Q.What will be the output of the program? //EXACTLY SAME QUESTION

public class Test

{

public static void aMethod() throws Exception

{

try /\* Line 5 \*/

{

throw new Exception(); /\* Line 7 \*/

}

finally /\* Line 9 \*/

{

System.out.print("finally "); /\* Line 11 \*/

}

}

public static void main(String args[])

{

try

{

aMethod();

}

catch (Exception e) /\* Line 20 \*/

{

System.out.print("exception ");

}

System.out.print("finished"); /\* Line 24 \*/

}

}

A. finally

B. exception finished

C. finally exception finished

D. Compilation fails

Answer: Option C

Explanation:

This is what happens:

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

(2) The exception cannot be assigned to the parameter of any catch clause of the try statement 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).

Q. class MyThread extends Thread // EXACT YA SIMILAR

{

public static void main(String [] args)

{

MyThread t = new MyThread();

t.start();

System.out.print("one. ");

t.start();

System.out.print("two. ");

}

public void run()

{

System.out.print("Thread ");

}

}

A.Compilation fails

B.An exception occurs at runtime.

C.It prints "Thread one. Thread two."

D.The output cannot be determined.

Answer: Option B

Explanation:

When the start() method is attempted a second time on a single Thread object, the method will throw an IllegalThreadStateException (you will not need to know this exception name for the exam). Even if the thread has finished running, it is still illegal to call start() again.

Q.What will be the output of the program? //EXACTLY SAME QUESTION

try

{

int x = 0;

int y = 5 / x;

}

catch (Exception e)

{

System.out.println("Exception");

}

catch (ArithmeticException ae)

{

System.out.println(" Arithmetic Exception");

}

System.out.println("finished");

A.finished

B.Exception

C.Compilation fails.

D.Arithmetic Exception

Answer: Option C

Explanation:

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

Q.

public class Test { }

What is the prototype of the default constructor?

A.Test( )

B.Test(void)

C.public Test( )

D.public Test(void)

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.

Q.You want subclasses in any package to have access to members of a superclass. Which is the most restrictive access that accomplishes this objective? //EXACT YA SIMILAR

A.public

B.private

C.protected

D.transient

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.

Q.Given a method in a protected class, what access modifier do you use to restrict access to that method to only the other members of the same class? //EXACT YA SIMILAR

A.final

B.static

C.private

D.protected

E.volatile

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.

Q.class A //EXACTLY SAME QUESTION

{

protected int method1(int a, int b)

{

return 0;

}

}

Which is valid in a class that extends class A?

A.public int method1(int a, int b) {return 0; }

B.private int method1(int a, int b) { return 0; }

C.public short method1(int a, int b) { return 0; }

D.static protected int method1(int a, int b) { return 0; }

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.

Q.What will be the output of the program? // EXACT YA SIMILAR

public class BoolTest

{

public static void main(String [] args)

{

int result = 0;

Boolean b1 = new Boolean("TRUE");

Boolean b2 = new Boolean("true");

Boolean b3 = new Boolean("tRuE");

Boolean b4 = new Boolean("false");

if (b1 == b2) /\* Line 10 \*/

result = 1;

if (b1.equals(b2) ) /\* Line 12 \*/

result = result + 10;

if (b2 == b4) /\* Line 14 \*/

result = result + 100;

if (b2.equals(b4) ) /\* Line 16 \*/

result = result + 1000;

if (b2.equals(b3) ) /\* Line 18 \*/

result = result + 10000;

System.out.println("result = " + result);

}

}

A.0

B.1

C.10

D.10010

Answer: Option D

Explanation:

Line 10 fails because b1 and b2 are two different objects. Lines 12 and 18 succeed because the Boolean String constructors are case insensitive. Lines 14 and 16 fail because true is not equal to false.

Q.What will be the output of the program?public class X //EXACT YA SIMILAR

{

public static void main(String [] args)

{

try

{

badMethod();

System.out.print("A");

}

catch (Exception ex)

{

System.out.print("B");

}

finally

{

System.out.print("C");

}

System.out.print("D");

}

public static void badMethod() {}

}

A.AC

B.BC

C.ACD

D.ABCD

Answer: Option C

Explanation:

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

Q.What will be the output of the program? //EXACT YA SIMILAR

public class ObjComp

{

public static void main(String [] args )

{

int result = 0;

ObjComp oc = new ObjComp();

Object o = oc;

if (o == oc)

result = 1;

if (o != oc)

result = result + 10;

if (o.equals(oc) )

result = result + 100;

if (oc.equals(o) )

result = result + 1000;

System.out.println("result = " + result);

}

}

A.1

B.10

C.101

D.1101

Answer: Option D

Explanation:

Even though o and oc are reference variables of different types, they are both referring to the same object. This means that == will resolve to true and that the default equals() method will also resolve to true.

Q.Which three are valid method signatures in an interface? //SIMILAR YA EXACT

private int getArea();

public float getVol(float x);

public void main(String [] args);

public static void main(String [] args);

boolean setFlag(Boolean [] test);

A.1 and 2

B.2, 3 and 5

C.3, 4, and 5

D.2 and 4

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.

Q.

Which one creates an instance of an array? //EXACTLY SAME QUESTION

A. int[ ] ia = new int[15];

B. float fa = new float[20];

C. char[ ] ca = "Some String";

D. int ia[ ] [ ] = { 4, 5, 6 }, { 1,2,3 };

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.

Q.Which four options describe the correct default values for array elements of the types indicated? //EXACTLY SAME QUESTION

1.int -> 0

2.String -> "null"

3.Dog -> null

4.char -> '\u0000'

5.float -> 0.0f

6.boolean -> true

A. 1, 2, 3, 4 B. 1, 3, 4, 5

C. 2, 4, 5, 6 D. 3, 4, 5, 6

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) is null, with no quotes.

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

Q.Which will legally declare, construct, and initialize an array? //EXACTLY SAME QUESTION

A. int [] myList = {"1", "2", "3"};

B. int [] myList = (5, 8, 2);

C. int myList [] [] = {4,9,7,0};

D. int myList [] = {4, 3, 7};

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.

Q.Which three are legal array declarations? //EXACTLY SAME QUESTION

1.int [] myScores [];

2.char [] myChars;

3.int [6] myScores;

4.Dog myDogs [];

5.Dog myDogs [7];

A. 1, 2, 4 B. 2, 4, 5

C. 2, 3, 4 D. All are correct.

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)

Q.public interface Foo //EXACTLY SAME QUESTION

{

int k = 4; /\* Line 3 \*/

}

Which three piece of codes are equivalent to line 3?

1.final int k = 4;

2.public int k = 4;

3.static int k = 4;

4.abstract int k = 4;

5.volatile int k = 4;

6.protected int k = 4;

A. 1, 2 and 3 B. 2, 3 and 4

C. 3, 4 and 5 D. 4, 5 and 6

Answer: Option A

Explanation:

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

Q.Which one of the following will declare an array and initialize it with five numbers? //EXACTLY SAME QUESTION

A. Array a = new Array(5);

B. int [] a = {23,22,21,20,19};

C. int a [] = new int[5];

D. int [5] array;

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.

Q.Which is the valid declarations within an interface definition? //EXACTLY SAME QUESTION

A. public double methoda();

B. public final double methoda();

C. static void methoda(double d1);

D. protected void methoda(double d1);

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 declared abstract.

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.