LANGUAGE FUNDAMENTALS

1.

Which four options describe the correct default values for array elements of the types indicated?

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.

2.

Which one of these lists contains only Java programming language keywords?

a) class, if, void, long, Int, continue
b) goto, instanceof, native, finally, default, throws
c) try, virtual, throw, final, volatile, transient
d) strictfp, constant, super, implements, do
e) byte, break, assert, switch, include

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 marked static and final.

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

3.

Which will legally declare, construct, and initialize an array?

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.

4.

Which is a reserved word in the Java programming language?

a) method
b) native
c) subclasses
d) reference
e) array

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.

Which is a valid keyword in java?

a) interface
b) string
c) Float
d) unsigned

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 is float.

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

6.

Which three are legal array declarations?

1. int [] myScores [];
2. char [] myChars;
3. int [6] myScores;
4. Dog myDogs [];
5. Dog myDogs [7];

1, 2, 4

2, 4, 5

2, 3, 4

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).

7.

public interface Foo

{

```
  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.

8.

Which one of the following will declare an array and initialize it with five numbers?

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.

9.

Which three are valid declarations of a char?

1. char c1 = 064770;
2. char c2 = 'face';
3. char c3 = 0xbeef;
4. char c4 = \u0022;
5. char c5 = '\iface';
6. char c6 = '\uface';

a) 1, 2, 4
b) 1, 3, 6
c) 3, 5
d) 5 only

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.

Which is the valid declarations within an interface definition?

public double methoda();

public final double methoda();

static void methoda(double d1);

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.

11.

Which one is a valid declaration of a boolean?

boolean b1 = 0;

boolean b2 = 'false';

boolean b3 = false;

boolean b4 = Boolean.false();

boolean b5 = no;

Answer: Option C

Explanation:

A boolean can only be assigned the literal true or false

12.

Which three are valid declarations of a float?

1. float f1 = -343;
2. float f2 = 3.14;
3. float f3 = 0x12345;
4. float f4 = 42e7;
5. float f5 = 2001.0D;
6. float f6 = 2.81F;

a) 1, 2, 4
b) 2, 3, 5
c) 1, 3, 6
d) 2, 4, 6

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 float rather than a double (the default for floating point literals).

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

13.

Which is a valid declarations of a String?

a) String s1 = null;
b) String s2 = 'null';
c) String s3 = (String) 'abc';
d) String s4 = (String) '\ufeed';

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.

What is the numerical range of a char?

a) -128 to 127
b) $-(2^{15)}$ to $(2^{15}) - 1$
c) 0 to 32767
d) 0 to 65535

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.

DECLARATION AND ACCESS CONTROL:

1.
You want subclasses in any package to have access to members of a superclass. Which is the most restrictive access that accomplishes this objective?

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.

2.

```
public class Outer
{
    public void someOuterMethod()
```

```
    {
        //Line 5
    }
    public class Inner { }

    public static void main(String[] argv)
    {
        Outer ot = new Outer();
        //Line 10
    }
}
```

Which of the following code fragments inserted, will allow to compile?

a) new Inner(); //At line 5
b) new Inner(); //At line 10
c) new ot.Inner(); //At line 10
d) new Outer.Inner(); //At line 10

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.

```
interface Base
{
    boolean m1 ();
    byte m2(short s);
}
```

which two code fragments will compile?

1. `interface Base2 implements Base {}`
2. `abstract class Class2 extends Base { public boolean m1(){ return true; }}`
3. `abstract class Class2 implements Base {}`
4. `abstract class Class2 implements Base { public boolean m1(){ return (7 > 4); }}`
5. `abstract class Class2 implements Base { protected boolean m1(){ return (5 > 7) }}`

1 and 2

2 and 3

3 and 4

1 and 5

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 implicitly `public`, so the methods being implemented must be `public`.

4.
Which three form part of correct array declarations?

1. public int a [ ]
2. static int [ ] a
3. public [ ] int a
4. private int a [3]
5. private int [3] a [ ]
6. public final int [ ] a

1, 3, 4

2, 4, 5

1, 2, 6

2, 5, 6

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.

```
public class Test { }
```

What is the prototype of the default constructor?
Test( )

a) Test(void)
b) public Test( )
c) public Test(void)

Answer: Option C

6.
What is the most restrictive access modifier that will allow members of one class to have access to members of another class in the same package?

a) public
b) abstract
c) protected
d) synchronized
e) default access

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 Answer: Option
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 abstractandsynchronized are not access modifiers..

7.
Which of the following is/are legal method declarations?

1. protected abstract void m1();
2. static final void m1(){}
3. synchronized public final void m1() {}
4. private native void m1();

a) 1 and 3
b) 2 and 4
c) 1 only
d) All of them are legal declarations.

Answer: Option d
Explanation:
All the given statements are legal declarations.

8.
Which cause a compiler error?

a) int[ ] scores = {3, 5, 7};
b) int [ ][ ] scores = {2,7,6}, {9,3,45};
c) String cats[ ] = {"Fluffy", "Spot", "Zeus"};
d) boolean results[ ] = new boolean [] {true, false, true};
e) Integer results[ ] = {new Integer(3), new Integer(5), new Integer(8)};

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.
Which three are valid method signatures in an interface?

1. private int getArea();
2. public float getVol(float x);
3. public void main(String [] args);
4. public static void main(String [] args);
5. 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.

10.
You want a class to have access to members of another class in the same package. Which is the most restrictive access that accomplishes this objective?

a) public
b) private
c) protected
d) default access

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.
What is the widest valid returnType for methodA in line 3?

```
public class ReturnIt
{
   returnType methodA(byte x, double y) /* Line 3 */
   {
      return (long)x / y * 2;
   }
}
```

a) int
b) byte
c) long
d) double

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 to double.
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.

```
class A
{
   protected int method1(int a, int b)
   {
      return 0;
   }
```

}

a) Which is valid in a class that extends class A?
b) public int method1(int a, int b) {return 0; }
c) private int method1(int a, int b) { return 0; }
d) public short method1(int a, int b) { return 0; }
e) 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.

13.

Which one creates an instance of an array?

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.

14.

Which two of the following are legal declarations for nonnested classes and interfaces?

1. final abstract class Test {}
2. public static interface Test {}
3. final public class Test {}
4. protected abstract class Test {}
5. protected interface Test {}

6. abstract public class Test {}
    a. 1 and 4
    b. 2 and 5
    C 3 and 6
    D 4 and 6

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 as static. (4) and (5) are wrong because classes and interfaces cannot be marked as protected.

15.

Which of the following class level (nonlocal) variable declarations will not compile?

a) protected int a;
b) transient int b = 3;
c) private synchronized int e;
d) volatile int d;

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.

Which two cause a compiler error?

1. float[ ] f = new float(3);
2. float f2[ ] = new float[ ];
3. float[ ]f1 = new float[3];
4. float f3[ ] = new float[3];
5. float f5[ ] = {1.0f, 2.0f, 2.0f};

a) 2, 4
b) 3, 5
c) 4, 5
d) 1, 2

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.
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?

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.

18.
Which is a valid declaration within an interface?

a) public static short stop = 23;
b) protected short stop = 23;
c) transient short stop = 23;
d) final void madness(short stop);

Answer: Option A
Explanation:
(A) is valid interface declarations.
(B) and (C) are incorrect because interface variables cannot be either protected or transient. (D) is incorrect because interface methods cannot be final or static.

FLOW CONTROL:

1.

```
public void foo( boolean a, boolean b)
{
   if( a )
   {
      System.out.println("A"); /* Line 5 */
   }
   else if(a && b) /* Line 7 */
   {
      System.out.println( "A && B");
   }
   else /* Line 11 */
```

```
  {
    if ( !b )
    {
      System.out.println( "notB") ;
    }
    else
    {
      System.out.println( "ELSE" ) ;
    }
  }
}
```

a) If a is true and b is true then the output is "A && B"
b) If a is true and b is false then the output is "notB"
c) If a is false and b is true then the output is "ELSE"
d) If a is false and b is false then the output is "ELSE"

Explanation: C

Option C is correct. The output is "ELSE". Only when a is false do the output lines after 11 get some chance of executing.

Option A is wrong. The output is "A". When a is true, irrespective of the value of b, only the line 5 output will be executed. The condition at line 7 will never be evaluated (when a is true it will always be trapped by the line 12 condition) therefore the output will never be "A && B".

Option B is wrong. The output is "A". When a is true, irrespective of the value of b, only the line 5 output will be executed.

Option D is wrong. The output is "notB".

2.

```
switch(x)
{
  default:
    System.out.println("Hello");
}
```

Which two are acceptable types for x?

1. byte

2. long
3. char
4. float
5. Short
6. Long

a) 1 and 3
b) 2 and 4
c) 3 and 5
d) 4 and 6
OPTION: A

Explanation:

Switch statements are based on integer expressions and since both bytes and chars can implicitly be widened to an integer, these can also be used. Also shorts can be used. Short and Long are wrapper classes and reference types can not be used as variables.

3.

```
public void test(int x)
{
  int odd = 1;
  if(odd) /* Line 4 */
  {
    System.out.println("odd");
  }
  else
  {
    System.out.println("even");
  }
}
```

Which statement is true?

a) Compilation fails.
b) "odd" will always be output.
c) "even" will always be output.
d) "odd" will be output for odd values of x, and "even" for even values.
OPTION A

Explanation:

The compiler will complain because of incompatible types (line 4), the if expects a boolean but it gets an integer.

4.

```
public class While
{
   public void loop()
   {
      int x= 0;
      while ( 1 ) /* Line 6 */
      {
         System.out.print("x plus one is " + (x + 1)); /* Line 8 */
      }
   }
}
```

Which statement is true?

a) There is a syntax error on line 1.
b) There are syntax errors on lines 1 and 6.
c) There are syntax errors on lines 1, 6, and 8.
d) There is a syntax error on line 6.

Answer: Option D

Explanation:

Using the integer 1 in the while statement, or any other looping or conditional construct for that matter, will result in a compiler error. This is old C Program syntax, not valid Java.

A, B and C are incorrect because line 1 is valid (Java is case sensitive so While is a valid class name). Line 8 is also valid because an equation may be placed in a String operation as show

```
      }
      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).

7.

What will be the output of the program?

```
public class X
{
   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.

8.

What will be the output of the program?

```
public class X
{
  public static void main(String [] args)
  {
    try
    {
      badMethod(); /* Line 7 */
      System.out.print("A");
    }
    catch (Exception ex) /* Line 10 */
    {
      System.out.print("B"); /* Line 12 */
    }
    finally /* Line 14 */
    {
```

```
        System.out.print("C"); /* Line 16 */
      }
      System.out.print("D"); /* Line 18 */
   }
   public static void badMethod()
   {
      throw new RuntimeException();
   }
}
```

a) AB
b) BC
c) ABC
d) BCD

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.

What will be the output of the program?

```
public class MyProgram
{
   public static void main(String args[])
   {
      try
      {
         System.out.print("Hello world ");
      }
      finally
```

```
    {
      System.out.println("Finally executing ");
    }
  }
}
```

a) Nothing. The program will not compile because no exceptions are specified.
b) Nothing. The program will not compile because no catch clauses are specified.
c) Hello world.
d) Hello world Finally executing

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.

What will be the output of the program?

```
class Exc0 extends Exception { }
class Exc1 extends Exc0 { } /* Line 2 */
public class Test
{
  public static void main(String args[])
  {
    try
    {
      throw new Exc1(); /* Line 9 */
    }
    catch (Exc0 e0) /* Line 11 */
    {
      System.out.println("Ex0 caught");
    }
    catch (Exception e)
```

```
    {
      System.out.println("exception caught");
    }
  }
}
```

a) Ex0 caught
b) exception caught
c) Compilation fails because of an error at line 2.
d) Compilation fails because of an error at line 9.

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.

EXCEPTIONS:

1.

What will be the output of the program?

```
public class Foo
{
  public static void main(String[] args)
  {
    try
    {
      return;
    }
    finally
    {
      System.out.println( "Finally" );
    }
  }
}
```

a) Finally
b) Compilation fails.

c) The code runs with no output.
d) An exception is thrown at runtime.

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.

What will be the output of the program?

```
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).

3.

What will be the output of the program?

```
public class X
{
   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()
   {
      throw new Error(); /* Line 22 */
```

```
  }
}
```

a) ABCD
b) Compilation fails.
c) C is printed before exiting with an error message.
d) BC is printed before exiting with an error message.

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.

What will be the output of the program?

```
public class X
{
  public static void main(String [] args)
  {
    try
    {
      badMethod();
      System.out.print("A");
    }
    catch (RuntimeException ex) /* Line 10 */
    {
      System.out.print("B");
    }
    catch (Exception ex1)
    {
      System.out.print("C");
    }
    finally
```

```
    {
      System.out.print("D");
    }
    System.out.print("E");
  }
  public static void badMethod()
  {
    throw new RuntimeException();
  }
}
```

a) BD
b) BCD
c) BDE
d) BCDE

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.

What will be the output of the program?

```
public class RTExcept
{
  public static void throwit ()
  {
    System.out.print("throwit ");
    throw new RuntimeException();
  }
  public static void main(String [] args)
  {
    try
    {
```

```
        System.out.print("hello ");
        throwit();
      }
      catch (Exception re )
      {
        System.out.print("caught ");
      }
      finally
      {
        System.out.print("finally ");
      }
      System.out.println("after ");
    }
}
```

a) hello throwit caught
b) Compilation fails
c) hello throwit RuntimeException caught after
d) hello throwit caught finally after

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.

What will be the output of the program?

```
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).

7.

What will be the output of the program?

```
public class X
{
   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.

8.

What will be the output of the program?

```
public class X
{
   public static void main(String [] args)
   {
      try
      {
         badMethod(); /* Line 7 */
         System.out.print("A");
      }
      catch (Exception ex) /* Line 10 */
      {
         System.out.print("B"); /* Line 12 */
      }
      finally /* Line 14 */
      {
         System.out.print("C"); /* Line 16 */
      }
      System.out.print("D"); /* Line 18 */
   }
   public static void badMethod()
   {
      throw new RuntimeException();
   }
}
```

a) AB
b) BC
c) ABC

d) BCD

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.

What will be the output of the program?

```
public class MyProgram
{
   public static void main(String args[])
   {
      try
      {
         System.out.print("Hello world ");
      }
      finally
      {
         System.out.println("Finally executing ");
      }
   }
}
```

a) Nothing. The program will not compile because no exceptions are specified.
b) Nothing. The program will not compile because no catch clauses are specified.
c) Hello world.
d) Hello world Finally executing

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.

What will be the output of the program?

```
class Exc0 extends Exception { }

class Exc1 extends Exc0 { } /* Line 2 */

public class Test

{

   public static void main(String args[])

   {

      try

      {

         throw new Exc1(); /* Line 9 */

      }

      catch (Exc0 e0) /* Line 11 */

      {

         System.out.println("Ex0 caught");

      }

      catch (Exception e)

      {

         System.out.println("exception caught");

      }

   }

}
```

a) Ex0 caught
b) exception caught
c) Compilation fails because of an error at line 2.
d) Compilation fails because of an error at line 9.

Answer: Option

Explanation: A

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.

ASSERTIONS:

1.

What will be the output of the program?

```
public class Test
{
   public static void main(String[] args)
   {
      int x = 0;
      assert (x > 0) ? "assertion failed" : "assertion passed" ;
      System.out.println("finished");
   }
}
```

a) finished
b) Compiliation fails.
c) An AssertionError is thrown and finished is output.
d) An AssertionError is thrown with the message "assertion failed."

Answer: Option B

Explanation:

Compilation Fails. You can't use the Assert statement in a similar way to the ternary operator. Don't confuse.

2.

```
public class Test
{
   public void foo()
   {
      assert false; /* Line 5 */
      assert false; /* Line 6 */
   }
   public void bar()
   {
```

```
        while(true)
        {
            assert false; /* Line 12 */
        }
        assert false;  /* Line 14 */
    }
}
```

What causes compilation to fail?

a) Line 5
b) Line 6
c) Line 12
d) Line 14
OPTION: D

3.

What will be the output of the program?

```
public class Test
{
    public static int y;
    public static void foo(int x)
    {
        System.out.print("foo ");
        y = x;
    }
    public static int bar(int z)
    {
        System.out.print("bar ");
        return y = z;
    }
    public static void main(String [] args )
    {
        int t = 0;
```

```
        assert t > 0 : bar(7);
        assert t > 1 : foo(8); /* Line 18 */
        System.out.println("done ");
    }
}
```

a) bar
b) bar done
c) foo done
d) Compilation fails

Answer: Option D

Explanation:

The foo() method returns void. It is a perfectly acceptable method, but because it returns void it cannot be used in an assert statement, so line 18 will not compile.

4.

What will be the output of the program (when you run with the -ea option) ?

```
public class Test
{
    public static void main(String[] args)
    {
        int x = 0;
        assert (x > 0) : "assertion failed"; /* Line 6 */
        System.out.println("finished");
    }
}
```

a) finished
b) Compilation fails.
c) An AssertionError is thrown.
d) An AssertionError is thrown and finished is output.

Answer: Option C

Explanation:

An assertion Error is thrown as normal giving the output "assertion failed". The word "finished" is not printed (ensure you run with the -ea option)

Assertion failures are generally labeled in the stack trace with the file and line number from which they were thrown, and also in this case with the error's detail message "assertion failed". The detail message is supplied by the assert statement in line 6.

5.

```
public class Test2
{
   public static int x;
   public static int foo(int y)
   {
      return y * 2;
   }
   public static void main(String [] args)
   {
      int z = 5;
      assert z > 0; /* Line 11 */
      assert z > 2: foo(z); /* Line 12 */
      if ( z < 7 )
         assert z > 4; /* Line 14 */

      switch (z)
      {
         case 4: System.out.println("4 ");
         case 5: System.out.println("5 ");
         default: assert z < 10;
      }

      if ( z < 10 )
         assert z > 4: z++; /* Line 22 */
      System.out.println(z);
   }
}
```

which line is an example of an inappropriate use of assertions?

a) Line 11
b) Line 12
c) Line 14
d) Line 22

Answer: Option D

Explanation:

Assert statements should not cause side effects. Line 22 changes the value of z if the assert statement is false.

Option A is fine; a second expression in an assert statement is not required.

Option B is fine because it is perfectly acceptable to call a method with the second expression of an assert statement.

Option C is fine because it is proper to call an assert statement conditionally.

6.

Which three statements are true?

1. Assertion checking is typically enabled when a program is deployed.
2. It is never appropriate to write code to handle failure of an assert statement.
3. Assertion checking is typically enabled during program development and testing.
4. Assertion checking can be selectively enabled or disabled on a per-package basis, but not on a per-class basis.
5. Assertion checking can be selectively enabled or disabled on both a per-package basis and a per-class basis.

a) 1, 2 and 4
b) 2, 3 and 5
c) 3, 4 and 5
d) 1, 2 and 5

Answer: Option B

Explanation:

(1) is wrong. It's just not true.

(2) is correct. You're never supposed to handle an assertion failure.

(3) is correct. Assertions let you test your assumptions during development, but the assertion codeâ€”in effectâ€”evaporates when the program is deployed, leaving behind no overhead or debugging code to track down and remove.

(4) is wrong. See the explanation for (5) below.

(5) is correct. Assertion checking can be selectively enabled or disabled on a per-package basis. Note that the package default assertion status determines the assertion status for classes initialized in the future that belong to the named package or any of its "subpackages".

The assertion status can be set for a named top-level class and any nested classes contained therein. This setting takes precedence over the class loader's default assertion status, and over any applicable per-package default. If the named class is not a top-level class, the change of status will have no effect on the actual assertion status of any class.

Java.lang Class:

1.

What is the value of "d" after this line of code has been executed?

double d = Math.round (2.5 + Math.random());

a) 2
b) 3
c) 4
d) 2.5

Answer: Option B

Explanation:

The Math.random() method returns a number greater than or equal to 0 and less than 1 . Since we can then be sure that the sum of that number and 2.5 will be greater than or equal to 2.5 and less than 3.5, we can be sure that Math.round() will round that number to 3. So Option B is the answer.

2.

Which of the following would compile without error?

a) int a = Math.abs(-5);
b) int b = Math.abs(5.0);
c) int c = Math.abs(5.5F);
d) int d = Math.abs(5L);

Answer: Option A

Explanation:

The return value of the Math.abs() method is always the same as the type of the parameter passed into that method.

In the case of A, an integer is passed in and so the result is also an integer which is fine for assignment to "int a".

The values used in B, C & D respectively are a double, a float and a long. The compiler will complain about a possible loss of precision if we try to assign the results to an "int".

3.

Which of the following are valid calls to Math.max?

1. Math.max(1,4)
2. Math.max(2.3, 5)
3. Math.max(1, 3, 5, 7)
4. Math.max(-1.5, -2.8f)

a) 1, 2 and 4
b) 2, 3 and 4
c) 1, 2 and 3
d) 3 and 4

Answer: Option A

Explanation:

(1), (2), and (4) are correct. The max() method is overloaded to take two arguments of type int, long, float, or double.

(3) is incorrect because the max() method only takes two arguments.

4.

```
public class Myfile
{
   public static void main (String[] args)
   {
      String biz = args[1];
      String baz = args[2];
      String rip = args[3];
      System.out.println("Arg is " + rip);
   }
}
```

Select how you would start the program to cause it to print: Arg is 2

a) java Myfile 222
b) java Myfile 1 2 2 3 4
c) java Myfile 1 3 2 2
d) java Myfile 0 1 2 3

Answer: Option C

Explanation:

Arguments start at array element 0 so the fourth arguement must be 2 to produce the correct output.

Java.lang Class - Finding the output

1.

What will be the output of the program?

String x = new String("xyz");

String y = "abc";

x = x + y;

How many String objects have been created?

a) 2
b) 3
c) 4
d) 5

Answer: Option C

Explanation:

Line 1 creates two, one referred to by x and the lost String "xyz". Line 2 creates one (for a total of three). Line 3 creates one more (for a total of four), the concatenated String referred to by x with a value of "xyzabc".

2.

What will be the output of the program?

```
public class WrapTest
{
   public static void main(String [] args)
   {
      int result = 0;
      short s = 42;
      Long x = new Long("42");
      Long y = new Long(42);
      Short z = new Short("42");
      Short x2 = new Short(s);
```

```
    Integer y2 = new Integer("42");
    Integer z2 = new Integer(42);

    if (x == y) /* Line 13 */
      result = 1;
    if (x.equals(y) ) /* Line 15 */
      result = result + 10;
    if (x.equals(z) ) /* Line 17 */
      result = result + 100;
    if (x.equals(x2) ) /* Line 19 */
      result = result + 1000;
    if (x.equals(z2) ) /* Line 21 */
      result = result + 10000;

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

a) result = 1
b) result = 10
c) result = 11
d) result = 11010

Answer: Option B

Explanation:

Line 13 fails because == compares reference values, not object values. Line 15 succeeds because both String and primitive wrapper constructors resolve to the same value (except for the Character wrapper). Lines 17, 19, and 21 fail because the equals() method fails if the object classes being compared are different and not in the same tree hierarchy.

3.

What will be the output of the program?

```
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.

4.

What will be the output of the program?

```
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.

5.

What will be the output of the program?

```
public class Example
{
   public static void main(String [] args)
   {
      double values[] = {-2.3, -1.0, 0.25, 4};
      int cnt = 0;
      for (int x=0; x < values.length; x++)
      {
         if (Math.round(values[x] + .5) == Math.ceil(values[x]))
         {
            ++cnt;
         }
      }
      System.out.println("same results " + cnt + " time(s)");
   }
}
```

a) same results 0 time(s)
b) same results 2 time(s)
c) same results 4 time(s)
d) Compilation fails.

Answer: Option B

Explanation:

Math.round() adds .5 to the argument then performs a floor(). Since the code adds an additional .5 before round() is called, it's as if we are adding 1 then doing a floor(). The values that start out as integer values will in effect be incremented by 1 on the round() side but not on the ceil() side, and the noninteger values will end up equal.