer: Option **B**

Explanation:

LinkedHashMap is the collection class used for caching purposes. FIFO is another way to indicate caching behavior. To retrieve LinkedHashMap elements in cached order, use the values() method and iterate over the resultant collection.

8. Answer & Explanation

Answer: Option **D**

Explanation:

Hashtable is the only class listed that provides synchronized methods. If you need synchronization great; otherwise, use HashMap, it's faster.

9. Answer & Explanation

Answer: Option **A**

Explanation:

Option A is valid declaration of float.

Option B is incorrect because any literal number with a decimal point u declare the computer will implicitly cast to double unless you include "F or f"

Option C is incorrect because it is a String.

Option D is incorrect because "d" tells the computer it is a double so therefore you are trying to put a double value into a float variable i.e there might be a loss of precision.

10. Answer & Explanation

Answer: Option **A**

Explanation:

The usual method for using/importing the java packages/classes is by using an import statement at the top of your code. However it is possible to explicitly import the specific class that you want to use as you use it which is shown in the code above. The disadvantage of this however is that every time you create a new object you will have to use the class path in the case "java.io" then the class name in the long run leading to a lot more typing.

11. Answer & Explanation

Answer: Option **B**

Explanation:

The char type is integral but unsigned. The range of a variable of type char is from 0 to 2^{16}-1 or 0 to 65535. Java characters are Unicode, which is a 16-bit encoding capable of representing a wide range of international characters. If the most significant nine bits of a char are 0, then the encoding is the same as seven-bit ASCII.

12. Answer & Explanation

Answer: Option **B**

Explanation:

(2) - This is a Java keyword

(3) - This is a Java keyword

(1) - Is incorrect because although it is a method of Thread/Runnable it is not a keyword

(4) - This is not a Java keyword the keyword is implements

GARBAGE COLLECTIONS

1. Answer & Explanation

Answer: Option **D**

2. Answer & Explanation

Answer: Option **D**

Explanation:

Option D is correct. Garbage collection takes place after the method has returned its reference to the object. The method returns to line 6, there is no reference to store the return value. so garbage collection takes place after line 6.

Option A is wrong. Because the reference to obj1 is stored in obj2[0]. The Objectobj1 still exists on the heap and can be accessed by an active thread through the reference stored in obj2[0].

Option B is wrong. Because it is only one of the references to the object obj1, the other reference is maintained in obj2[0].

Option C is wrong. The garbage collector will not be called here because a reference to the object is being maintained and returned in obj2[0].

3. Answer & Explanation

Answer: Option **B**

Explanation:

Option B is correct. All references to the Bar object created on line 6 are destroyed when a new reference to a new Bar object is assigned to the variable newBar on line 14. Therefore the Bar object, created on line 6, is eligible for garbage collection after line 14.

Option A is wrong. This actually protects the object from garbage collection.

Option C is wrong. Because the reference in the doBar() method is returned on line 7 and is stored in newBar on line 12. This preserver the object created on line 6.

Option D is wrong. Not applicable because the object is eligible for garbage collection after line 14.

4. Answer & Explanation

Answer: Option **D**

Explanation:

Option D is correct. By a process of elimination.

Option A is wrong. The variable d is a member of the Test class and is never directly set to null.

Option B is wrong. A copy of the variable d is set to null and not the actual variable d.

Option C is wrong. The variable d exists outside the start() method (it is a class member). So, when the start() method finishes the variable d still holds a reference

5. Answer & Explanation

Answer: Option **B**

Explanation:

By the time line 8 has run, the only object without a reference is the one generated as a result of line 6. Remember that "Java is pass by value," so the reference variable x is not affected by the m1() method.

6. Answer & Explanation

Answer: Option **C**

Explanation:

Option A is wrong. This simply copies the object reference into the array.

Option B is wrong. The reference o is set to null, but, oa[0] still maintains the reference to the Float object.

Option C is correct. The thread of execution will then not have access to the object.

7. Answer & Explanation

Answer: Option **C**

Explanation:

This is an example of the islands of isolated objects. By the time line 11 has run, the objects instantiated in lines 6 and 7 are referring to each other, but no live thread can reach either of them.

8. Answer & Explanation

Answer: Option **D**

Explanation:

Option D is correct. When an object is no longer referenced, it may be reclaimed by the garbage collector. If an object declares a finalizer, the finalizer is executed before the object is reclaimed to give the object a last chance to clean up resources that would not otherwise be released. When a class is no longer needed, it may be unloaded.

Option A is wrong. I found 4 delete() methods in all of the Java class structure. They are:

1. delete() - Method in class java.io.File : Deletes the file or directory denoted by this abstract pathname.
2. delete(int, int) - Method in class java.lang.StringBuffer : Removes the characters in a substring of this StringBuffer.
3. delete(int, int) - Method in interfacejavax.accessibility.AccessibleEditableText : Deletes the text between two indices
4. delete(int, int) - Method in class :javax.swing.text.JTextComponent.AccessibleJTextComponent; Deletes the text between two indices

None of these destroy the object to which they belong.

Option B is wrong. I found 19 finalize() methods. The most interesting, from this questions point of view, was the finalize() method in class java.lang.Objectwhich is called by the garbage collector on an object when garbage collection determines that there are no more references to the object. This method does not destroy the object to which it belongs.

Option C is wrong. But it is interesting. The Runtime class has many methods, two of which are:

1. getRuntime() - Returns the runtime object associated with the current Java application.
2. gc() - Runs the garbage collector. Calling this method suggests that the Java virtual machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method

call, the virtual machine has made its best effort to recycle all discarded objects. Interesting as this is, it doesn't destroy the object.

www.vtucs.com