**Arrays (1 Dimension)**

**Question 1**

```
class MWC101 {
  public static void main(String[] args) {
    int[] a1 = new int[];              // 1
    int a2[] = new int[5];             // 2
    int[] a3 = new int[]{1,2};         // 3
    int []a4 = {1,2};                  // 4
    int[] a5 = new int[5]{1,2,3,4,5};  // 5
}}
```

Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3
d. 4
e. 5

---

**Question 2**

```
class MWC102 {
  public static void main (String[] args) {
    byte[] a = new byte[1]; long[] b = new long[1];
    float[] c = new float[1]; Object[] d = new Object[1];
    System.out.print(a[0]+","+b[0]+","+c[0]+","+d[0]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0,0,null
b. Prints: 0,0,0.0,null

c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 3**

```
class MWC103 {
  public static void main(String[] args) {
    int[] a1 = {1,2,3};
    int []a2 = {4,5,6};
    int a3[] = {7,8,9};
    System.out.print(a1[1]+","+a2[1]+","+a3[1]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 12
b. Prints: 15
c. Prints: 18
d. Prints: 1,4,7
e. **Prints: 2,5,8**
f. Prints: 3,6,9
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 4**

```
class MWC104 {
  public static void main(String[] args) {
    int[5] a1;  // 1
    int []a2;   // 2
```

```
   int[   ]a3; // 3
   int a4[];   // 4
}}
```

A compile-time error is generated at which line?

a. **1**
b. 2
c. 3
d. 4
e. None of the above

---

**Question 5**

```
class MWC105 {
  static boolean b1;
  public static void main(String[] args) {
    boolean[] array = new boolean[1];
    boolean b2;
    System.out.print(b1+",");
    System.out.print(array[0]+",");
    System.out.print(b2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: true,true,true
b. Prints: false,false,false
c. Prints: null,null,null
d. Prints: false,true,false
e. Compile-time error
f. Run-time error
g. None of the above

1a e
1 5
An array creation expression must have either a dimension expression or an initializer. If both are present, then a compile-time error is generated. Similarly, if neither is present, then a compile-time error is generated. If only the dimension expression is present, then an array with the specified dimension is created with all elements set to the default values. If only the initializer is present, then an array will be created that has the required dimensions to accommodate the values specified in the initializer. Java avoids the possibility of an incompatible dimension expression and initializer by not allowing both to appear in the same array creation expression. A compile-time error is generated by the array creation expression for a1, because it needs either a dimension expression or an initializer. A compile-time error is generated at 5, because either the dimension expression or the initializer must be removed.

2b Prints: 0,0,0.0,null
Each array contains the default value for its type. The default value of a primitive byte or a primitive long is printed as 0. The default value of a float primitive is printed as 0.0. The default value of an Object is null and is printed as null.

3e Prints: 2,5,8
Arrays a1, a2 and a3 all contain 3 integers. The first element of the array has an index of 0 so an index of one refers to the second element of each array.

4a 1
A compile-time error occurs at the line marked 1, because the array reference declaration can not be used to declare the number of components contained in the array. Instead, the dimension expression should be contained in an array creation expression such as new int[5].

5e Compile-time error
Variable b1 is initialized to false, because it is a class member. The array component array[0] is initialized to the default value, false, of the array type, boolean[], even though the array is declared locally. Local variable b2 is not initialized, because it is local. A compile-time error is generated by the statement that attempts to print the value of b2.

**Arrays (Multi-Dimensional)**

**Question 1**

```
class MWC201 {
  public static void main(String[] args) {
    int[][] a1 = {{1,2,3},{4,5,6},{7,8,9,10}};
    System.out.print(a1[0][2]+","+a1[1][0]+","+a1[2][1]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 3,4,8
b. Prints: 7,2,6
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 2**

```
class MWC202 {
  public static void main(String[] args) {
    int[] a1[] = {{1,2},{3,4,5},{6,7,8,9}};
    int []a2[] = {{1,2},{3,4,5},{6,7,8,9}};
    int a3[][] = {{1,2},{3,4,5},{6,7,8,9}};
    System.out.print(a1[0][1]+","+a2[1][2]+","+a3[2][3]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 14
b. Prints: 16
c. Prints: 1,5,9
d. Prints: 2,4,8
e. Prints: 2,5,9
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 3**

```
class MWC207 {
```

```
  public static void main(String[] args) {
    int[][] a1 = {{1;2};{3;4;5};{6;7;8;9}};
    System.out.print(a1[0][1]+","+a1[1][2]+","+a1[2][3]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 14
b. Prints: 16
c. Prints: 1,5,9
d. Prints: 2,4,8
e. Prints: 2,5,9
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 4**

```
class MWC208 {
  public static void main(String[] args) {
    int[][] a1 = ((1,2),(3,4,5),(6,7,8,9));
    System.out.print(a1[0][1]+","+a1[1][2]+","+a1[2][3]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 14
b. Prints: 16
c. Prints: 1,5,9
d. Prints: 2,4,8
e. Prints: 2,5,9
f. Compile-time error
g. Run-time error

h. None of the above

---

**Question 5**

```
class MWC209 {
  public static void main(String[] args) {
    int[][] a1 = [[1,2],[3,4,5],[6,7,8,9]];
    int[][] a2 = [[1,2],[3,4,5],[6,7,8,9]];
    int[][] a3 = [[1,2],[3,4,5],[6,7,8,9]];
    System.out.print(a1[0][1]+","+a2[1][2]+","+a3[2][3]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 14
b. Prints: 16
c. Prints: 1,5,9
d. Prints: 2,4,8
e. Prints: 2,5,9
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 6**

```
class MWC210 {
  public static void main(String[] args) {
    int[] a2 = {1,2}, a3 = {3,4,5}, a4 = {6,7,8,9}; // 1
    int[][] a1 = {a2,a3,a4};                        // 2
    System.out.print(a1[0][1]+","+a1[1][2]+","+a1[2][3]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 14
b. Prints: 16
c. Prints: 1,5,9
d. Prints: 2,4,8
e. Prints: 2,5,9
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 7**

```
class MWC211 {
  public static void main(String[] args) {
    int a1[3];    // 1
    int []a2[];   // 2
    int[   ]a3;   // 3
    int[] a4[];   // 4
}}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. None of the above

---

**Question 8**

```
class MWC212 {
  public static void main(String[] args) {
    int[] a1[],a2[];  // 1
```

```
   int []a3,[]a4;    // 2
   int []a5,a6[];    // 3
   int[] a7,a8[];    // 4
}}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. None of the above

---

**Question 9**

```
class MWC203 {
  public static void main(String[] args) {
    int[] a1[] = {new int[]{1,2},new int[]{3,4,5}};
    int []a2[] = new int[][]{{1,2},{3,4,5}};
    int a3[][] = {{1,2},new int[]{3,4,5}};
    System.out.print(a1[0][1]+","+a2[1][0]+","+a3[1][2]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 14
b. Prints: 16
c. Prints: 1,2,4
d. Prints: 2,3,4
e. Prints: 2,3,5
f. Prints: 2,4,5
g. Compile-time error
h. Run-time error

i. None of the above

---

**Question 10**

```
class MWC204 {
  public static void main(String[] args) {
    int[][] a1 = {{1,2},{3,4,5},{6,7,8,9},{}};
    System.out.print(a1.length);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0
b. Prints: 3
c. Prints: 4
d. Prints: 9
e. Prints: 10
f. Prints: 11
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 11**

```
class MWC205 {
  public static void main(String[] args) {
    int[][] a1 = {{1,2},{3,4,5},{6,7,8,9},{}};
    for (int i = 0; i < a1.length; i++) {
      System.out.print(a1[i].length+",");
}}}
```

What is the result of attempting to compile and run the program?

a. Prints: 2,3,4,0,
b. Prints: 1,2,5,0,
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 12**

```
class MWC206 {
  public static void main (String[] args) {
    int[][] a1 = {{1,2,3},{4,5,6},{7,8,9}};
    for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
        System.out.print(a1[j][i]);
}}}}
```

What is the result of attempting to compile and run the program?

a. Prints: 123456789
b. Prints: 147258369
c. Prints: 321654987
d. Prints: 369258147
e. Run-time error
f. Compile-time error
g. None of the above

---

**Question 13**

```
class A11 {public String toString() {return "A11";}}
class A12 {
  public static void main(String[] arg) {
    A11[]     a1 = new A11[1];         // 1
```

```
   A11[][]   a2 = new A11[2][];       // 2
   A11[][][] a3 = new A11[3][][];     // 3
   a1[0] = new A11();                 // 4
   a2[0] = a2[1] = a1;                // 5
   a3[0] = a3[1] = a3[2] = a2;        // 6
   System.out.print(a3[2][1][0]);     // 7
}}
```

What is the result of attempting to compile and run the program?

a. Prints: null
b. Prints: A11
c. Compile-time error at 1.
d. Compile-time error at 2.
e. Compile-time error at 3.
f. Compile-time error at 4.
g. Compile-time error at 5.
h. Compile-time error at 6.
i. Compile-time error at 7.
j. Run-time error
k. None of the above

---

**Question 14**

```
class A13 {}
class A14 {
  public static void main(String[] arg) {
    A13[]     a1 = new A13[1];        // 1
    A13[][]   a2 = new A13[2][1];     // 2
    A13[][][] a3 = new A13[3][3][3];  // 3
    System.out.print(a3[2][2][2]);    // 4
    a1[0] = new A13();                // 5
    a2[0] = a2[1] = a1;               // 6
```

```
    a3[0] = a3[1] = a3[2] = a2;       // 7
    System.out.print(a3[2][2][2]);    // 8
}}
```

What is the result of attempting to compile and run the program?

a. Prints: null
b. Prints: nullnull
c. Compile-time error at 1.
d. Compile-time error at 2.
e. Compile-time error at 3.
f. Compile-time error at 4.
g. Compile-time error at 5.
h. Compile-time error at 6.
i. Compile-time error at 7.
j. Compile-time error at 8.
k. Run-time error

1 a Prints: 3,4,8 An array variable a1 is declared, and the declaration contains the initializer {{1,2,3},{4,5,6},{7,8,9,10}}. The initializer creates an array containing three components, and each is a reference to a subarray of type int[]. Each subarray contains components of type int, so the elements of the array referenced by a1 are of type int. The array access expression, a1[0][2] = a1[1st subarray][third component] = 3.

2 e Prints: 2,5,9 The declarations of the array variables, a1, a2 and a3, are different in terms of the position of the square brackets, but each is of type int[][]. The array access expression, a1[0][1] = a[1st subarray][2nd component] = 2. Each of the three array variable declarations has an array initializer. The same initializer, {{1,2},{3,4,5},{6,7,8,9}}, is used in each of the three cases. The initializer specifies three components of type int[], so each component is a reference to an array object containing components of type int. The size of each of the subarrays is different: The size of the first is 2, the second is 3, and the third is 4.

3 f Compile-time error The array initializer, {{1;2};{3;4;5};{6;7;8;9}} generates a compile-time error, because the commas have been replaced by semicolons. The array initializer should have been specified as follows: {{1,2},{3,4,5},{6,7,8,9}}.

4 f Compile-time error The array initializer, ((1,2),(3,4,5),(6,7,8,9)), generates a compile-time error, because the curly braces have been replaced by parentheses. The array initializer should have been specified as follows: {{1,2},{3,4,5},{6,7,8,9}}.

5 f Compile-time error The array initializer, [[1,2],[3,4,5],[6,7,8,9]], generates a compile-time error, because the curly braces have been replaced by square brackets. The array initializer should have been specified as follows: {{1,2},{3,4,5},{6,7,8,9}}.

6 e Prints: 2,5,9 The line marked 1 declares three array variables, a2, a3 and a4, and each references an array with components of type int. The line marked 2 declares an array variable, a1, where each component is initialized with a reference to one of the previously created arrays. The array access expression, a1[0][1], is evaluated as follows: a1[0][1] = a1[first subarray][second component] = 2.

7 a 1 A compile-time error occurs at the line marked 1, because the array variable declaration can not be used to specify the number of components contained in the array. Instead, the dimension expression should be contained in an array creation expression such as the following, new int[3].

8 b 2 An array variable is declared by placing brackets after the identifier or after the type name. A compile-time error occurs at the line marked 2, because the brackets appearing before the identifier for array variable a4 are not associated with the type or the identifier.

9 e Prints: 2,3,5 Each of the three array variable declarations, a1, a2 and a3, is different in terms of the position of the square brackets, but each declares a variable of type int[][]. Each of the three declarations contains an array initializer. In each case, the initializer could be simplified as follows: {{1,2},{3,4,5}}. The initializer creates an array containing two components, and each is a reference to an array containing components of type int. The size of each of the subarrays is different: The size of the first is 2 and the second is 3. The array access expression, a1[0][1] = a[1st subarray][2nd component] = 2.

10 c Prints: 4 The array initializer, {{1,2},{3,4,5},{6,7,8,9},{}}, creates an array containing four components, and each is a reference to a subarray of type int[]. The size of the array referenced by variable a1, is 4, because a1 contains four components. The size of the first subarray is 2, the second is 3, the third is 4 and the fourth is zero.

11 a Prints: 2,3,4,0, The array initializer, {{1,2},{3,4,5},{6,7,8,9},{}}, creates an array containing four components, and each is a reference to an array of type int[]. The size of the first subarray is 2, the second is 3, the third is 4, and the fourth is zero. While the components of the array referenced by a1 are of type int[], the elements of the array referenced by a1 are of type int.

12 b Prints: 147258369 The array variable a1 is declared with the initializer, {{1,2,3},{4,5,6},{7,8,9}}. The array access expression, a1[0][1] = a1[first subarray][second element] = 2. If the argument of the print statement had been a1[i][j] then the output would have been 123456789. The tricky feature of this question is the reversal of i and j to produce the deceptive array access expression, a1[j][i]. The output is 147258369.

13 b Prints: A11 The declaration A11[] a1 = new A11[1] declares a variable a1 that references an array that contains one component of type A11. The declaration A11[][] a2 = new A11[2][] declares a variable a2 that references an array that contains two components of type A11[]. In other words, the array referenced by a2 contains two reference variables, and each is able to reference a subarray. The initial value of the subarray references is null. The size of the subarrays has not been specified. The declaration A11[][][] a3 = new A11[3][][] declares a variable a3 that references an array that contains three components of type A11[][]. In other words, the array referenced by a3 contains three reference variables, and each is able to reference a subarray. The initial value of each subarray reference is null. The dimensions of the subarrays have not been specified. At line 5, a reference to the array referenced by a1 is assigned to each of the two components of the array referenced by a2, a2[0] = a2[1] = a1. At line 6, a reference to the array referenced by a2 is assigned to each of the three components of the array referenced by a3, a3[0] = a3[1] = a3[2] = a2. In other words, after line 6, each component of the array referenced by a3 is a reference to the array referenced by a2, and each element of the array referenced by a2 is a reference to the array referenced by a1, and the array referenced by a1 contains a reference to

an instance of class A11. Every element of the multi-dimensional array referenced by a3 contains a reference to a single instance of class A11. The print method invokes the toString method on the instance, and produces the output, A11.

14 a k Prints: null Run-time error The declaration A13[] a1 = new A13[1] declares a variable a1 that references an array that contains one component of type A13. The declaration A13[][] a2 = new A13[2][1] declares a variable a2 that references an array that contains two components of type A13[]. In other words, the array referenced by a2 contains two references to two subarrays of type A13[]. The size of each subarray is 1. The declaration A13[][][] a3 = new A13[3][3][3] declares a variable a3 that references an array that contains three components of type A13[][]. In other words, the array referenced by a3 contains three references to three subarrays of type A13[][]. The dimensions of the subarrays are 3 X 3. The dimensions of the array referenced by a3 are 3 X 3 X 3. The number of elements in the array referenced by a3 is 3 * 3 * 3 = 27 elements. Each of the three subarrays contains three components, where each is a reference to a subarray of type A13[]. Each of the 27 elements of the array referenced by a3 is a reference of type A13 that has been initialized to the default value of null. At line 7, a reference to the array referenced by a2 is assigned to each of the three components of the array referenced by a3, a3[0] = a3[1] = a3[2] = a2. Since the array referenced by a2 has dimensions 2 X 1, the array referenced by a3 now has dimensions 3 X 2 X 1. The number of elements is 6. An attempt to access any element beyond those 6 results in an exception at run-time.

**Arrays (Multi-Dimensional)**

**Question 1**

```
class MWC201 {
  public static void main(String[] args) {
    int[][] a1 = {{1,2,3},{4,5,6},{7,8,9,10}};
    System.out.print(a1[0][2]+","+a1[1][0]+","+a1[2][1]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 3,4,8
b. Prints: 7,2,6
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 2**

```
class MWC202 {
  public static void main(String[] args) {
    int[] a1[] = {{1,2},{3,4,5},{6,7,8,9}};
    int []a2[] = {{1,2},{3,4,5},{6,7,8,9}};
    int a3[][] = {{1,2},{3,4,5},{6,7,8,9}};
    System.out.print(a1[0][1]+","+a2[1][2]+","+a3[2][3]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 14
b. Prints: 16
c. Prints: 1,5,9
d. Prints: 2,4,8
e. Prints: 2,5,9
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 3**

```
class MWC207 {
  public static void main(String[] args) {
    int[][] a1 = {{1;2};{3;4;5};{6;7;8;9}};
    System.out.print(a1[0][1]+","+a1[1][2]+","+a1[2][3]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 14
b. Prints: 16

c. Prints: 1,5,9
d. Prints: 2,4,8
e. Prints: 2,5,9
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 4**

```
class MWC208 {
  public static void main(String[] args) {
    int[][] a1 = ((1,2),(3,4,5),(6,7,8,9));
    System.out.print(a1[0][1]+","+a1[1][2]+","+a1[2][3]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 14
b. Prints: 16
c. Prints: 1,5,9
d. Prints: 2,4,8
e. Prints: 2,5,9
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 5**

```
class MWC209 {
  public static void main(String[] args) {
    int[][] a1 = [[1,2],[3,4,5],[6,7,8,9]];
    int[][] a2 = [[1,2],[3,4,5],[6,7,8,9]];
```

```
   int[][] a3 = [[1,2],[3,4,5],[6,7,8,9]];
   System.out.print(a1[0][1]+","+a2[1][2]+","+a3[2][3]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 14
b. Prints: 16
c. Prints: 1,5,9
d. Prints: 2,4,8
e. Prints: 2,5,9
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 6**

```
class MWC210 {
  public static void main(String[] args) {
    int[] a2 = {1,2}, a3 = {3,4,5}, a4 = {6,7,8,9}; // 1
    int[][] a1 = {a2,a3,a4};                        // 2
    System.out.print(a1[0][1]+","+a1[1][2]+","+a1[2][3]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 14
b. Prints: 16
c. Prints: 1,5,9
d. Prints: 2,4,8
e. Prints: 2,5,9
f. Compile-time error
g. Run-time error

h. None of the above

---

**Question 7**

```
class MWC211 {
  public static void main(String[] args) {
    int a1[3];    // 1
    int []a2[];   // 2
    int[    ]a3;  // 3
    int[] a4[];   // 4
}}
```

A compile-time error is generated at which line?

a. 1

b. 2

c. 3

d. 4

e. None of the above

---

**Question 8**

```
class MWC212 {
  public static void main(String[] args) {
    int[] a1[],a2[];  // 1
    int []a3,[]a4;    // 2
    int []a5,a6[];    // 3
    int[] a7,a8[];    // 4
}}
```

A compile-time error is generated at which line?

a. 1

b. 2

c. 3

d. 4

e. None of the above

---

**Question 9**

```
class MWC203 {
  public static void main(String[] args) {
    int[] a1[] = {new int[]{1,2},new int[]{3,4,5}};
    int []a2[] = new int[][]{{1,2},{3,4,5}};
    int a3[][] = {{1,2},new int[]{3,4,5}};
    System.out.print(a1[0][1]+","+a2[1][0]+","+a3[1][2]);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 14

b. Prints: 16

c. Prints: 1,2,4

d. Prints: 2,3,4

e. Prints: 2,3,5

f. Prints: 2,4,5

g. Compile-time error

h. Run-time error

i. None of the above

---

**Question 10**

```
class MWC204 {
  public static void main(String[] args) {
    int[][] a1 = {{1,2},{3,4,5},{6,7,8,9},{}};
```

```
    System.out.print(a1.length);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0
b. Prints: 3
c. Prints: 4
d. Prints: 9
e. Prints: 10
f. Prints: 11
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 11**

```
class MWC205 {
  public static void main(String[] args) {
    int[][] a1 = {{1,2},{3,4,5},{6,7,8,9},{}};
    for (int i = 0; i < a1.length; i++) {
      System.out.print(a1[i].length+",");
}}}
```

What is the result of attempting to compile and run the program?

a. Prints: 2,3,4,0,
b. Prints: 1,2,5,0,
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 12**

```
class MWC206 {
  public static void main (String[] args) {
    int[][] a1 = {{1,2,3},{4,5,6},{7,8,9}};
    for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
        System.out.print(a1[j][i]);
}}}}
```

What is the result of attempting to compile and run the program?

a. Prints: 123456789
b. Prints: 147258369
c. Prints: 321654987
d. Prints: 369258147
e. Run-time error
f. Compile-time error
g. None of the above

---

**Question 13**

```
class A11 {public String toString() {return "A11";}}
class A12 {
  public static void main(String[] arg) {
    A11[]     a1 = new A11[1];         // 1
    A11[][]   a2 = new A11[2][];       // 2
    A11[][][] a3 = new A11[3][][];     // 3
    a1[0] = new A11();                 // 4
    a2[0] = a2[1] = a1;                // 5
    a3[0] = a3[1] = a3[2] = a2;        // 6
    System.out.print(a3[2][1][0]);     // 7
}}
```

What is the result of attempting to compile and run the program?

a. Prints: null
b. Prints: A11
c. Compile-time error at 1.
d. Compile-time error at 2.
e. Compile-time error at 3.
f. Compile-time error at 4.
g. Compile-time error at 5.
h. Compile-time error at 6.
i. Compile-time error at 7.
j. Run-time error
k. None of the above

---

**Question 14**

```
class A13 {}
class A14 {
  public static void main(String[] arg) {
    A13[]     a1 = new A13[1];         // 1
    A13[][]   a2 = new A13[2][1];      // 2
    A13[][][] a3 = new A13[3][3][3];   // 3
    System.out.print(a3[2][2][2]);     // 4
    a1[0] = new A13();                 // 5
    a2[0] = a2[1] = a1;                // 6
    a3[0] = a3[1] = a3[2] = a2;        // 7
    System.out.print(a3[2][2][2]);     // 8
}}
```

What is the result of attempting to compile and run the program?

a. Prints: null

b. Prints: nullnull
c. Compile-time error at 1.
d. Compile-time error at 2.
e. Compile-time error at 3.
f. Compile-time error at 4.
g. Compile-time error at 5.
h. Compile-time error at 6.
i. Compile-time error at 7.
j. Compile-time error at 8.
k. Run-time error

1a Prints: 3,4,8 An array variable a1 is declared, and the declaration contains the initializer {{1,2,3},{4,5,6},{7,8,9,10}}. The initializer creates an array containing three components, and each is a reference to a subarray of type int[]. Each subarray contains components of type int, so the elements of the array referenced by a1 are of type int. The array access expression, a1[0][2] = a1[1st subarray][third component] = 3.

2e Prints: 2,5,9 The declarations of the array variables, a1, a2 and a3, are different in terms of the position of the square brackets, but each is of type int[][]. The array access expression, a1[0][1] = a[1st subarray][2nd component] = 2. Each of the three array variable declarations has an array initializer. The same initializer, {{1,2},{3,4,5},{6,7,8,9}}, is used in each of the three cases. The initializer specifies three components of type int[], so each component is a reference to an array object containing components of type int. The size of each of the subarrays is different: The size of the first is 2, the second is 3, and the third is 4.

3f Compile-time error The array initializer, {{1;2};{3;4;5};{6;7;8;9}} generates a compile-time error, because the commas have been replaced by semicolons. The array initializer should have been specified as follows: {{1,2},{3,4,5},{6,7,8,9}}.

4f Compile-time error The array initializer, ((1,2),(3,4,5),(6,7,8,9)), generates a compile-time error, because the curly braces have been replaced by parentheses. The array initializer should have been specified as follows: {{1,2},{3,4,5},{6,7,8,9}}.

5f Compile-time error The array initializer, [[1,2],[3,4,5],[6,7,8,9]], generates a compile-time error, because the curly braces have been replaced by square brackets. The array initializer should have been specified as follows: {{1,2},{3,4,5},{6,7,8,9}}.

6e Prints: 2,5,9 The line marked 1 declares three array variables, a2, a3 and a4, and each references an array with components of type int. The line marked 2 declares an array variable, a1, where each component is initialized with a reference to one of the previously created arrays. The array access expression, a1[0][1], is evaluated as follows: a1[0][1] = a1[first subarray][second component] = 2.

7a 1 A compile-time error occurs at the line marked 1, because the array variable declaration can not be used to specify the number of components contained in the array. Instead, the dimension expression should be contained in an array creation expression such as the following, new int[3].

8b  2 An array variable is declared by placing brackets after the identifier or after the type name. A compile-time error occurs at the line marked 2, because the brackets appearing before the identifier for array variable a4 are not associated with the type or the identifier.

9e  Prints: 2,3,5  Each of the three array variable declarations, a1, a2 and a3, is different in terms of the position of the square brackets, but each declares a variable of type int[][]. Each of the three declarations contains an array initializer. In each case, the initializer could be simplified as follows: {{1,2},{3,4,5}}. The initializer creates an array containing two components, and each is a reference to an array containing components of type int. The size of each of the subarrays is different: The size of the first is 2 and the second is 3. The array access expression, a1[0][1] = a[1st subarray][2nd component] = 2.

10c  Prints: 4  The array initializer, {{1,2},{3,4,5},{6,7,8,9},{}}, creates an array containing four components, and each is a reference to a subarray of type int[]. The size of the array referenced by variable a1, is 4, because a1 contains four components. The size of the first subarray is 2, the second is 3, the third is 4 and the fourth is zero.

11a
Prints: 2,3,4,0,
The array initializer, {{1,2},{3,4,5},{6,7,8,9},{}}, creates an array containing four components, and each is a reference to an array of type int[]. The size of the first subarray is 2, the second is 3, the third is 4, and the fourth is zero. While the components of the array referenced by a1 are of type int[], the elements of the array referenced by a1 are of type int.

12b  Prints: 147258369  The array variable a1 is declared with the initializer, {{1,2,3},{4,5,6},{7,8,9}}. The array access expression, a1[0][1] = a1[first subarray][second element] = 2. If the argument of the print statement had been a1[i][j] then the output would have been 123456789. The tricky feature of this question is the reversal of i and j to produce the deceptive array access expression, a1[j][i]. The output is 147258369.

13b  Prints: A11  The declaration A11[] a1 = new A11[1] declares a variable a1 that references an array that contains one component of type A11. The declaration A11[][] a2 = new A11[2][] declares a variable a2 that references an array that contains two components of type A11[]. In other words, the array referenced by a2 contains two reference variables, and each is able to reference a subarray. The initial value of the subarray references is null. The size of the subarrays has not been specified. The declaration A11[][][] a3 = new A11[3][][] declares a variable a3 that references an array that contains three components of type A11[][]. In other words, the array referenced by a3 contains three reference variables, and each is able to reference a subarray. The initial value of each subarray reference is null. The dimensions of the subarrays have not been specified. At line 5, a reference to the array referenced by a1 is assigned to each of the two components of the array referenced by a2, a2[0] = a2[1] = a1. At line 6, a reference to the array referenced by a2 is assigned to each of the three components of the array referenced by a3, a3[0] = a3[1] = a3[2] = a2. In other words, after line 6, each component of the array referenced by a3 is a reference to the array referenced by a2, and each element of the array referenced by a2 is a reference to the array referenced by a1, and the array referenced by a1 contains a reference to an instance of class A11. Every element of the multi-dimensional array referenced by a3 contains a reference to a single instance of class A11. The print method invokes the toString method on the instance, and produces the output, A11.

14a  k  Prints: null  Run-time error  The declaration A13[] a1 = new A13[1] declares a variable a1 that references an array that contains one component of type A13. The declaration A13[][] a2 = new A13[2][1] declares a variable a2 that references an array that contains two components of type A13[]. In other words, the

array referenced by a2 contains two references to two subarrays of type A13[]. The size of each subarray is 1. The declaration A13[][][] a3 = new A13[3][3][3] declares a variable a3 that references an array that contains three components of type A13[][]. In other words, the array referenced by a3 contains three references to three subarrays of type A13[][]. The dimensions of the subarrays are 3 X 3. The dimensions of the array referenced by a3 are 3 X 3 X 3. The number of elements in the array referenced by a3 is 3 * 3 * 3 = 27 elements. Each of the three subarrays contains three components, where each is a reference to a subarray of type A13[]. Each of the 27 elements of the array referenced by a3 is a reference of type A13 that has been initialized to the default value of null. At line 7, a reference to the array referenced by a2 is assigned to each of the three components of the array referenced by a3, a3[0] = a3[1] = a3[2] = a2. Since the array referenced by a2 has dimensions 2 X 1, the array referenced by a3 now has dimensions 3 X 2 X 1. The number of elements is 6. An attempt to access any element beyond those 6 results in an exception at run-time.

**Class Declarations**

**Question 1**

```
public class Basics {}      // 1   9036810446
class Basics1 {}            // 2
protected class Basics2 {}  // 3
private class Basics3 {}    // 4
Class Basics4 {}            // 5
```

Suppose these are top-level class declarations and not nested class declarations; and suppose that all of the declarations are contained in one file named Basics.java. Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3
d. 4
e. 5

---

**Question 2**

```
public class Basics {}   // 1
public class Basics2 {}  // 2
public class Basics3 {}  // 3
public class Basics4 {}  // 4
```

Suppose these are top-level class declarations and not nested class declarations; and suppose that all of the declarations are contained in one file named Basics.java. A compile-time error is not generated at which line?

a. 1
b. 2
c. 3
d. 4
e. None of the above

---

**Question 3**

Which of the following modifiers can be applied to a class that is not a nested class?

a. Public
b. Protected
c. Private
d. Abstract
e. Static
f. Final

---

**Question 4**

Which of the follow statements is true.

a. An anonymous class can be declared abstract.
b. A local class can be declared abstract.
c. An abstract class can be instantiated.
d. An abstract class is implicitly final.
e. An abstract class must declare at least one abstract method.
f. An abstract class can not extend a concrete class.

---

**Question 5**

public class A {int i1; void m1() {}}

Which of the following statements are true?

a. class A extends Object.
b. Field i1 is implicitly public, because class A is public.
c. Method m1 is implicitly public, because class A is public.
d. The compiler will insert a default constructor implicitly.
e. The default constructor has no throws clause.
f. The default constructor of class A has package access.
g. The default constructor accepts one parameter for each field in class A.
h. The default constructor invokes the no-parameter constructor of the superclass.

---

**Question 6**

```
abstract class A {}   // 1
transient class G {}  // 2
private class C {}    // 3
static class E {}     // 4
```

Suppose these are top-level class declarations and not nested class declarations. Which of these declarations would not produce a compile-time error?

a. 1
b. 2
c. 3
d. 4
e. None of the above

---

**Question 7**

```
protected class D {}    // 1
synchronized class F {} // 2
volatile class H {}     // 3
final class B {}        // 4
```

Suppose these are top-level class declarations and not nested class declarations. Which of these declarations would not produce a compile-time error?

a. 1
b. 2
c. 3
d. 4
e. None of the above

1 c d e 3 4 5 If a class C is declared as a member of an enclosing class then C may be declared using no access modifier or any of the three access modifiers, private, protected or public. However, if class C is not a local class, anonymous class or a member of an enclosing class or interface; then C may be declared with the public modifier or with package access (i.e. no modifier). The other two access modifiers, private and protected, are not applicable to any class that is not a member class. The class declaration, Class Basics4 {}, generates a compile-time error, because all of the letters of the reserved word class must be lower case.

2 a 1 Only one class in a source code file can be declared public. The other classes may not be public. Therefore, the declarations for classes Basics2, Basics3 and Basics4 generate compile-time errors.

3 a d f public abstract final The access modifiers, protected and private, can be applied to a class that is a member of an enclosing class, but can not be applied to a local class or a class that is not nested inside another class. The static modifier can be applied to a class that is a member of an enclosing class, but can not be applied to a local class or a class that is not nested inside another class. The public modifier can be applied to a top level class to allow the class to be accessed from outside of the package. The abstract modifier prevents the class from being instantiated. An abstract class may include zero, one or more abstract methods. The final modifier prevents a class from being extended.

4 b A local class can be declared abstract. An anonymous class can not be extended; therefore, an anonymous class can not be declared abstract. A local class can be abstract. An abstract class can not be instantiated. If a class declaration contains an abstract method, then the class must also be declared abstract. A class can be declared abstract even if it does not contain an abstract method. An abstract class can never be declared final.

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

6 a 1 The modifiers, private and static, can be applied to a nested class, but can not be applied to a class that is not nested. A class that is not nested can have public or package access, but not private. The transient modifier can not be applied to any class; because it is a field modifier.

7 d 4 The modifier, protected, can not be applied to a top level class, but can be applied to a nested class. The synchronized modifier can not be applied to any class, because it is a method modifier. The modifier, volatile, can not be applied to any class; because it is a field modifier.

**Nested Classes**

**Question 1**

Which of the follow are true statements.

a. A nested class is any class that is declared within the body of another class or interface.
b. A nested class can not be declared within the body of an interface declaration.
c. An inner class is a nested class that is not static.
d. A nested class can not be declared static.
e. A named class is any class that is not anonymous.

---

**Question 2**

Which of the follow are true statements.

a. A local class is declared within a method, constructor or block.
b. An anonymous class is always a local class.
c. A local class is a nested class.
d. A local class is a member class.
e. A local class is always a named class.

---

**Question 3**

Which of the following are class modifiers? Select all that are applicable to a top-level class and all that are applicable to a nested class. It is not required that a modifier be applicable to both.

a. Abstract
b. Extends
c. final
d. implements
e. private
f. protected
g. public
h. static
i. synchronized
j. Transient
k. Volatile

---

**Question 4**

Which of the following modifiers can be applied to a member class?

a. Abstract
b. final
c. public
d. protected
e. private
f. Static
g. synchronized
h. transient

---

**Question 5**

Which of the following modifiers can be applied to a local class?

a. public
b. protected
c. private
d. abstract
e. Static
f. Final

---

**Question 6**

```
class Z {
  abstract class A {}     // 1
  final class B {}        // 2
  private class C {}      // 3
  protected class D {}    // 4
  public class E {}       // 5
}
```

Which class declaration results in a compile-time error?

a. 1
b. 2
c. 3
d. 4
e. 5
f. None of the above

---

**Question 7**

```
class Z {
  static class F {}       // 1
```

```
  synchronized class G {} // 2
  transient class H {}    // 3
  volatile class I {}     // 4
}
```

Which class declaration does not result in a compile-time error?

a. 1
b. 2
c. 3
d. 4
e. None of the above

---

**Question 8**

```
class Z {
  void m1() {
    abstract class A {}      // 1
    final class B {}         // 2
    private class C {}       // 3
    protected class D {}     // 4
    public class E {}        // 5
  }
}
```

Which class declarations result in compile-time errors?

a. 1
b. 2
c. 3
d. 4
e. 5

---

**Question 9**

```
class Z {
  void m1() {
    static class F {}        // 1
    synchronized class G {}  // 2
    transient class H {}     // 3
    volatile class I {}      // 4
    abstract class A {}      // 5
    final class B {}         // 6
  }
}
```

Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3
d. 4
e. 5
f. 6

---

**Question 10**

```
class A {
  A() {}       // 1
  int A;       // 2
  void A() {}  // 3
  class A {}   // 4
}
```

Which line results in a compile-time error?

a. 1

b. 2

c. 3

d. 4

e. None of the above

---

**Question 11**

```
class A {
  int B;       // 1
  void B() {}   // 2
  class B {}    // 3
}
```

Which line results in a compile-time error?

a. 1

b. 2

c. 3

d. None of the above

---

**Question 12**

```
abstract class A {              // 1
  private abstract void m1();    // 2
  private abstract class B {}    // 3
  private class C extends B {}   // 4
}
```

Which line results in a compile-time error?

a. 1

b. 2

c. 3

d. 4

e. None of the above.

---

**Question 13**

```
abstract class A {           // 1
  abstract final void m1();   // 2
  abstract final class B {}   // 3
  class C extends B {}        // 4
}
```

Which line does not result in a compile-time error?

a. 1

b. 2

c. 3

d. 4

e. None of the above

---

**Question 14**

```
abstract class A {                  // 1
  abstract synchronized void m1();   // 2
  abstract synchronized class B {}   // 3
  synchronized class C extends B {}  // 4
}
```

Which line does not result in a compile-time error?

a. 1

b. 2

c. 3

d. 4

e. None of the above

1 a c e

2 nested class is any class that is declared within the body of another class or interface. An inner class is a nested class that is not static. A named class is any class that is not anonymous. Every class declared within the body of another class or interface is known as a nested class. If the nested class does not have a name, then it is an anonymous class. If the nested class has a name, then it is not anonymous. If the nested class has a name and is not declared inside of a method, constructor or any block, then it is a member class. If a member class is not static, then it is an inner class. If a class is not nested, then it is a top level class. A nested class that is static is sometimes referred to as a top level class, but that usage of the term is confusing and should be avoided.

3 a c e

4 A local class is declared within a method, constructor or block. A local class is a nested class. A local class is always a named class. Every class declared within the body of another class is known as a nested class. If the nested class does not have a name, then it is an anonymous class. If the nested class has a name, then it is not anonymous. If the nested class has a name and is declared inside of a method, constructor or any block, then it is a local class. If a nested class does not have a name, then it can not be called a local class even if it is declared inside of a block. Therefore, an anonymous class is never called a local class. If the nested class has a name and is not declared inside of a method, constructor or any block, then it is a member class. If the class is not nested, then it is a top level class. Please note that this question is more rigorous than those that one might expect to find on the real exam. It has been included here as a convenient place to define some terms that are used to explain the answers to some of the other questions that appear in this mock exam.

3 a c e f g h abstract final private protected public static

The access modifiers, public, protected and private, can not be applied to a local class. The protected and private access modifiers can be applied to member classes, but not to a top level class. The public modifier can be applied to a top level class or a member class, but not a local class. The static modifier can be applied to a member class, but not to a local class or top level class. The keywords extends and implements are not modifiers. The synchronized modifier can be applied to a method, but not to a class. The modifiers, transient and volatile, can be applied to a field, but not to a class.

4 a b c d e f abstract final public protected private static

A nested class that has a name and is not a local class is a member class. A member class can be static or non-static. A non-static member class is also known as an inner class. All of the class modifiers may be applied to a member class. The modifiers, synchronized and transient, are not class modifiers.

5 d f abstract final

If a nested class has a name and is declared inside of a method, constructor or any block, then it is a local class. No access modifier can be applied to a local class declaration. A local class can not be static. A local class can be abstract, and can be final.

6 f None of the above

The following modifiers can be applied to a member class: abstract, private, protected, public, static and final.

7 a 1

Please note that this question asks which class declaration does **NOT** result in a compile-time error. The following modifiers can be applied to a member class: abstract, private, protected, public, static and final. The synchronized modifier can not be applied to any class, because it is a method modifier. The modifiers, transient and volatile, can not be applied to any class, because they are field modifiers.

8 c d e 3 4 5

The abstract and final modifiers can be applied to a method local class declaration.

9 a b c d 1 2 3 4

The modifiers, abstract and final, can be applied to a method local class declaration. The synchronized modifier can not be applied to a class, because it is a method modifier. The modifiers, transient and volatile, can not be applied to any class, because they are field modifiers.

10 d 4

A method and a field can share the same name, because they are used in different contexts and use different lookup procedures. They can even share the same name with the class in which they are declared. Please note that class names usually begin with an upper case letter while method and field names usually begin with a lower case letter. A nested class can not share the same name with its enclosing class.

11 d None of the above

A method, field, and a nested class can share the same name, because they are used in different contexts and use different lookup procedures. Please note that class names usually begin with an upper case letter while method and field names usually begin with a lower case letter. Also note that a nested class can not share the same name with its enclosing class; however, a method and field can share a name with the enclosing class. Even so, it is not a good idea to name a method with the name of the enclosing class, because it could be confused with a constructor.

12 b 2

An abstract method has no implementation, and is not useful until an extending class implements the method. Private methods are not inherited and can not be overridden. If an abstract method is declared private, then it can not be implemented in a subclass. Although a method may not be both private and abstract, a nested class can be; because another nested class can extend the abstract class and implement any abstract methods.

13 a 1

Please note that this question asks which line does **NOT** result in a compile-time error. An abstract method has no implementation and is not useful until an extending class implements the method. A final method can not be overridden by a subclass method. An abstract final method can not be implemented and is not legal. An abstract class may contain abstract method declarations and is assumed to be incomplete. A final class can not be extended. The implementation of an abstract final class could not be completed. The declaration of class C results in a compiler error, because a final class may not be listed in the extends clause.

14 a 1

Please note that this question asks which line does **NOT** result in a compile-time error. The modifier, synchronized, is a method modifier, but is not a class modifier. Any attempt to declare a synchronized class results in a compile-time error. Since the synchronized modifier specifies an implementation detail it makes no sense to use it with an abstract method that provides no implementation

**Anonymous Classes**

**Question 1**

Which of the following is a true statement?

a. An anonymous class can extend only the Object class.

b. An anonymous class can not implement an interface.
c. An anonymous class declaration can not have an implements clause.
d. An anonymous class declaration can name more than one interface in the implements clause.
e. The class instance creation expression for an anonymous class must never include arguments.
f. None of the above

---

**Question 2**

Which of the following are true statements?

a. An anonymous class is implicitly abstract.
b. An anonymous class is implicitly final.
c. An anonymous class is implicitly static.
d. A static reference variable can reference an instance of an anonymous class.
e. An anonymous class declaration must have at least one explicit constructor declaration.
f. An anonymous class declaration can have more than one explicit constructor declaration.

---

**Question 3**

```
abstract class A {
  private int x = 4, y = 2;
  public int x() {return x;}
  public void x(int x) {this.x = x;}
  public int y() {return y;}
  public void y(int y) {this.y = y;}
  public abstract int math();
}
class B {
  static A a1 = new A(2,1) {
    public A(int i1, int i2) {x(i1);y(i2);};
    public int math() {return x()+y();}
  };
  public static void main(String[] args) {
```

```
    System.out.print(a1.math());
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 8
b. Prints: 3122
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 4**

```
class A {
  private static int f1 = 1;
  private int f2 = 2;
  void m1(int p1, final int p2) {
    int l1 = 5;
    final int l2 = 6;
    Object x = new Object() {
      int a = f1;  // 1
      int b = f2;  // 2
      int c = p1;  // 3
      int d = p2;  // 4
      int e = l1;  // 5
      int f = l2;  // 6
};}}
```

Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3

d. 4

e. 5

f. 6

---

**Question 5**

```
abstract class A {
  private int x = 1, y = 1;
  public A(int x, int y) {this.x = x; this.y = y;}
  public abstract int math();
}
class B {
  static A a1 = new A(2,1) {public int math() {return x + y;}};
  static A a2 = new A(2,1) {public int math() {return x - y;}};
  static A a3 = new A(2,1) {public int math() {return x * y;}};
  static A a4 = new A(2,1) {public int math() {return x / y;}};
  public static void main(String[] args) {
    System.out.print("" + a1.math() + a2.math() +
                     a3.math() + a4.math());
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 3122

b. Prints: 2011

c. Compile-time error

d. Run-time error

e. None of the above

---

**Question 6**

```
abstract class A {
```

```
  private int x = 4, y = 2;
  public int x() {return x;}
  public void x(int x) {this.x = x;}
  public int y() {return y;}
  public void y(int y) {this.y = y;}
  public abstract int math();
}
class B {
  static A a1 = new A() {public int math() {return x()+y();}}
  static A a2 = new A() {public int math() {return x()-y();}}
  static A a3 = new A() {public int math() {return x()*y();}}
  static A a4 = new A() {public int math() {return x()/y();}}
  public static void main(String[] args) {
    System.out.print("" + a1.math() + a2.math() +
                       a3.math() + a4.math());
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 18
b. Prints: 6282
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 7**

```
class A {String m1() {return "A.m1";}}
interface B {String m2();}
class C {
  static class D extends A implements B {
    public String m1() {return "D.m1";}
    public String m2() {return "D.m2";}
  }
```

```
  static A a1 = new A() implements B {
    public String m1() {return "m1";}
    public String m2() {return "m2";}
  };
  public static void main(String[] args) {
    System.out.print(a1.m1() + "," + new C.D().m2());
}}
```

What is the result of attempting to compile and run the program?

a. Prints: m1,D.m2
b. Prints: A.m1,D.m2
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 8**

```
abstract class A {
  private int x = 4, y = 2;
  public A(int i1, int i2) {x=i1;y=i2;}
  public int x() {return x;}
  public void x(int x) {this.x = x;}
  public int y() {return y;}
  public void y(int y) {this.y = y;}
  public abstract int math();
}
class B {
  static A a1 = new A(2,1) {public int math() {return x()+y();}};
  static A a2 = new A(2,1) {public int math() {return x()-y();}};
  static A a3 = new A(2,1) {public int math() {return x()*y();}};
  static A a4 = new A(2,1) {public int math() {return x()/y();}};
  public static void main(String[] args) {
    System.out.print("" + a1.math() + a2.math() +
```

```
                a3.math() + a4.math());
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 8
b. Prints: 3122
c. Compile-time error
d. Run-time error
e. None of the above

1 c An anonymous class declaration can not have an implements clause. A class instance creation expression can create an instance of a named class or an anonymous class. For example, the class instance creation expression new Object() creates an instance of the class named Object. If a class body appears in the class instance creation expression, then an anonymous class is created. For example, the expression new Object() {void doNothing(){}} creates an instance of an anonymous class that extends Object and implements a method named doNothing. In other words, if a class name immediately follows the keyword new, then the anonymous class extends the named class. When a named class is being extended, then the class instance creation expression can contain an optional argument list. The arguments will be passed to the direct superclass constructor that has a matching parameter list. An anonymous class declaration can not have an implements clause or an extends clause.

2 b d An anonymous class is implicitly final. A static reference variable can reference an instance of an anonymous class.

An anonymous class can extend Object and implement an interface, or the anonymous class can extend a named class including Object. An anonymous class can not be extended; so it can not be abstract. An anonymous class declaration always creates an instance of a class; so it is not surprising that an anonymous class can not be declared static. Even so, a static reference variable can refer to an anonymous class. A constructor shares the same name as the class in which it is declared, but an anonymous class has no name. For that reason, it is not surprising that an anonymous class declaration can not contain an explicit constructor declaration. Instead, an anonymous class can contain an instance initializer.

3 c Compile-time error

An anonymous class declaration can not contain an explicit declaration of a constructor.

4 c e 3 5

Local method variables and method parameters are stored on the stack and go out of scope after the method is exited. Although a local reference variable is stored on the stack, the referenced object is stored on the heap; so the object can continue to exist long after the method runs to completion. An object that is instantiated within a method or block is not permitted to refer to a variable that is declared within the method or block unless the variable is declared final and the variable declaration precedes the creation of the object.

5 c Compile-time error

The instance variables x and y in class A are private, so they are not accessible to the anonymous subclasses. If you want to access the private variables of a superclass, then you will need to add accessor methods to the superclass such as getX and getY.

6 c Compile-time error When an anonymous class declaration is the last thing that appears in a statement, then a semicolon must follow the declaration. Anonymous class declarations provide an excellent opportunity for trick questions involving statements with missing semicolons.

7 c Compile-time error An anonymous class can extend Object and implement an interface, or the anonymous class can extend a named class including Object. An anonymous class declaration can not have an implements clause. In this case, the declaration of the anonymous class referenced by a1 generates a compile-time error as a result of the attempt to extend class A and implement interface B.
8 b Prints: 3122 The arguments that appear in the class instance creation expression of an anonymous class are passed to a constructor of the superclass.

**Constructors**

**Question 1**

```
class A {A(int i) {}  // 1
class B extends A {}   // 2
```

Which of the following statements are true?

a. The compiler attempts to create a default constructor for class A.
b. The compiler attempts to create a default constructor for class B.
c. Compile-time error at 1.
d. Compile-time error at 2.

---

**Question 2**

Which of the following modifiers can be applied to a constructor?

a. private
b. abstract
c. final
d. Volatile
e. Native
f. None of the above.

---

**Question 3**

Which of the following techniques can be used to prevent the instantiation of a class by any code outside of the class?

a. Do not declare any constructors.
b. Do not use a return statement in the constructor.
c. Declare all constructors using the keyword void to indicate that nothing is returned.
d. Declare all constructors using the private access modifier.
e. None of the above.

---

**Question 4**

Which of the following modifiers can be applied to a constructor?

a. protected
b. public
c. static
d. synchronized
e. Transient

---

**Question 5**

Which of the following statements are true?

a. The compiler will create a default constructor if no other constructor is declared.
b. The default constructor takes no arguments.
c. If a class A has a direct superclass, then the default constructor of class A invokes the no-argument constructor of the superclass.
d. The default constructor declares Exception in the throws clause.
e. The default constructor is always given the private access modifier.
f. The default constructor is always given the public modifier.
g. The default constructor is always given default package access.

---

**Question 6**

Suppose that the compiler generates a default constructor for a class. If a compile-time error is to be avoided, which of the following must be true?

a. The superclass must not have any constructor other than a default constructor.
b. The superclass must not have an accessible no-argument constructor.
c. The no-argument superclass constructor must not have a throws clause that includes a checked exception.
d. The no-argument superclass constructor must be declared private.
e. None of the above

---

**Question 7**

```
class A {A() throws Exception {}} // 1
class B extends A {B() throws Exception {}} // 2
class C extends A {} // 3
```

Which of the following statements is true?

a. Compile-time error at 1.
b. Compile-time error at 2.
c. Compile-time error at 3.
d. None of the above

1 b d

2 The compiler attempts to create a default constructor for class B. Compile-time error at 2. If no constructor is declared explicitly, then the compiler will implicitly create a default constructor that accepts no parameters, has no throws clause, and invokes its superclass constructor. Since class A has an explicitly declared constructor, the compiler will not create an implicit default constructor. Class B does not have an explicit constructor declaration, so the compiler attempts to create a default constructor. Since class A does not have a no-parameter constructor, the attempt by class B to invoke the no parameter constructor of A would fail. As a result, a compiler error is generated at marker 2.

2 a private

Constructors are not inherited and can not be overridden, so there is no need for the final modifier in a constructor declaration. Furthermore, an abstract constructor would be useless, since it could never be implemented. The volatile modifier can be applied to a field, but not to a constructor. Native constructors are not permitted, because it would be difficult for Java to verify that the native constructor properly invokes the superclass constructor.

3 d Declare all constructors using the private access modifier. If no constructors are declared explicitly; then the compiler will create one implicitly, and it will have the same access modifier as the class. The explicit declaration of any constructor will prevent the creation of a default constructor. If all

constructors are declared private, then code outside of the class will not have access to the constructors and will not have the ability to create an instance of the class. Constructors do not return a value and constructor declarations do not include a return type, so the keyword void is not applicable to a constructor declaration.

4 a b protected public

Constructors can not be inherited, so an abstract constructor would be useless, since it could never be implemented. A static constructor would also be useless--or nearly so--since it would be unable to access the non-static members of the new instance. An object is not available to multiple threads during the construction process, so the synchronized modifier would not provide additional protection. The transient modifier can be applied to a field, but not a constructor.

5 a b c The compiler will create a default constructor if no other constructor is declared. The default constructor takes no arguments. If a class A has a direct superclass, then the default constructor of class A invokes the no-argument constructor of the superclass. If no constructor is declared explicitly, then the compiler will implicitly insert a default constructor. The default constructor takes no arguments. The primordial class Object has no superclass; so the default constructor of type Object does not invoke a superclass constructor. If a class A has a direct superclass, then the default constructor of class A will invoke the no-argument superclass constructor. It is unlikely that the real exam would try to trick you with a question that requires you to know that the constructor of type Object does not invoke a superclass constructor. For the purposes of the real exam, it might be safer to overlook that particular unique feature of type Object. If a subclass constructor attempts to invoke the no-argument superclass constructor when none exists, then a compile-time error is generated. The access modifier implicitly assigned to the default constructor is the same as that assigned to the class. The default constructor does not have a throws clause. Consequently, a compile-time error is generated if the no-argument constructor of the superclass has a throws clause.

6 c The no-argument superclass constructor must not have a throws clause that includes a checked exception.

The default constructor takes no arguments, and it invokes the superclass constructor with no arguments. If the superclass does not have an accessible no-argument constructor, then a compile-time error is generated. The default constructor does not have a throws clause. Consequently, a compile-time error is generated if the no-parameter constructor of the superclass has a throws clause.

7 c Compile-time error at 3.

The compiler creates a constructor for class C implicitly. The implicitly created constructor accepts no parameters and has no throws clause. The constructors for class B and class C both invoke the constructor for A. The constructor for class A declares Exception in the throws clause. Since the constructors for B and C invoke the constructor for A implicitly, both B and C must declare Exception in their throws clause. A compile-time error is generated at marker 3, because the default constructor does not declare Exception in the throws clause.

## Field Declarations

**Question 1**

```
class JSC102 {
  public static void main (String[] args) {
    private int x = 1; protected int y = 2; public int z = 3;
    System.out.println(x+y+z);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 123
b. Prints: 1 2 3
c. Prints: 6
d. Run-time error
e. Compile-time error
f. None of the above

---

**Question 2**

```
class JSC105 {
  private static int x; protected static int y; public static int z;
  public static void main (String[] args) {System.out.println(x+y+z);}
}
```

What is the result of attempting to compile and run the program?

a. Prints nothing.
b. Prints an undefined value.
c. Prints: null
d. Prints: 0
e. Run-time error
f. Compile-time error
g. None of the above

---

**Question 3**

```
class Red {
  public int a; public static int b;
  public static void main (String[] in) {
    Red r1 = new Red(), r2 = new Red(); r1.a++; r1.b++;
```

```
    System.out.print(r1.a+", "+r1.b+", "+r2.a+", "+r2.b);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0, 0, 0, 0
b. Prints: 0, 1, 1, 1
c. Prints: 1, 1, 1, 0
d. Prints: 1, 1, 0, 1
e. Prints: 1, 1, 0, 0
f. Prints: 1, 1, 1, 1
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 4**

```
class Basics {
  int x = 1, y = 2;
  public static void main (String[] args) {System.out.println(x+y);}
}
```

What is the result of attempting to compile and run the program?

a. Prints: x+y
b. Prints: 12
c. Prints: 3
d. Run-time error
e. Compile-time error
f. None of the above

---

**Question 5**

Which of the following modifiers can be applied to the declaration of a field?

a. abstract
b. final
c. private
d. Protected
e. Public

---

**Question 6**

Which of the following modifiers can be applied to the declaration of a field?

a. Static
b. synchronized
c. Transient
d. Volatile
e. Native

---

**Question 7**

Which of the following is used to prevent the serialization of a non-static field?

a. Final
b. Protected
c. Synchronized
d. Transient
e. Volatile
f. Native

---

**Question 8**

Which of the following statements are true?

a. A value can not be assigned to a final field more than once.
b. A value can be assigned to a final field at any time or not at all.
c. Only static variables can be declared final.
d. A compile-time error is thrown if a blank final instance variable is not assigned a value before the end of each constructor.
e. A field can not be declared both final and volatile.

---

**Question 9**

```
class JSC101 {
  void m1() {
    public int a;      // 1
    protected int b;   // 2
    private int c;     // 3
    static int d;      // 4
    transient int e;   // 5
    volatile int f;    // 6
    final int g = 1;   // 7
}}
```

Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3
d. 4
e. 5
f. 6
g. 7

---

**Question 10**

```
class JSC103 {
  transient float e = 1;         // 1
  volatile float f = 1;          // 2
  abstract float j = 1;          // 3
  final float g = 1;             // 4
  private final float k = 1;     // 5
  private transient float l = 1; // 6
  volatile final float m = 1;    // 7
}
```

Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3
d. 4
e. 5
f. 6
g. 7

---

**Question 11**

```
class JSC104{
  public float a;        // 1
  protected float b;     // 2
  private float c;       // 3
  static float d;        // 4
  synchronized float i;  // 5
}
```

A compile-time error is generated at which line?

a. 1
b. 2

c. 3

d. 4

e. 5

f. None of the above.

---

**Question 12**

```
class Identifiers {
  int i1;   // 1
  int _i2;  // 2
  int i_3;  // 3
  int #i4;  // 4
  int $i5;  // 5
  int %i6;  // 6
  int i$7;  // 7
  int 8i;   // 8
}
```

Compile-time errors are generated at which lines?

a. 1

b. 2

c. 3

d. 4

e. 5

f. 6

g. 7

h. 8

1e
Compile-time error
The access modifiers public, protected and private, can not be applied to variables declared inside methods.

2d
Prints: 0
Member variables are initialized automatically. Type int variables are initialized to zero.

3d
Prints: 1, 1, 0, 1
Both instances of class Red share a single copy of the static field b. Although field b is only incremented using the r1 reference, the change is visible in the r2 instance of class Red.

4e
Compile-time error
A static method can not access a non-static variable.

5b c d e
final private protected public
A field is a class member. A static field is sometimes called a class variable. A non-static field is sometimes called an instance variable. A variable declaration that is immediately contained by a block such as a method body is called a local variable. The access modifiers, private, protected and public, can be applied to a field. A final field can not have its value assigned more than once. The abstract modifier may be applied to methods but not to fields.

6a c d
static transient volatile
A transient field is not part of the persistent state of an object. Transient fields are not serialized. Fields that are shared between threads may be marked volatile to force each thread to reconcile its own working copy of the field with the master copy stored in the main memory. The synchronized modifier may be applied to methods but not to fields.

7d
transient
A transient field is not part of the persistent state of an object, so it is not serialized. A static field is also not part of the persistent state of an object, and also is not serialized.

8a d e
A value can not be assigned to a final field more than once. A compile-time error is thrown if a blank final instance variable is not assigned a value before the end of each constructor. A field can not be declared both final and volatile.
Static and non-static field variables may be declared final. All final fields must be definitely assigned a value once and only once. If the declaration of a final variable does not include an initializer then the variable is called a blank final. All blank, final, static variables must be assigned in a static initializer. All blank final non-static variables must be assigned by the end of the instance construction process. A field is sometimes shared between threads. The volatile modifier is

used to force threads to reconcile their own working copy of a field with the master copy in main memory. If a field is declared final then its value does not change and there is no need for threads to reconcile their own working copies of the variable with the master copy in main memory.

9a b c d e f
1 2 3 4 5 6
A variable that is local to a method can not be accessed from outside of the class, so the access modifiers are not useful and not legal. A variable that is local to a method can not be part of the persistent state of an object, so the transient modifier is not useful and not legal. Local variables can not be shared between threads, so the volatile modifier is not useful and not legal. A local variable can be declared final to prevent its value from being assigned more than once. If the value of the variable needs to be accessed from a local class or an anonymous class, then the local variable or method parameter must be declared final.

10c g
3 7
The abstract modifier can be applied to a class and a method but not a field. A field is sometimes shared between threads. The volatile modifier is used to force threads to reconcile their own working copy of a field with the master copy in main memory. If a field is declared final then its value does not change and there is no need for threads to reconcile their own working copies of the variable with the master copy in main memory.

11e
5
The synchronized modifier is a method modifier and can not be applied to a field.

12d f h
4 6 8
The first letter of an identifier can be any *Unicode JLS 3.1* character that is a *Java letter*. The first letter can not be a number. For historical reasons, the dollar sign $ and underscore _ are considered Java letters along with many currency symbols in use in the world today.

**Method Declarations**

**Question 1**

Which of the following modifiers can not be applied to a method?

a. abstract
b. private
c. protected
d. public
e. Volatile

f. None of the above.

---

**Question 2**

Which of the following modifiers can not be applied to a method?

a. Final
b. Static
c. synchronized
d. transient
e. native
f. None of the above.

---

**Question 3**

Which of the following modifiers can not be used with the abstract modifier in a method declaration?

a. final
b. private
c. protected
d. public
e. Static
f. synchronized
g. native

---

**Question 4**

Which of the following statements is true?

a. An abstract method can not be overridden by an abstract method.

b. An instance method that is not abstract can not be overridden by an abstract method.
c. An abstract method declaration can not include a throws clause.
d. The body of an abstract method is represented by a set of empty brackets.
e. None of the above.

---

**Question 5**

Which of the following statements is not true?

a. A static method is also known as a class method.
b. A class method is not associated with a particular instance of the class.
c. The keyword, this, can not be used inside the body of a static method.
d. The keyword, super, may be used in the body of a static method.
e. A method that is not static is known as an instance method.
f. None of the above.

---

**Question 6**

Which of the following statements are true?

a. A final method can not be overridden.
b. All methods declared in a final class are implicitly final.
c. The methods declared in a final class must be explicitly declared final or a compile-time error occurs.
d. It is a compile-time error if a private method is declared final.
e. A machine-code generator can inline the body of a final method.

---

**Question 7**

Which of the following statements are true?

a. If an accessible superclass method is static, then any method with the same signature in a subclass must also be static.
b. If a superclass method is synchronized, then the overriding method must also be synchronized.
c. If a superclass method is public, then the overriding method must also be public.
d. If a superclass method is native, then the overriding method must also be native.
e. If a superclass method is protected, then the overriding method must be protected or public.

1e
volatile
An abstract method declaration provides no method body. If one abstract method is appears within a class declaration, then the entire class must be declared abstract and the class can not be instantiated. The access modifiers, private, protected and public, can be applied to a method. The field modifiers, transient and volatile, are not applicable to method declarations.

2d
transient
A final method can not be overridden. A static method is associated with a class, but not a particular instance of the class. A thread can not enter a synchronized method without first acquiring a lock. The field modifiers, transient and volatile, are not applicable to method declarations. A native method is implemented in platform-dependent code.

3a b e f g
final private static synchronized native
A final or private method can not be overridden, and can not be abstract. An abstract method declaration provides no implementation of the method, and all implementation details are left to the overriding method in the subclass. The synchronized modifier specifies an implementation detail that can be omitted from the declaration of an overriding method of a subclass, so it makes no sense to allow the use of the synchronized modifier in an abstract method declaration.

4e
None of the above.
An abstract method of a subclass can override by an abstract method of a superclass. The overriding abstract method declaration can be a good place to add comments. An abstract method of a subclass can override a concrete implementation of a method of a superclass. An abstract method declaration can have a throws clause. The body of an abstract method is a semicolon.

5d
The keyword, super, may be used in the body of a static method.
The keyword, this, refers to the instance on which the method has been invoked. A static method--also known as a *class method*-- is not invoked on a particular instance of an object, but is instead invoked on the class. Since a static method is not associated with a particular instance, an attempt to use the keyword, this, within the body of a static method results in a Compile-time error. Similarly, the keyword, super, can not be used within the body of a static method.

6a b e
A final method can not be overridden. All methods declared in a final class are implicitly final. A machine-code generator can inline the body of a final method. All methods declared in a final class are implicitly final. It is permissible--but not required--that all such methods be explicitly declared final. All private methods are implicitly final. It is permissible--but not required--that all private methods be explicitly declared final. The body of an inline method is inserted directly into the code at the point where the method is invoked. If the method is invoked at 10 different points in the code, then the body can be copied to all 10 points. Inline code runs very quickly, because it removes the need to do the work associated with a method invocation. If the method is executed repeatedly in a loop, then inlining can improve performance significantly. The machine-code generator has the option to inline a final method.

7a c e
If an accessible superclass method is static, then any method with the same signature in a subclass must also be static. If a superclass method is public, then the overriding method must also be public. If a superclass method is protected, then the overriding method must be protected or public.
The signature of a method is the name of the method and the number and types of the method parameters. The return type is not part of the method signature. An instance method declared in a subclass overrides an accessible superclass method with the same signature. A static method of a subclass hides--but does not override--an accessible superclass method with the same signature. A compile-time error is generated if a subclass contains a declaration of an instance method that shares the same signature as an accessible static method of the superclass. The modifiers, synchronized, native and strictfp, specify implementation details that the programmer is free to change in a subclass. Similarly, a subclass can override a concrete implementation of a method with an abstract method declaration. A subclass method may not have weaker access than the overridden superclass method. For example, a public method may not be overridden by a private method. However, a subclass method can provide greater access than a superclass method. For example, a protected method can be overridden by a public method.

**Return Types**

**Question 1**

```
class JSC201 {
  static byte m1() {
    final char c1 = '\u0001';
    return c1;                          // 1
  }
  static byte m2(final char c2) {return c2;}  // 2
  public static void main(String[] args) {
    char c3 = '\u0003';
    System.out.print(""+m1()+m2(c3));        // 3
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 13
b. Prints: 4
c. Compile-time error at 1
d. Compile-time error at 2
e. Run-time error
f. None of the above

---

**Question 2**

```
class JSC202 {
  static byte m1() {final short s1 = 2; return s1;}  // 1
  static byte m2(final short s2) {return s2;}        // 2
  public static void main(String[] args) {
    short s3 = 4;
    System.out.print(""+m1()+m2(s3));                // 3
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 24
b. Prints: 6
c. Compile-time error at 1.
d. Compile-time error at 2.
e. Run-time error
f. None of the above

---

**Question 3**

```
class JSC203 {
  static int m1(byte b) {return b;}   // 1
  static int m2(char c) {return c;}   // 2
  static int m3(long l) {return l;}   // 3
```

```
  public static void main(String[] args) {
    byte b = 1; char c = '\u0002'; long l = 4L;
    System.out.print(""+m1(b)+m2(c)+m3(l));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 124
b. Prints: 7
c. Compile-time error at 1.
d. Compile-time error at 2.
e. Compile-time error at 3.
f. Run-time error

---

**Question 4**

```
class JSC204 {
  static int m1(short s)  {return s;}   // 1
  static int m2(float f) {return f;}    // 2
  public static void main(String[] args) {
    short s = 3; float f = 5.0f;
    System.out.print(""+m1(s)+m2(f));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 35.0
b. Prints: 8.0
c. Compile-time error at 1.
d. Compile-time error at 2.
e. Run-time error
f. None of the above

---

**Question 5**

```
class JSC205 {
  static int m1(int i) {return i;}     // 1
  static void m2(int i) {return i;}    // 2
  static int m3(int i) {return;}       // 3
  public static void main(String[] args) {
    System.out.print(""+m1(1)+m2(2)+m3(3)); // 4
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 123

b. Prints: 6

c. Compile-time error at 1.

d. Compile-time error at 2.

e. Compile-time error at 3.

f. Compile-time error at 4.

1d

Compile-time error at 2

There is a compile-time error at 2. The char type variable c2 is not a compile-time constant, so it can not be assigned to type byte without an explicit cast. The method parameter c2 is declared final, so the value of c2 can not be changed within method m2. The value of method parameter c2 is set at run time to the value of the argument that is provided when m2 is invoked at line 3. For that reason, the method parameter c2 is not a compile-time constant. In method m2, the statement, "return c2;", is a return statement with an expression, c2. A compile-time error occurs if the type of the expression is not assignable to the declared result type of the method. The declared result type of method m2 is byte. The return statement attempts to return the value of the char type variable c2. If a char value is a compile-time constant, and if the value falls within the range of type byte, then the char value is assignable to type byte. In method m2, variable c2 is not a compile-time constant, because the value of c2 is not known at compile time. Instead, the value of c2 is assigned at run time to the value of the argument. Since the char type variable c2 is not a compile-time constant, the value of variable c2 is not assignable to the return type of method m2 without an explicit cast. While the declaration of method m2 produces a compile-time error, the declaration of method m1 does not. The local variable c1 is declared final and the value is set at compile time; so c1 is a compile-time constant. The value \u0001 falls within the range of type byte; so the value of the compile-time constant c1 is assignable to the return type of method m1 without an explicit cast.

2d

Compile-time error at 2.

There is a compile-time error at 2. The short type variable s2 is not a compile-time constant, so it can not be assigned to type byte without an explicit cast. The method parameter s2 is declared final, so the value of s2 can not be changed within method m2. The value of method parameter s2 is set at run time to the value of the argument that is provided when m2 is invoked at line 3. For that reason, the method parameter s2 is not a compile-time constant. In method m2, the statement,

"return s2;", is a return statement with an expression, s2. A compile-time error occurs if the type of the expression is not assignable to the declared result type of the method. The declared result type of method m2 is byte. The return statement attempts to return the value of the short type variable s2. If a short value is a compile-time constant, and if the value falls within the range of type byte, then the short value is assignable to type byte without an explicit cast. In method m2, variable s2 is not a compile-time constant, because the value of s2 is not known at compile time. Instead, the value of s2 is assigned at run time to the value of the argument. Since the short type variable s2 is not a compile-time constant, the value of variable s2 is not assignable to the return type of method m2 without an explicit cast. While the declaration of method m2 produces a compile-time error, the declaration of method m1 does not. The local variable s1 is declared final and the value is set at compile time; so s1 is a compile-time constant. The value 2 falls within the range of type byte; so the value of the compile-time constant s1 is assignable to the return type of method m1 without an explicit cast.

3e

Compile-time error at 3.

There is a compile-time error at line 3. The long type variable, l, can not be assigned to type int without an explicit cast. The statement, "return l;", is a return statement with an expression, l. A compile-time error occurs if the type of the expression is not assignable to the declared result type of the method. The declared result type of the method, m3, is int. The type of the variable, l, is long, so an explicit cast is needed to perform the narrowing primitive conversion, "return (int)l;". The declarations of methods m1 and m2 do not generate compile-time errors, because the types of the expressions contained in the return statements are assignable to type int. Widening conversions from types byte, char, or short to type int do not require an explicit cast.

4d

Compile-time error at 2.

There is a compile-time error at 2, because a narrowing primitive conversion from type float to type int requires an explicit cast. There is no compile-time error at 1, because widening primitive conversions from types byte, char, or short to type int do not require an explicit cast.

5d e f

Compile-time error at 2. Compile-time error at 3. Compile-time error at 4.

At line 2, the statement, "return i;", contains the expression, i. The enclosing method, m2, is declared void. The return statement generates a compile-time error, because it contains an expression. At line 3, the statement, "return;", does not contain an expression. The enclosing method, m3, is declared with the result type, int. The return statement generates a compile-time error, because it does not contain an expression that produces a value that is assignable to the declared result type

.

**Control**

**Question 1**

```
class JMM102 {
  public static void main(String args[]) {
    for (int i = 0; i<5 ;i++) {
      switch(i) {
        case 0: System.out.print("v ");break;
        case 1: System.out.print("w ");
        case 2: System.out.print("x ");break;
```

```
      case 3: System.out.print("y ");
      case 4: System.out.print("z ");break;
      default: System.out.print("d ");
}}}}
```

What is the result of attempting to compile and run the program?

a. Prints: v w x y z
b. Prints: v w x y z d
c. Prints: v w x x y z z
d. Prints: v w w x y y z d
e. Prints: d d d d d d
f. Run-time error
g. Compile-time error
h. None of the above

---

**Question 2**

```
class JMM103 {
  public static void main(String args[]) {
    for (int i = 0; i < 5 ;i++) {
      switch(i) {
        0: System.out.print("v ");break;
        1: System.out.print("w ");
        2: System.out.print("x ");break;
        3: System.out.print("y ");
        4: System.out.print("z ");break;
}}}}
```

What is the result of attempting to compile and run the program?

a. Prints: v w x y z
b. Prints: v w x x y z z

c. Prints: v w w x y y z

d. Run-time error.

e. Compile-time error.

f. None of the above.

---

**Question 3**

```
class JMM107 {
  public static void main(String[] args) {
    boolean b = true;
    if (b = false) {System.out.print("A");
    } else if (b) {System.out.print("B");
    } else {System.out.print("C");}
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A

b. Prints: B

c. Prints: C

d. Run-time error

e. Compile-time error

f. None of the above

---

**Question 4**

```
class JMM108 {
  static boolean b;
  public static void main(String[] args) {
    if (b) {System.out.print("A");
    } else if (b = false) {System.out.print("B");
    } else if (b) {System.out.print("C");
    } else if (!b) {System.out.print("D");
```

```
   } else {System.out.print("E");}
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A
b. Prints: B
c. Prints: C
d. Prints: D
e. Prints: E
f. Run-time error
g. Compile-time error
h. None of the above

---

**Question 5**

```
class JMM109 {
  public static void main(String[] args) {
    boolean b;
    if (b = false) {System.out.print("A");
    } else if (b) {System.out.print("B");
    } else if (!b) {System.out.print("C");
    } else {System.out.print("D");}
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A
b. Prints: B
c. Prints: C
d. Prints: D
e. Run-time error
f. Compile-time error

g. None of the above

---

**Question 6**

```
class JMM122 {
  public static void main (String[] args) {
    for (int i = 0; i < 4; i++) {
      switch (i) {
        case 0: System.out.print("A");
        case 1: System.out.print("B");
        case 2: System.out.print("C");
}}}}
```

What is the result of attempting to compile and run the program?

a. Prints: ABC
b. Prints: ABCC
c. Prints: CBA
d. Prints: ABCBCC
e. Run-time error
f. Compile-time error
g. None of the above

---

**Question 7**

```
class JMM123 {
  public static void main (String args[]) {
    int i = 0, j = 0, k = 0;
label1:
    for (int h = 0; h < 6; h++) {
label2:
      do { i++; k = h + i + j;
        switch (k) {
```

```
        default: break label1;
        case 1: continue label2;
        case 2: break;
        case 3: break label2;
        case 4: continue label2;
        case 5: continue label1;
      }
    } while (++j<5);
  }
  System.out.println(h + "," + i + "," + j);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,1,0
b. Prints: 0,2,1
c. Prints: 1,3,1
d. Prints: 2,4,1
e. Run-time error
f. Compile-time error
g. None of the above

---

**Question 8**

```
class JMM103 {
  public static void main(String args[]) {
    byte b = -1;
    switch(b) {
      case 0:    System.out.print("zero "); break;
      case 100:  System.out.print("100 ");  break;
      case 1000: System.out.print("1000 "); break;
      default: System.out.print("Default ");
}}}
```

What is the result of attempting to compile and run the program?

a. Prints: zero
b. Prints: 100
c. Prints: 1000
d. Prints: Default
e. Run-time error
f. Compile-time error
g. None of the above

---

**Question 9**

```
class JMM104 {
  public static void main (String args[]) {
    char c = 'b';
    switch(c) {
      case 'a': System.out.print("1");
      case 'b': System.out.print("2");
      case 'c': System.out.print("3");
      default:  System.out.print("4");
}}}
```

What is the result of attempting to compile and run the program?

a. Prints: 3
b. Prints: 34
c. Prints: 234
d. Prints: 1234
e. Run-time error
f. Compile-time error
g. None of the above

---

**Question 10**

```
class JMM105 {
  public static void main(String args[]) {
    int x = 6; int success = 0;
    do  {
      switch(x) {
        case 0: System.out.print("0"); x += 5; break;
        case 1: System.out.print("1"); x += 3; break;
        case 2: System.out.print("2"); x += 1; break;
        case 3: System.out.print("3"); success++; break;
        case 4: System.out.print("4"); x -= 1; break;
        case 5: System.out.print("5"); x -= 4; break;
        case 6: System.out.print("6"); x -= 5; break;
      }
    } while ((x != 3) || (success < 2));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 60514233
b. Prints: 6152433
c. Prints: 61433
d. Prints: 6143
e. Run-time error
f. Compile-time error

---

**Question 11**

```
class JMM106 {
  public static void main(String args[]) {
    int x = -5; int success = 0;
    do  {
      switch(x) {
        case 0: System.out.print("0"); x += 5; break;
```

```
      case 1: System.out.print("1"); x += 3; break;
      case 2: System.out.print("2"); x += 1; break;
      case 3: System.out.print("3"); success++; break;
      case 4: System.out.print("4"); x -= 1; break;
      case 5: System.out.print("5"); x -= 4; break;
      case 6: System.out.print("6"); x -= 5; break;
      default: x += x < 0 ? 2 : -2;
    }
  } while ((x != 3) || (success < 2));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 60514233
b. Prints: 1433
c. Prints: 61433
d. Prints: 051433
e. Run-time error
f. Compile-time error

---

**Question 12**

```
class JMM124 {
  public static void main(String args[]) {
    int k;
    for (int i=0, j=0; i<2; i++,j++) {System.out.print(i);}     // 1
    for (int i=0, k=0; i<2; i++,k++) {System.out.print(i);}     // 2
    for (int i=0, int j=0; i<2; i++,j++) {System.out.print(i);} // 3
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 012345
b. Prints: 010101

c. Compile-time error at line 1

d. Compile-time error at line 2

e. Compile-time error at line 3

f. Run-time error

---

**Question 13**

```
class JMM125 {
  static int i;
  public static void main(String args[]) {
    for (i=1; i<3; i++) {System.out.print(i);}     // 1
    for (int i=1; i<3; i++) {System.out.print(i);} // 2
    int i;                                         // 3
    for (i=0; i<2; i++) {System.out.print(i);}     // 4
    System.out.print(JMM125.i);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 1212010

b. Prints: 1212013

c. Compile-time error at line 1

d. Compile-time error at line 2

e. Compile-time error at line 4

f. Run-time error

g. None of the above

---

**Question 14**

```
class JMM126 {
  static int i;
  public static void main(String args[]) {
    for (i=1; i<3; i++)     {System.out.print(i);} // 1
```

```
    for (int i=1; i<3; i++) {System.out.print(i);} // 2
    int i;                                         // 3
    for (int i=0; i<2; i++) {System.out.print(i);} // 4
    System.out.print(JMM126.i);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 1212010
b. Prints: 1212013
c. Compile-time error at line 1
d. Compile-time error at line 2
e. Compile-time error at line 4
f. Run-time error
g. None of the above

---

**Question 15**

```
class JMM101 {
  public static void main(String[] args) {
    int i = 0;
    while (i++ < args.length) {
      System.out.print(args[i]);
}}}
java JMM101 A B C D E F
```

What is the result of attempting to compile and run the program using the specified command line?

a. Prints: JMM101ABCDEF
b. Prints: ABCDEF
c. Compile-time error
d. Run-time error

e. None of the above

---

**Question 16**

```
class JMM110 {
  public static void main (String[] args) {
    int j = 0;
    do for (int i = 0; i++ < 2;)
      System.out.print(i);
    while (j++ < 2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0001
b. Prints: 012
c. Prints: 012012
d. Prints: 012345
e. Prints: 001122
f. Prints: 1112
g. Prints: 111222
h. Prints: 121212
i. Run-time error
j. Compile-time error
k. None of the above

---

**Question 17**

```
class JMM111 {
  public static void main (String[] args) {
    int j = 0;
    for (int i = 0; i < 2; i++) do
```

```
    System.out.print(i);
  while (j++ < 2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0001
b. Prints: 012
c. Prints: 012012
d. Prints: 012345
e. Prints: 001122
f. Prints: 1112
g. Prints: 111222
h. Prints: 121212
i. Run-time error
j. Compile-time error
k. None of the above

---

**Question 18**

```
class JMM112 {
  public static void main (String[] args) {
    int j = 0;
    for (int i = 0; i++ < 2;) do
      System.out.print(i);
    while (j++ < 2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0001
b. Prints: 012
c. Prints: 012012

d. Prints: 012345
e. Prints: 001122
f. Prints: 1112
g. Prints: 111222
h. Prints: 121212
i. Run-time error
j. Compile-time error
k. None of the above

---

**Question 19**

```
class JMM113 {
  public static void main (String[] args) {
    int i = 0, j = 0, k = 0;
    do while (i++ < 3)
      System.out.print(k++);
    while (j++ < 3);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0001
b. Prints: 012
c. Prints: 012012
d. Prints: 012345
e. Prints: 001122
f. Prints: 1112
g. Prints: 111222
h. Prints: 121212
i. Run-time error
j. Compile-time error
k. None of the above

---

**Question 20**

```
class JMM114 {
  public static void main (String[] args) {
    int i = 0, j = 0;
    while (i++ < 3) do
      System.out.print(j);
    while (j++ < 3);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0001
b. Prints: 001122
c. Prints: 012012
d. Prints: 012345
e. Prints: 1112
f. Prints: 111222
g. Prints: 121212
h. Run-time error
i. Compile-time error
j. None of the above

---

**Question 21**

```
class JMM115 {
  static int m1(String s, int i) {
    System.out.print(s + i);
    return i;
  }
  public static void main (String[] args) {
    int i = 0, j = 0, k = 0;
```

```
   do while (m1("i", ++i) < 2)
     System.out.print("k" + ++k);
   while (m1("j", ++j) < 2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: i1k1i2k2j1i3j2
b. Prints: i1k1i2k2j1i1k1i2k2j2
c. Prints: i1k1i2j1i3j2
d. Run-time error
e. Compile-time error
f. None of the above

---

**Question 22**

```
class JMM116 {
  static int m1(String s, int i) {
    System.out.print(s + i);
    return i;
  }
  public static void main (String[] args) {
    int j = 0;
    for (int i = m1("A",0); m1("B",i) < 2; m1("C",++i)) {
      m1("J",++j);
    }
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A0B0C1J1B1C2J2B2
b. Prints: A0B0J1C1B1J2C2B2
c. Prints: A0B0J1C1A1B1J2C2A2B2
d. Run-time error

e. Compile-time error
f. None of the above

---

**Question 23**

```
class JMM117 {
  public static void main (String[] args) {
    int i = 0, j = 9;
    do {
      i++;
      if (j-- < i++) {break;}
    } while (i < 5);
    System.out.print(i + "," + j);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 5,4
b. Prints: 6,3
c. Prints: 6,6
d. Prints: 7,2
e. Run-time error
f. Compile-time error
g. None of the above

---

**Question 24**

```
class JMM118 {
  public static void main (String[] args) {
    int i = 0, j = 9;
    while (i++ <= j--) {i++; if (j < 5) break;}
    System.out.print(i + "," + j);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 4,7
b. Prints: 6,6
c. Prints: 7,2
d. Prints: 8,5
e. Prints: 9,4
f. Run-time error
g. Compile-time error
h. None of the above

---

**Question 25**

```
class JMM119 {
  public static void main (String[] args) {
    int i = 0, j = 9;
    do {
      if (j < 4) {break;} else if (j-- < 7) {continue;}
      i++;
    } while (i++ < 7);
    System.out.print(i + "," + j);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 4,7
b. Prints: 6,6
c. Prints: 6,5
d. Prints: 6,4
e. Prints: 7,5
f. Prints: 8,4
g. Run-time error

h. Compile-time error

i. None of the above

---

**Question 26**

```
class JMM120 {
  public static void main (String args[]) {
    int i = 0, j = 0, k = 0;
label1:
    for (;;) { i++;
label2:
      do {
        k = i + j;
        switch (k) {
          case 0: continue label2;
          case 1: continue label1;
          case 2: break;
          case 3: break label2;
          case 4: break label1;
          default: break label1;
        }
      } while (++j<5);
    }
    System.out.println(i + "," + j);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 2,1

b. Prints: 2,2

c. Prints: 3,1

d. Prints: 3,2

e. Prints: 3,3

f. Run-time error

g. Compile-time error

h. None of the above

---

**Question 27**

```
class JMM121 {
  public static void main (String args[]) {
    int h = 0, i = 0, j = 0, k = 0;
label1:
    for (;;) { h++;
label2:
      do { i++; k = h + i + j;
        switch (k) {
          default: break label1;
          case 1: continue label1;
          case 2: break;
          case 3: break label2;
          case 4: continue label2;
          case 5: continue label1;
        }
      } while (++j < 5);
    }
    System.out.println(h + "," + i + "," + j);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 1,2,3

b. Prints: 1,3,2

c. Prints: 2,2,2

d. Prints: 2,4,1

e. Prints: 2,4,2

f. Run-time error

g. Compile-time error

h. None of the above

1.c .Prints: v w x x y z z .Cases one and three have no break statement, so the next case is also executed and x and z are printed twice.

2.e .Compile-time error. .The keyword, case, is missing from the case labels.

3.c .Prints: C .The boolean type variable, b, is initialized to true. The boolean expression of the first if statement, b = false, is a simple assignment expression that sets b to false. Always look carefully at the boolean expression of an if statement to verify that the expected equality operator (==) has not been replaced by the simple assignment operator (=).

4.d .Prints: D .The expression, b = false, appears to be testing the value of b, but it is really setting the value of b. Always look carefully at the boolean expression of an if statement to verify that the expected equality operator (==) has not been replaced by the simple assignment operator (=).

5.c .Prints: C .The first if statement initializes the value of b. The expression, b = false, appears to be testing the value of b, but it is really setting the value of b. Always look carefully at the boolean expression of an if statement to verify that the expected equality operator (==) has not been replaced by the simple assignment operator (=).

6.d .Prints: ABCBCC .The switch statement does not contain any break statements. The first case falls through to the second. The second case falls through to the third. There is no default switch label, so nothing is printed when variable i is 3.

7.f .Compile-time error .The loop variable, h, is local to the for statement. The print statement causes a compile-time error, because it refers to the out-of-scope local variable, h. Always check for variables that are out-of-scope!

8.f .Compile-time error .The case constant expression, 1000, produces a compile-time error, because it exceeds the range of the switch expression type, byte. The type of a switch expression can be byte, short, int or char. A constant expression is associated with each case label. Each case constant expression must be assignable to the type of the switch expression. In this case, the switch expression, b, is of type byte; so the maximum positive value of each case constant expression is limited to the maximum byte value, 127.

9.c .Prints: 234 .No break statements appear inside the switch statement, so more than one case is processed.

10.c .Prints: 61433 .On the first pass through the loop, the value of x is 6, so 5 is subtracted from x. On the second pass, the value of x is 1, so 3 is added to x. On the third pass, the value of x is 4, so 1 is subtracted from x. On the fourth pass, the value of x is 3, so the variable, success, is incremented from zero to one. On the final pass, the value of x is 3 and the variable, success, is incremented to the value, 2. The boolean expression of the do loop is now false, so control passes out of the loop.

11.b .Prints: 1433 .On the first pass through the loop, the switch expression, x, has the value, -5. None of the case constants are matched, so the statement following the default label is executed causing the value of x to be set to -3. On the second pass, the default case of the switch statement is executed again, and two is again added to the value of x. The new value is -1. On the third pass, the value of x is again incremented, and the new value is +1. After that, the value of x is printed on each pass through the loop. For the last two passes, the value of x is 3.

12.d e .Compile-time error at line 2 Compile-time error at line 3 .A compile-time error occurs at line 2 as a result of the attempt to shadow the method local variable, k, within the for-loop. A variable declared within the for-loop can not shadow a local variable of the enclosing method. At line 3, the declaration of variable j causes a compile-time error. The initialization part of a for statement may use a single local variable declaration statement to declare more than one local variable. Each of the new variables is declared in a comma separated list of variable declarators. In this program, the local variables should have been declared as follows: "int i=0, j=0;". Instead, the type of variable j has been incorrectly added to the declarator: "int i=0, int j=0;". The result is a compile-time error.

13.b .Prints: 1212013 .A method local variable, i, is declared at line 3. At line 4, the initialization part of the for statement initializes the method local variable to the value zero. The for statement does not declare a new variable, i, so the method local variable is not shadowed within the loop. Suppose a local variable, v, is declared within a statement or block. Variable, v, can shadow a class variable or an instance variable of the same name, but a compile-time error is generated if v

shadows a method local variable. Please note that a compile-time error is not generated if a local variable is shadowed within the declaration of a class that is nested within the scope of the local variable.

14.e .Compile-time error at line 4 .A method local variable, i, is declared at line 3. At line 4, the initialization part of the for statement attempts to shadow the method local variable with a new variable that is intended to be local to the for statement. The result is a compile-time error. At line 2, a new variable, i, is declared within the for statement. That variable shadows the class variable, but it does not shadow the method local variable declared at line 3. The scope of a method local variable begins with its own initializer--if present--and extends to the end of the method body. At line 2, the method local variable is not in scope, so shadowing does not occur. Suppose a local variable, v, is declared within a statement or block. Variable, v, can shadow a class variable or an instance variable of the same name, but a compile-time error is generated if v shadows a method local variable. Please note that a compile-time error is not generated if a local variable is shadowed within the declaration of a class that is nested within the scope of the local variable.

15.d .Run-time error .The index, i, is incremented before the array access expression is evaluated, so the first element accessed is at index one instead of zero. The arguments, BCDEF, are printed before an ArrayIndexOutOfBoundsException is thrown when the attempt is made to access an element beyond the end of the array.

16.h .Prints: 121212 .This trick question has a for loop nested inside of a do loop. For each iteration, the values of i and j are as follows: (1,0)(2,0)(1,1)(2,1)(1,2)(2,2).

17.a .Prints: 0001 .This trick question has a do-loop nested inside of a for-loop. For each iteration, the values of i and j are as follows: (0,0)(0,1)(0,2)(1,3).

18.f .Prints: 1112 .This trick question has a do-loop nested inside of a for-loop. For each iteration, the values of i and j are as follows: (1,0)(1,1)(1,2)(2,3).

19.b .Prints: 012 .This trick question has a while-loop nested inside of a do-loop. For each iteration, the values of i, j and k are as follows: (1,0,0)(2,0,1)(3,0,2).

20.d .Prints: 012345 .This trick question has a do-loop nested inside of a while-loop. For each iteration, the values of i and j are as follows: (1,0)(1,1)(1,2)(1,3)(2,4)(3,5).

21.c .Prints: i1k1i2j1i3j2 .A while loop is nested inside of a do loop. The while loop iterates once during the first iteration of the do loop. The body of the while loop does not execute during the final iteration of the do loop.

22.b .Prints: A0B0J1C1B1J2C2B2 .The initialization statement, labeled A, is processed first. The boolean expression, labeled B, is processed before each iteration of the loop. The body of the loop is processed next followed by the update expression, labeled C.

23.c .Prints: 6,6 .Suppose the print statement, System.out.print("(" + i + "," + j + ")");, is inserted at the top of the loop. The output of the program would be as follows: (0,9)(2,8)(4,7)6,6. The variable, i, is incremented twice with each pass through the loop. The variable, j, is decremented once with each pass. The loop terminates when i reaches six.

24.e .Prints: 9,4 .Suppose the print statement, System.out.print("(" + i + "," + j + ")");, is inserted at the top of the loop. The output of the program would be as follows: (1,8)(3,7)(5,6)(7,5)9,4. The variable, i, is incremented twice with each pass through the loop. The variable, j, is decremented once with each pass. The boolean expression of the while loop, i++ <= j--, is false when i reaches 8 and j reaches 5. The value of i is subsequently incremented and j is decremented yielding 9 and 4 respectively. Those values are printed.

25.f .Prints: 8,4 .Suppose the print statement, System.out.print("(" + i + "," + j + ")");, is inserted at the top of the loop. The output of the program would be as follows: (0,9)(2,8)(4,7)(6,6)(7,5)8,4. The initial value of j is 9. With each pass through the loop, the expression, j-- < 7, decrements j. The initial value of variable i is 0. With each pass through the loop, the expression, i++ < 7, increments variable i. Inside the body of the loop, i is also incremented as long as j is greater than or equal to 7. When j is less than seven, variable i is no longer incremented inside the body of the loop, but it is still incremented by the expression, i++ < 7.

26.c .Prints: 3,1 .Suppose the print statement, System.out.print("("+i+","+j+","+k+")");, is inserted before the switch statement. The output of the program would be as follows: (1,0,1)(2,0,2)(2,1,3)(3,1,4)3,1. On the first iteration, case 1 is processed. The "continue label1;" statement causes control to go directly to the top of the for-loop following the label, label1. The boolean expression of the do-loop is not processed. On the second iteration, case 2 is processed. The break statement

causes the switch statement to complete, and control passes to the boolean expression of the do-loop. On the third iteration, case 3 is processed. The break with label statement, "break label2;", completes abruptly; and the do-loop completes abruptly without processing the loop expression, ++j<5, so j is not incremented. On the fourth iteration, case 4 is processed. The break with label statement, "break label1;", completes abruptly, and the for-loop completes abruptly.
27.b .Prints: 1,3,2 .Suppose the print statement, System.out.print("("+h+","+i+","+j+","+k+")");, is inserted before the switch statement. The output of the program would be as follows: (1,1,0,2)(1,2,1,4)(1,3,2,6)1,3,2. Three iterations of the loop are executed. On the first iteration, the switch expression, k, matches the case constant, 2. The subsequent break statement is executed and the switch statement completes normally. On the second iteration, the switch expression matches the case constant, 4; and the subsequent continue statement causes control to pass to the loop-continuation point of the do statement where the expression, ++j < 5, is evaluated and found to be true. On the final iteration, the switch expression does not match any case constant; so the break statement following the default label is processed.

**Exception Handling**

**Question 1**

```
class A {
  public static void main (String[] args) {
    Error error = new Error();
    Exception exception = new Exception();
    System.out.print((exception instanceof Throwable) + ",");
    System.out.print(error instanceof Throwable);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false
d. Prints: true,true
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 2**

```
class A {A() throws Exception {}} // 1
class B extends A {B() throws Exception {}} // 2
class C extends A {C() {}} // 3
```

Which of the following statements are true?

a. class A extends Object.
b. Compile-time error at 1.
c. Compile-time error at 2.
d. Compile-time error at 3.

---

**Question 3**

```
class A {
  public static void main (String[] args) {
    Object error = new Error();
    Object runtimeException = new RuntimeException();
    System.out.print((error instanceof Exception) + ",");
    System.out.print(runtimeException instanceof Exception);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false
d. Prints: true,true
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 4**

```
class Level1Exception extends Exception {}
class Level2Exception extends Level1Exception {}
class Level3Exception extends Level2Exception {}
class Purple {
  public static void main(String args[]) {
    int a,b,c,d,f,g,x;
    a = b = c = d = f = g = 0;
    x = 1;
    try {
      try {
        switch (x) {
          case 1: throw new Level1Exception();
          case 2: throw new Level2Exception();
          case 3: throw new Level3Exception();
      } a++; }
      catch (Level2Exception e) {b++;}
      finally {c++;}
    }
    catch (Level1Exception e) { d++;}
    catch (Exception e) {f++;}
    finally {g++;}
    System.out.print(a+","+b+","+c+","+d+","+f+","+g);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0,0,1,0,0
b. Prints: 0,0,1,1,0,1
c. Prints: 0,1,1,1,0,1
d. Prints: 1,0,1,1,0,1
e. Prints: 1,1,1,1,0,1
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 5**

```
class Level1Exception extends Exception {}
class Level2Exception extends Level1Exception {}
class Level3Exception extends Level2Exception {}
class Purple {
  public static void main(String args[]) {
    int a,b,c,d,f,g,x;
    a = b = c = d = f = g = 0;
    x = 2;
    try {
      try {
        switch (x) {
          case 1: throw new Level1Exception();
          case 2: throw new Level2Exception();
          case 3: throw new Level3Exception();
      } a++; }
      catch (Level2Exception e) {b++;}
      finally {c++;}
    }
    catch (Level1Exception e) { d++;}
    catch (Exception e) {f++;}
    finally {g++;}
    System.out.print(a+","+b+","+c+","+d+","+f+","+g);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0,1,0,0,1
b. Prints: 0,1,0,0,0,0
c. Prints: 0,1,1,0,0,1
d. Prints: 0,1,0,0,0,1
e. Prints: 1,1,1,0,0,1
f. Compile-time error
g. Run-time error

h. None of the above

---

**Question 6**

```
class Level1Exception extends Exception {}
class Level2Exception extends Level1Exception {}
class Level3Exception extends Level2Exception {}
class Purple {
  public static void main(String args[]) {
    int a,b,c,d,f,g,x;
    a = b = c = d = f = g = 0;
    x = 3;
    try {
      try {
        switch (x) {
          case 1: throw new Level1Exception();
          case 2: throw new Level2Exception();
          case 3: throw new Level3Exception();
      } a++; }
      catch (Level2Exception e) {b++;}
      finally {c++;}
    }
    catch (Level1Exception e) { d++;}
    catch (Exception e) {f++;}
    finally {g++;}
    System.out.print(a+","+b+","+c+","+d+","+f+","+g);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 1,1,1,0,0,1
b. Prints: 0,1,1,0,0,1
c. Prints: 0,1,0,0,0,0
d. Prints: 0,1,0,0,0,1

e. Prints: 0,0,1,0,0,1
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 7**

```
class Level1Exception extends Exception {}
class Level2Exception extends Level1Exception {}
class Level3Exception extends Level2Exception {}
class Purple {
  public static void main(String args[]) {
    int a,b,c,d,f,g,x;
    a = b = c = d = f = g = 0;
    x = 4;
    try {
      try {
              switch (x) {
          case 1: throw new Level1Exception();
          case 2: throw new Level2Exception();
          case 3: throw new Level3Exception();
          case 4: throw new Exception();
      } a++; }
      catch (Level2Exception e) {b++;}
      finally{c++;}
    }
    catch (Level1Exception e) { d++;}
    catch (Exception e) {f++;}
    finally {g++;}
    System.out.print(a+","+b+","+c+","+d+","+f+","+g);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0,0,0,0,1

b. Prints: 0,0,0,0,1,0
c. Prints: 0,0,1,0,0,1
d. Prints: 0,0,1,0,1,1
e. Prints: 0,1,1,1,1,1
f. Prints: 1,1,1,1,1,1
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 8**

```
class Level1Exception extends Exception {}
class Level2Exception extends Level1Exception {}
class Level3Exception extends Level2Exception {}
class Purple {
  public static void main(String args[]) {
    int a,b,c,d,f,g,x;
    a = b = c = d = f = g = 0;
    x = 5;
    try {
      try {
              switch (x) {
          case 1: throw new Level1Exception();
          case 2: throw new Level2Exception();
          case 3: throw new Level3Exception();
          case 4: throw new Exception();
      } a++; }
      catch (Level2Exception e) {b++;}
      finally {c++;}
    }
    catch (Level1Exception e) { d++;}
    catch (Exception e) {f++;}
    finally {g++;}
    System.out.print(a+","+b+","+c+","+d+","+f+","+g);
```

}}

What is the result of attempting to compile and run the program?

a. Prints: 1,0,0,0,0,0
b. Prints: 1,0,1,0,0,1
c. Prints: 0,0,1,0,0,1
d. Prints: 1,1,1,1,1,1
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 9**

```
class ColorException extends Exception {}
class WhiteException extends ColorException {}
class White {
  void m1() throws ColorException {throw new WhiteException();}
  void m2() throws WhiteException {}
  public static void main (String[] args) {
    White white = new White();
    int a,b,d,f; a = b = d = f = 0;
    try {white.m1(); a++;} catch (ColorException e) {b++;}
    try {white.m2(); d++;} catch (WhiteException e) {f++;}
    System.out.print(a+","+b+","+d+","+f);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,1,0,0
b. Prints: 1,1,0,0
c. Prints: 0,1,1,0
d. Prints: 1,1,1,0

e. Prints: 1,1,1,1
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 10**

```
class ColorException extends Exception {}
class WhiteException extends ColorException {}
class White {
  void m1() throws ColorException {throw new ColorException();}
  void m2() throws WhiteException {throw new ColorException();}
  public static void main (String[] args) {
    White white = new White();
    int a,b,d,f; a = b = d = f = 0;
    try {white.m1(); a++;} catch (ColorException e) {b++;}
    try {white.m2(); d++;} catch (WhiteException e) {f++;}
    System.out.print(a+","+b+","+d+","+f);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,1,0,0
b. Prints: 1,1,0,1
c. Prints: 0,1,0,1
d. Prints: 0,1,1,1
e. Prints: 1,1,1,1
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 11**

```
class ColorException extends Exception {}
class WhiteException extends ColorException {}
class White {
  void m1() throws ColorException {throw new ColorException();}
  void m2() throws WhiteException {throw new WhiteException();}
  public static void main (String[] args) {
    White white = new White();
    int a,b,d,f; a = b = d = f = 0;
    try {white.m1(); a++;} catch (WhiteException e) {b++;}
    try {white.m2(); d++;} catch (WhiteException e) {f++;}
    System.out.print(a+","+b+","+d+","+f);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,1,0,0
b. Prints: 1,1,0,1
c. Prints: 0,1,0,1
d. Prints: 0,1,1,1
e. Prints: 1,1,1,1
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 12**

```
class Level1Exception extends Exception {}
class Level2Exception extends Level1Exception {}
class Level3Exception extends Level2Exception {}
class Brown {
  public static void main(String args[]) {
    int a, b, c, d, f; a = b = c = d = f = 0;
    int x = 1;
    try {
```

```
     switch (x) {
       case 1: throw new Level1Exception();
       case 2: throw new Level2Exception();
       case 3: throw new Level3Exception();
   } a++; }
   catch (Level3Exception e) {b++;}
   catch (Level2Exception e) {c++;}
   catch (Level1Exception e) {d++;}
   finally {f++;}
   System.out.print(a+","+b+","+c+","+d+","+f);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0,0,1,1
b. Prints: 0,0,1,1,1
c. Prints: 0,1,1,1,1
d. Prints: 1,1,1,1,1
e. Prints: 0,0,1,0,1
f. Prints: 0,1,0,0,1
g. Prints: 1,0,0,0,1
h. Compile-time error
i. Run-time error
j. None of the above

---

**Question 13**

```
class Level1Exception extends Exception {}
class Level2Exception extends Level1Exception {}
class Level3Exception extends Level2Exception {}
class Brown {
  public static void main(String args[]) {
    int a, b, c, d, f; a = b = c = d = f = 0;
    int x = 2;
```

```
   try {
    switch (x) {
      case 1: throw new Level1Exception();
      case 2: throw new Level2Exception();
      case 3: throw new Level3Exception();
   } a++; }
   catch (Level3Exception e) {b++;}
   catch (Level2Exception e) {c++;}
   catch (Level1Exception e) {d++;}
   finally {f++;}
   System.out.print(a+","+b+","+c+","+d+","+f);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0,0,1,1
b. Prints: 0,0,1,1,1
c. Prints: 0,1,1,1,1
d. Prints: 1,1,1,1,1
e. Prints: 0,0,1,0,1
f. Prints: 0,1,0,0,1
g. Prints: 1,0,0,0,1
h. Compile-time error
i. Run-time error
j. None of the above

---

**Question 14**

```
class Level1Exception extends Exception {}
class Level2Exception extends Level1Exception {}
class Level3Exception extends Level2Exception {}
class Brown {
  public static void main(String args[]) {
    int a, b, c, d, f; a = b = c = d = f = 0;
```

```
   int x = 4;
   try {
     switch (x) {
       case 1: throw new Level1Exception();
       case 2: throw new Level2Exception();
       case 3: throw new Level3Exception();
   } a++; }
   catch (Level3Exception e) {b++;}
   catch (Level2Exception e) {c++;}
   catch (Level1Exception e) {d++;}
   finally {f++;}
   System.out.print(a+","+b+","+c+","+d+","+f);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0,0,1,1
b. Prints: 0,0,1,1,1
c. Prints: 0,1,1,1,1
d. Prints: 1,1,1,1,1
e. Prints: 0,0,1,0,1
f. Prints: 0,1,0,0,1
g. Prints: 1,0,0,0,1
h. Compile-time error
i. Run-time error
j. None of the above

---

**Question 15**

```
class ColorException extends Exception {}
class WhiteException extends ColorException {}
abstract class Color {
  abstract void m1() throws ColorException;
}
```

```
class White extends Color {
  void m1() throws WhiteException {throw new WhiteException();}
  public static void main (String[] args) {
    White white = new White();
    int a,b,c; a = b = c = 0;
    try {white.m1(); a++;}
      catch (WhiteException e) {b++;}
      finally {c++;}
    System.out.print(a+","+b+","+c);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0,0
b. Prints: 0,0,1
c. Prints: 0,1,0
d. Prints: 0,1,1
e. Prints: 1,0,0
f. Prints: 1,0,1
g. Prints: 1,1,0
h. Prints: 1,1,1
i. Compile-time error
j. Run-time error
k. None of the above

---

**Question 16**

```
class RedException extends Exception {}
class BlueException extends Exception {}
class White {
  void m1() throws RedException {throw new RedException();}
  public static void main (String[] args) {
    White white = new White();
    int a,b,c,d; a = b = c = d = 0;
```

```
    try {white.m1(); a++;}
      catch (RedException e) {b++;}
      catch (BlueException e) {c++;}
      finally {d++;}
    System.out.print(a+","+b+","+c+","+d);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,1,0,0
b. Prints: 1,1,0,1
c. Prints: 0,1,0,1
d. Prints: 0,1,1,1
e. Prints: 1,1,1,1
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 17**

```
class Level1Exception extends Exception {}
class Level2Exception extends Level1Exception {}
class Level3Exception extends Level2Exception {}
class Purple {
  public static void main(String args[]) {
    int a,b,c,d,f,g,x;
    a = b = c = d = f = g = 0;
    x = 1;
    try {
      throw new Level1Exception();
      try {
        switch (x) {
          case 1: throw new Level1Exception();
          case 2: throw new Level2Exception();
```

```
        case 3: throw new Level3Exception();
      } a++; }
      catch (Level2Exception e) {b++;}
      finally {c++;}
    }
    catch (Level1Exception e) { d++;}
    catch (Exception e) {f++;}
    finally {g++;}
    System.out.print(a+","+b+","+c+","+d+","+f+","+g);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 1,1,1,0,0,1

b. Prints: 0,1,1,0,0,1

c. Prints: 0,1,0,0,0,0

d. Prints: 0,1,0,0,0,1

e. Prints: 0,0,1,0,0,1

f. Compile-time error

g. Run-time error

h. None of the above

1.d .Prints: true,true .Both Error and Exception are subclasses of Throwable.

2.a d .class A extends Object. Compile-time error at 3. .The constructors for class B and class C both invoke the constructor for A. The constructor for class A declares Exception in the throws clause. Since the constructors for B and C invoke the constructor for A, it is necessary to declare Exception in the throws clauses of B and C. A compile-time error is generated at marker 3, because the constructor does not declare Exception in the throws clause.

3.b .Prints: false,true .Error is a direct subclass of Throwable. RuntimeException is a direct subclass of Exception.

4.b .Prints: 0,0,1,1,0,1 .The nested catch clause is able to catch a Level2Exception or any subclass of it. The switch statement throws a Level1Exception that can not be caught by the nested catch clause; so the nested finally block is executed as control passes to the first of the two outer catch clauses. The outer finally block is executed as control passes out of the try statement.

5.c .Prints: 0,1,1,0,0,1 .The nested catch block is able to catch a Level2Exception or any subclass of it causing b to be incremented. Both of the finally blocks are then executed.

6.b .Prints: 0,1,1,0,0,1 .The nested catch block is able to catch a Level2Exception or any subclass of it causing b to be incremented. Both of the finally blocks are then executed.

7.d .Prints: 0,0,1,0,1,1 .The nested catch clause is able to catch a Level2Exception or any subclass of it. The switch statement throws an Exception that can not be caught by the nested catch clause; so the nested finally block is executed as control passes to the second of the two outer catch clauses. The outer finally block is executed as control passes out of the try statement.

8.b .Prints: 1,0,1,0,0,1 .The switch statement does not throw an exception; so the switch completes normally. The subsequent statement increments the variable, a; and the try block completes normally. Both of the finally blocks are then executed.

9.c .Prints: 0,1,1,0 .The first try block contains two statements. The first invokes method m1, and the subsequent statement contains a post increment expression with the variable, a, as the operand. Method m1 throws a WhiteException exception, so variable a is not incremented as control passes to the catch block where b is incremented. The throws clause of m1 declares a ColorException, so the body may throw a ColorException or any subclass of ColorException. The second try block also contains two statements. The first invokes method m2, and the subsequent statement contains a post increment expression with the variable, d, as the operand. Method m2 does not throw an exception, so d is incremented, and the try block completes normally. Although the throws clause of m2 declares a WhiteException, there is no requirement to throw any exception.

10.f .Compile-time error .The throws clause of White.m2 declares a WhiteException, so the body of m2 may throw a WhiteException or any subclass of WhiteException. Instead, the body of m2 throws a superclass of WhiteException. The result is a compile-time error.

11.f .Compile-time error .The throws clause of White.m1 declares a ColorException, but the catch clause in the main method catches only a subclass of ColorException. The result is a compile-time error.

12.a .Prints: 0,0,0,1,1 .The first catch clause has a parameter e of type Level3Exception, so the first catch clause is able to catch any exception type that is assignable to type Level3Exception. Since Level2Exception is the superclass of Level3Exception, an instance of Level2Exception is not assignable to a catch clause parameter of type Level3Exception. Similarly, Level1Exception is also a superclass of Level3Exception, so an instance of Level1Exception is not assignable to a catch clause parameter of type Level3Exception. The only exception type that can be caught by the first catch clause is a Level3Exception. The second catch clause has a parameter e of type Level2Exception, so the second catch clause is able to catch a Level2Exception. The Level1Exception is the superclass of Level2Exception. An instance of Level1Exception is not assignable to a catch clause parameter of type Level2Exception, so the second catch clause can not catch a Level1Exception. Since a Level3Exception is a subclass of Level2Exception an exception of type Level3Exception is assignable to a catch clause parameter type Level2Exception. All exceptions of type Level3Exception will be caught by the first catch clause, so the second catch clause in this program will not have an opportunity to catch a Level3Exception. The third catch clause has a parameter e of type Level1Exception, so the third catch clause is able to catch a Level1Exception. The exceptions of type Level2Exception and Level3Exception are assignable to the catch clause parameter of the third catch clause, but the exceptions of those subclass types will be caught by the first two catch clauses. The switch statement throws a Level1Exception. The try block completes abruptly as control passes to the third catch block where d is incremented. The finally block is also executed, so f is incremented.

13.e .Prints: 0,0,1,0,1 .The first catch block is able to catch a Level3Exception or any subclass of Level3Exception. The second catch block is able to catch a Level2Exception or any subclass of Level2Exception. The switch statement throws a Level2Exception. The try block completes abruptly as control passes to the second catch block where c is incremented. The finally block is also executed, so f is incremented.

14.g .Prints: 1,0,0,0,1 .The switch statement does not throw an exception; so the switch completes normally. The subsequent statement increments the variable, a; and the try block completes normally. The finally block is also executed, so f is incremented.

15.d .Prints: 0,1,1 .The try block contains two statements. The first invokes method m1, and the subsequent statement contains a post increment expression with the variable, a, as the operand. Method m1 throws a WhiteException exception, so variable a is not incremented as control passes to the catch block where b is incremented. Although Color.m1 declares a ColorException in the throws clause, a subclass of Color is free to declare only a subclass of ColorException in the throws clause of the overriding method.

16.f .Compile-time error .A compile-time error is generated, because the second catch clause attempts to catch an exception that is never thrown in the try block.

17.f .Compile-time error .A throw statement is the first statement in the outer try block. A throw statement appearing in a try block causes control to pass out of the block. Consequently, statements can not be reached if they appear in the block after the throw statement. The switch statement that appears after the throw statement is unreachable and results in a compile-time error.

**Assertions**

**Question 1**

Which of the following are command-line switches used to enable assertions in non-system classes?

a. -ea
b. -ae
c. -aon
d. -aoff
e. -enableassertions
f. -assertionsenable
g. -assertionson
h. -assertionsoff

---

**Question 2**

Which of the following are command-line switches used to disable assertions in non-system classes?

a. -da
b. -ad
c. -aon
d. -aoff
e. -disableassertions
f. -assertionsdisable
g. -assertionson
h. -assertionsoff

---

**Question 3**

Which of the following are command-line switches used to disable assertions in non-system classes?

a. -disableassertions
b. -disableAssertions
c. -assertionsDisable
d. -assertionsOn
e. -assertionsOff
f. None of the above

---

**Question 4**

Which of the following are command-line switches used to enable assertions in system classes?

a. -saon
b. -saoff
c. -esa
d. -sae
e. -systemassertionson
f. -systemassertionsoff
g. -enablesystemassertions
h. -systemassertionsenable

---

**Question 5**

Which of the following statements are true?

a. By default assertions are disabled at run-time.

b. Assertions can be selectively enabled for any named class.
c. If assertions are selectively enabled for a named class then assertions are automatically enabled for any subclass of the named class.
d. Assertions can be selectively enabled for any named package.
e. If assertions are selectively enabled for any named package then assertions are automatically enabled for the subpackages.
f. Assertions can be selectively enable for the unnamed package in the current working directory.
g. Assertions can be selectively enable for any unnamed package in any named directory.

---

**Question 6**

Which of the following is thrown to indicate that an assertion has failed?

a. AssertError
b. AssertException
c. AssertionError
d. AssertionException
e. None of the above

---

**Question 7**

```
class A {
  void m1(int i) {
    int j = i % 3;
    switch (j) {
      case 0: System.out.print("0"); break;
      case 1: System.out.print("1"); break;
      default:
        assert j == 2;
        System.out.print(j);
  }}
  public static void main (String[] args) {
    A a = new A();
    for (int i=5; i >= -1; i--) {a.m1(i);}
}}
```

Which statements are true?

a. With assertions enabled it prints 210210-1 followed by an AssertionError message.
b. With assertions enabled it prints 210210 followed by an AssertionError message.
c. With assertions enabled it prints only 210210.
d. With assertions enabled it prints nothing.
e. With assertions disabled it prints 210210-1
f. With assertions disabled it prints only 210210
g. Assertions should not be used within the default case of a switch statement.

---

**Question 8**

```
class B {
  private static int x1;
  public void setX(int x) {x1 = x;}
  public int getX() {return x1;}
  public static void main(String[] args) {
    int i1 = 0, j1 = 0;
    do {
      System.out.print(j1++);
      assert (i1 = j1 + x1) < 6;
    } while (i1 < 3);
}}
```

Assuming that the setX method is never invoked, which statements are true?

a. With assertions enabled it prints 012345 followed by an AssertionError message.
b. With assertions enabled it prints 012.
c. With assertions disabled it prints: 012345
d. With assertions disabled it prints: 012
e. With assertions disabled it attempts to print an infinite sequence of numbers.
f. With assertions disabled the variable i1 is incremented with each pass through the loop.
g. As a rule, the boolean expression of an assert statement should not be used to perform actions that are required for normal operation of the program.

---

**Question 9**

Which statements are true?

a. Assertions should not be used to validate arguments passed to public methods.
b. Assertions should not be used to validate arguments passed to nonpublic methods.
c. Assertions should not be used within the default case of a switch statement.
d. Assertions should not be used to perform processing that is required for the normal operation of the application.

---

**Question 10**

Which statements are true?

a. Assertions should never be placed at any point in the code that should not be reached under normal circumstances.
b. The compiler will generate an error if an assert statement is "unreachable" as defined by the Java Language Specification.
c. A catch clause should not be used to catch an AssertionError.
d. AssertionError is a checked exception.

---

**Question 11**

```
class C {
  String m1(int i) {
    switch (i) {
      case 0: return "A";
      case 1: return "B";
      case 2: return "C";
      default: throw new AssertionError();
  }}
  public static void main(String[] args) {
    C c = new C();
    for (int i = 0; i < 4; i++) {
```

```
    System.out.print(c.m1(i));
}}}
```

Which statements are true?

a. With assertions enabled it prints ABC followed by an AssertionError message.
b. With assertions disabled it prints ABC followed by an AssertionError message.
c. Assertions should not be used within the default case of a switch statement.
d. In this code example an assert statement could not be used in place of the "throw" statement.

---

**Question 12**

```
class C {
  String m1(int i) {
    switch (i) {
      case 0: return "A";
      case 1: return "B";
      case 2: return "C";
      default:
        assert false;
    }
    return "E";
  }
  public static void main(String[] args) {
    C c = new C();
    for (int i = 0; i < 4; i++) {
      System.out.print(c.m1(i));
}}}
```

Which statements are true?

a. With assertions enabled it prints ABC followed by an AssertionError message.
b. With assertions enabled it prints ABCE followed by an AssertionError message.
c. With assertions disabled it prints ABC

d. With assertions disabled it prints ABCE

e. Assertions should not be used within the default case of a switch statement.

---

**Question 13**

```
class C {
  String m1(int i) {
    switch (i) {
      case 0: return "A";
      case 1: return "B";
      case 2: return "C";
      default:
        assert false;
    }
  }
  public static void main(String[] args) {
    C c = new C();
    for (int i = 0; i < 4; i++) {
      System.out.print(c.m1(i));
}}}
```

Which statements are true?

a. With assertions enabled it prints ABC followed by an AssertionError message.

b. With assertions disabled it prints ABC followed by an AssertionError message.

c. Assertions should not be used within the default case of a switch statement.

d. In this code example a throw statement must be used in place of the assert statement.

e. Compile-time error

---

**Question 14**

```
class D {
```

```
  private boolean b1, b2;
  public void setB1(boolean b) {b1 = b;}
  public void setB2(boolean b) {b2 = b;}
  public void m1 () {
    if (!b2 & !b1) {System.out.print("A");
    } else if (!b2 & b1) {System.out.print("B");
    } else if (b2 & !b1) {System.out.print("C");
    } else {assert false;}
  }
  public static void main (String[] args) {
    D d = new D();
    d.setB1(true); d.setB2(true);
    d.m1();
}}
```

Which statements are true?

a. With assertions enabled it prints an AssertionError message.
b. With assertions enabled it prints nothing.
c. With assertions disabled it prints an AssertionError message.
d. With assertions disabled it prints nothing.
e. An assertion should not be placed at any location that the programmer believes will never be reached under normal operating conditions.
f. The assert statement is being used to check a control-flow invariant to verify that the control flow never reaches a point in the program.

---

**Question 15**

```
class A {
  private void m1 (int i) {
    assert i < 10 : i; System.out.print(i);
  }
  public void m2 (int i) {
    assert i < 10 : i; System.out.print(i);
  }
  public static void main (String[] args) {
    A a = new A(); a.m1(11); a.m2(12);
```

}}

Which statements are true?

a. If assertions are enabled at run time it prints an error message.
b. With assertions enabled it prints nothing.
c. With assertions disabled it prints an error message.
d. With assertions disabled it prints 1112.
e. With assertions disabled it prints nothing.
f. The assert statements are being used to check a precondition--something that must be true when the method is invoked.
g. Method m1 is an example of an improper use of an assert statement: an assert statement should not be used for argument checking in a non-public method.
h. Method m2 is an example of an improper use of an assert statement: an assert statement should not be used for argument checking in a public method.

---

**Question 16**

```
class B {
  int a, b, c;
  private void setA(int i) {a = i;}
  private void setB(int i) {b = i;}
  private int m1 (int i) {
    c = a + b + i; assert c < 200 : c; return c;
  }
  public static void main (String[] args) {
    B b = new B(); b.setA(50); b.setB(100); b.m1(50);
}}
```

Which statements are true?

a. If assertions are not enabled at run time it prints an error message.
b. If assertions are not enabled at run time it prints nothing.
c. With assertions enabled it prints an error message.
d. With assertions enabled it prints nothing.
e. The assert statement is being used to check a postcondition--something that must be true when the method completes successfully.

---

**Question 17**

```
class C {
  int a, b, c;
  public void setA(int i) {a = i; assert validateC() : c;}
  public void setB(int i) {b = i; assert validateC() : c;}
  private boolean validateC() {
    return c > a + 2 * b;
  }
  public int m1(int i) {
    c = a + b + i;
    assert validateC() : c;
    return c;
  }
  public C(int i) {
    c = i; assert validateC() : c;
  }
  public static void main(String[] args) {
    C c = new C(251); c.setA(50); c.setB(100);
}}
```

Which statements are true?

a. If assertions are not enabled at run time it prints an error message.
b. If assertions are not enabled at run time it prints nothing.
c. With assertions enabled it prints an error message.
d. With assertions enabled it prints nothing.
e. The assert statement is being used to check a class invariant--something that must be true about each instance of the class.
f. The assert statements are being used to check a precondition--something that must be true when the method is invoked.

---

**Question 18**

```
class E {
```

```
  private boolean b1, b2, b3;
  public void setB1(boolean b) {b1 = b;}
  public void setB2(boolean b) {b2 = b;}
  public void setB3(boolean b) {b3 = b;}
  public void m1 (int i) {
    b2 = i % 2 == 0;
    if (!b3 & !b2 & !b1) {System.out.print("A");
    } else if (!b3 & !b2 & b1) {System.out.print("B");
    } else if (!b3 & b2 & !b1) {System.out.print("C");
    } else { // Only b3 is true.
      assert b3 & !b2 & !b1;
    }
    System.out.print(b1 + "," + b2 + "," + b3);
    b1 = b2 = b3 = false;
  }
  public static void main (String[] args) {
    E e = new E(); e.setB1(true); e.m1(2);
}}
```

Which statements are true?

a. With assertions enabled it prints an error message.
b. With assertions enabled it prints: true,true,false
c. With assertions disabled it prints an error message.
d. With assertions disabled it prints: true,true,false
e. With assertions disabled it prints nothing.
f. The combination of the if/else statements and the assert statement indicate that the programmer expects no more than one boolean, b1, b2 or b3, to be true.
g. The assert statement is being used to check a precondition--something that must be true when the method begins.
h. The assert statement is being used to check an internal invariant--something that the programmer assumes to be true at a particular point in the program.

---

**Question 19**

```
class F {
  private boolean b1, b2, b3;
```

```
  public void setB1(boolean b) {b1 = b;}
  public void setB2(boolean b) {b2 = b;}
  public void setB3(boolean b) {b3 = b;}
  public String m1 (int i) {
    b2 = i % 2 == 0;
    if (!b3 & !b2 & !b1) {return "None are true.";
    } else if (!b3 & !b2 & b1) {return "Only b1 is true.";
    } else if (!b3 & b2 & !b1) {return "Only b2 is true.";
    } else if (b3 & !b2 & !b1) {return "Only b3 is true.";
    } else {throw new AssertionError();}
  }
  public static void main (String[] args) {
    F f = new F();
    f.setB1(true);
    System.out.println(f.m1(5));
    System.out.println(f.m1(6));
}}
```

Which statements are true?

a. Prints "Only b1 is true" followed by an error message.
b. An assertion should not be placed at any location that the programmer believes will never be reached under normal operating conditions.
c. The combination of the if/else statements and the assert statement indicate that the programmer expects no more than one boolean, b1, b2, or b3, to be true.
d. The assert statement is being used to check a precondition--something that must be true when the method begins.
e. The assert statement is being used to check a control-flow invariant to verify that the control flow never reaches a point in the program.
f. The throw statement could be replaced by an assert statement.

---

**Question 20**

An assert statement can be used to check a control-flow invariant to verify which of the following?

a. A particular assumption is true when the flow of control enters a method.
b. The flow of control does not reach a particular point in the program.
c. The normal flow of control has reached a particular point in the program.

d. The normal flow of control has reached the end of a method.

e. The default case of a switch statement is not reached.

f. The else block of an if/else statement is not reached.

1.a e .-ea -enableassertions .Two command-line switches used to enable assertions in non-system classes are -ea and -enableassertions.

2.a e .-da -disableassertions .Two command-line switches used to disable assertions are -da and -disableassertions. Remember that all of the letters are lower case.

3.a .-disableassertions .Two command-line switches used to disable assertions in non-system classes are -da and -disableassertions. Remember that all of the letters are lower case.

4.c g .-esa -enablesystemassertions .Two command-line switches used to enable assertions in system classes are -esa and -enablesystemassertions. Remember that all of the letters are lower case.

5.a b d e f .By default assertions are disabled at run-time. Assertions can be selectively enabled for any named class. Assertions can be selectively enabled for any named package. If assertions are selectively enabled for any named package then assertions are automatically enabled for the subpackages. Assertions can be selectively enable for the unnamed package in the current working directory. .

6.c .AssertionError .An AssertionError is thrown to indicate that an assertion has failed. Don't be fooled by the name AssertionException.

7.b e .With assertions enabled it prints 210210 followed by an AssertionError message. With assertions disabled it prints 210210-1 .If, under normal operating circumstances, the default label of a switch statement should not be reached, then an assert statement can be placed after the default label to verify that an unexpected condition has not not occurred.

8.b e g .With assertions enabled it prints 012. With assertions disabled it attempts to print an infinite sequence of numbers. As a rule, the boolean expression of an assert statement should not be used to perform actions that are required for normal operation of the program. .An assert statement should not be used as demonstrated in the program. The boolean expression of the do-loop depends on the value of the local variable i1. The value of i1 is set within the boolean expression of the assert statement. If assertions are disabled, then the boolean expression of the assert statement is not processed and the value of i1 is not updated with each iteration of the loop; so the loop runs indefinitely.

9.a d .Assertions should not be used to validate arguments passed to public methods. Assertions should not be used to perform processing that is required for the normal operation of the application. .Assertions may be enabled or disabled at run time. Since assertions are not always enabled, they should not be used to validate the parameters of public methods. Parameter checking is typically published in the API specification of a method and must be enforced even when assertions are not enabled. Rather than use an assertion, an appropriate runtime exception should be thrown such as IllegalArgumentException, IndexOutOfBoundsException, or NullPointerException. However, an assertion may be used to validate the parameters of a nonpublic method. Since assertions are not always enabled, an assertion should not be used to perform operations that are required for the normal operation of the program. For example, the boolean expression of an assertion should not be used to produce the side effect of incrementing a variable that controls a loop statement. If assertions are disabled then the loop is unlikely to function as intended. Section 14.20 of the Java Language Specification defines "unreachable" statements. If an assert statement is "unreachable" as defined by the JLS, then a compile-time error is generated. In contrast, a programmer may believe that some points in the code will not be reached as a result of design assumptions. For example, a programmer may believe that the default case of a switch statement will never be reached. An assertion can be placed in the default case to verify the behavior of the switch statement.

10.b c .The compiler will generate an error if an assert statement is "unreachable" as defined by the Java Language Specification. A catch clause should not be used to catch an AssertionError. .Section 14.20 of the Java Language Specification defines "unreachable" statements. If an assert statement is "unreachable" as defined by the JLS, then a compile-time error is generated. In contrast, a programmer may believe that some points in the code will not be reached as a result of

design assumptions. For example, a programmer may believe that the default case of a switch statement will never be reached. An assertion can be placed in the default case to verify the behavior of the switch statement. While the exception handling mechanisms of Java have been designed to allow for recovery from Exceptions, the assertion mechanisms have been designed to discourage recovery attempts. An assertion is used to verify that the program has not strayed beyond the bounds of expected behavior. For example, suppose that you go to bed one night, and your pet dog is sleeping on the floor next to your bed. Before going to sleep, you make the assertion that your dog will still be there in the morning. When you wake up, you find that a different dog is sleeping in place of your pet. How do you recover from the failure of your assertion? Since you probably did not expect your dog to be mysteriously replaced during the night, it is unlikely that you have already developed an effective recovery routine. However, if you had planned for a dog swapping exception, then the recovery should be handled by the exception handling mechanism rather than the assertion mechanism.

11.a b d .With assertions enabled it prints ABC followed by an AssertionError message. With assertions disabled it prints ABC followed by an AssertionError message. In this code example an assert statement could not be used in place of the "throw" statement. .If the default label of a switch statement should not be reached under normal operating circumstances, then the default label might be a good location for an assert statement. If a method is declared with a non-void return type and if no return statement appears after the switch statement, then each case of the switch must have a return statement or a throw statement. The throw statement is used rather than an assert, because the compiler knows that the assert statement is not functional when assertions are disabled.

12.a d .With assertions enabled it prints ABC followed by an AssertionError message. With assertions disabled it prints ABCE .If the default label of a switch statement should not be reached under normal operating circumstances, then the default label might be a good candidate for the use of an assert statement.

13.d e .In this code example a throw statement must be used in place of the assert statement. Compile-time error .If the default label of a switch statement should not be reached under normal operating circumstances, then the default case becomes a good candidate for the use of an assert statement. If a method is declared with a non-void return type and if no return statement appears after the switch statement, then each case of the switch must have a return statement or a throw statement. The throw statement is used rather than an assert, because the compiler knows that the assert statement is not functional when assertions are disabled.

14.a d f .With assertions enabled it prints an AssertionError message. With assertions disabled it prints nothing. The assert statement is being used to check a control-flow invariant to verify that the control flow never reaches a point in the program. .The assert statement indicates that the programmer believes that b1 and b2 will never be true simultaneously, and the assert statement should not be reached under normal operating conditions.

15.a d f h .If assertions are enabled at run time it prints an error message. With assertions disabled it prints 1112. The assert statements are being used to check a precondition--something that must be true when the method is invoked. Method m2 is an example of an improper use of an assert statement: an assert statement should not be used for argument checking in a public method. .Assertions may be enabled or disabled at run time. Since assertions are not always enabled, they should not be used to validate the parameters of public methods. Parameter checking is typically published in the API specification of a method and must be enforced even when assertions are not enabled. Rather than use an assertion, an appropriate runtime exception should be thrown such as IllegalArgumentException, IndexOutOfBoundsException, or NullPointerException. However, an assertion may be used to validate the parameters of a nonpublic method.

16.b c e .If assertions are not enabled at run time it prints nothing. With assertions enabled it prints an error message. The assert statement is being used to check a postcondition--something that must be true when the method completes successfully. .Variable c equals 200 when the assertion is checked.

17.b d e .If assertions are not enabled at run time it prints nothing. With assertions enabled it prints nothing. The assert statement is being used to check a class invariant--something that must be true about each instance of the class. .This question is an example of using assertions to check a class invariant--something that must be true about each instance of the class. Although a class invariant must be true before and after the execution of each public method, the invariant is typically only checked at the end of each method and constructor.

18.a d f h .With assertions enabled it prints an error message. With assertions disabled it prints: true,true,false The combination of the if/else statements and the assert statement indicate that the programmer expects no more than one boolean, b1, b2 or b3, to be true. The assert statement is being used to check an internal

invariant--something that the programmer assumes to be true at a particular point in the program. .Method m1 has a series of if/else statements. The first if statement is processed if none of the booleans are true. The second is processed if only b1 is true. The third is processed if only b2 is true. A set of three booleans can exist is eight states. The three if statements account for three of those states; so five more states remain. The assert statement indicates that the programmer assumes that only one of those five remaining states is valid--that is the state where only b3 is true. The combination of the three if statements and the assert statement indicate that the programmer believes that no more than one of the booleans will be true at that point in the program. That assumption is called an internal invariant.

19.a c e .Prints "Only b1 is true" followed by an error message. The combination of the if/else statements and the assert statement indicate that the programmer expects no more than one boolean, b1, b2, or b3, to be true. The assert statement is being used to check a control-flow invariant to verify that the control flow never reaches a point in the program. .Method m1 has a series of if/else statements. The first if statement is processed if none of the booleans are true. The second is processed if only b1 is true. The third is processed if only b2 is true. The fourth is processed if only b3 is true. A set of three booleans can exist in one of eight states. The first four if statements account for four of those states; so four more states remain. The combination of the three if statements and the fact that an AssertionError is thrown from the last else block indicates that the programmer believes that no more than one of the booleans will be true when method m1 is being processed. An assumption concerning the state of a set of variables is called an internal invariant. In this case, however, the assertion was tested by verifying that control never reached a particular point in the program. Based on the testing technique, we would say that the assertion tests a control-flow invariant. A throw statement is used in place of an assert statement, because the throw statement can not be disabled. As a result, the method is certain to generate an error once control passes beyond all of the return statements. The declared return type of method m1 is String. No return statement appears after the sequence of if statements; therefore, every if statement must either return a String or throw an exception. Assertions can be disabled at run time, so an assert statement in the final if block is no guarantee that an exception will be thrown. For that reason, an assert can not replace the throw statement.

20.b e f .The flow of control does not reach a particular point in the program. The default case of a switch statement is not reached. The else block of an if/else statement is not reached. .A control-flow invariant is placed at a point in the program that the programmer assumes will never be reached. Two examples are the default case of a switch statement or the else block of an if/else statement. It makes no sense to use an assert statement to verify that the flow of control does reach a particular point in the program, because it is unlikely that an assertion error is helpful when the program is found to be functioning correctly. An assert statement placed at the beginning of a method is generally used to check a precondition. An assert statement that is placed at the end of a method to check the state of some variables is generally said to be checking a post condition. However, it is also possible that an assert statement placed at the end of a method might also be checking a control-flow invariant. The correct term depends on the usage.

**Garbage Collection**

**Question 1**

```
class B {
  private String name;
  public B(String s) {name = s;}
  protected void finalize() {System.out.print(name);}
}
class E {
  public static void m() {
    B x1 = new B("X"), y1 = new B("Y");
```

```
  }
  public static void main(String[] args) {
    m(); System.gc();
}}
```

Which of the following could not be a result of attempting to compile and run the program?

a. Prints: XY
b. Prints: YX
c. Prints: XXYY
d. Nothing is printed.
e. None of the above

---

**Question 2**

```
void m1() {
  Q q1 = null;
  for (int i = 0; i < 10; i++) {
    q1 = new Q();   // 1
    m2(q1);         // 2
  }
  System.out.print("All done"); // 3
}
```

When the processing of line 3 begins, how many objects of type Q that were created at line 1 have become eligible for garbage collection?

a. 0
b. 1
c. 9
d. 10
e. Indeterminate.
f. Compile-time error
g. Run-time error

h. None of the above

---

**Question 3**

```
class Q {
  private int id;
  protected void finalize() {System.out.print(id);}
  public Q(int i) {id = i;}
}
class R {
  public static void main(String[] args) {
    Q q1 = null;
    for (int i = 0; i < 10; i++) {q1 = new Q(i);} // 1
    System.gc();     // 2
}}
```

When the processing of line 2 begins, how many objects of type Q that were created at line 1 have become eligible for garbage collection?

a. 0
b. 1
c. 9
d. 10
e. Indeterminate.
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 4**

```
class I {
  private I other;
  public void other(I i) {other = i;}
}
```

```
class J {
  private void m1() {
    I i1 = new I(), i2 = new I();
    I i3 = new I(), i4 = new I();
    i1.other(i3); i2.other(i1);
    i3.other(i2); i4.other(i4);
  }
  public static void main (String[] args) {
    new J().m1();
}}
```

Which object is not eligible for garbage collection after method m1 returns?

a. i1
b. i2
c. i3
d. i4
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 5**

```
class I {
  private String name;
  public I(String s) {name = s;}
  private I other;
  public void other(I i) {other = i;}
}
class J {
  private I i1 = new I("A"), i2 = new I("B"), i3 = new I("C");
  private void m1() {
    i1.other(i2); i2.other(i1); i3.other(i3);
    i1 = i3; i2 = i3;
```

```
    m2();
  }
  private void m2() {/* Do amazing things. */}
  public static void main (String[] args) {
    new J().m1();
}}
```

Which of the three objects, A, B or C, is not eligible for garbage collection when method m2 begins to execute?

a. A
b. B
c. C
d. None of the above

---

**Question 6**

```
class I {
  private String name;
  protected void finalize() {System.out.print(name);}
  public I(String s) {name = s;}
}
class J {
  private static void m1(I[] a1) {
    a1[0] = a1[1] = a1[2] = null;
  }
  public static void main (String[] args) {
    I[] a1 = new I[3];   // 1
    a1[0] = new I("A");  // 2
    a1[1] = new I("B");  // 3
    a1[2] = new I("C");  // 4
    m1(a1);
    System.gc();
}}
```

After method m1 returns, the object created on which line is not eligible for garbage collection?

a. 1

b. 2

c. 3

d. 4

e. None of the above

f. Compile-time error

g. Run-time error

---

**Question 7**

```
class I {
  private String name;
  public String toString() {return name;}
  public I(String s) {name = s;}
}
class J {
  private static void m1(I[] a1) {a1 = null;}
  public static void main (String[] args) {
    I[] a1 = new I[3];   // 1
    a1[0] = new I("A");  // 2
    a1[1] = new I("B");  // 3
    a1[2] = new I("C");  // 4
    m1(a1);
    for (int i = 0; i < a1.length; i++) {
      System.out.print(a1[i]);
}}}
```

After method m1 returns, the object created on which line is eligible for garbage collection?

a. 1

b. 2

c. 3

d. 4

e. Compile-time error

f. Run-time error

g. None of the above

---

**Question 8**

```
class A {
  private String name;
  private A otherA;
  public A(String name) {this.name = name;}
  public void other(A otherA) {this.otherA = otherA;}
  public A other() {return otherA;}
  public String toString() {return name;}
  protected void finalize() {System.out.print(name);}
}
class B {
  public static void m1() {
    A a1 = new A("A1"), a2 = new A("A2"), a3 = new A("A3"), a0 = a3;
    a1.other(a2); a2.other(a3); a3.other(a1);
    for(int i = 0; i<4; i++){System.out.print(a0 = a0.other());}
  }
  public static void main(String[] args) {m1(); System.gc();}
}
```

Which of the following could be a result of attempting to compile and run the program?

a. A1A2A3A1

b. A0A0A0A0A1A2A3

c. A1A2A3A1A2A3

d. A1A2A3A1A1A2A3

e. A1A2A3A1A3A2A1

f. A0A1A2A3A1A2A3

---

**Question 9**

```
class B {
  private String name;
  public B(String name) {this.name = name;}
  public String toString() {return name;}
  protected void finalize() {System.out.print(name);}
}
class H {
  static B ba = new B("Ba");
  static int i = 1;
  static B m1(B b) {return b = new B("B" + i++);}
  public static void main (String[] args) {
    B x = m1(ba); m1(x);
    System.out.println(", " + ba + ", " + x);
}}
```

Which of the following could be a result of attempting to compile and run the program?

a. Ba, B1, B2
b. B1, Ba, B2
c. , Ba, B1
d. B2, Ba, B1
e. BaB1b2, null, null
f. B1B2, ba, null

---

**Question 10**

```
class B {
  private String name;
  public B(String name) {this.name = name;}
  public String toString() {return name;}
  protected void finalize() {System.out.print(name);}
}
class J {
```

```
  static B bc;
  static int i = 1;
  static B m1(B b) {bc = b; return new B("B" + i++);}
  public static void main (String[] args) {
    B x = m1(new B("Ba")), y = m1(new B("Bb"));
    System.out.println(", " + x + ", " + y + ", " + bc);
}}
```

Which of the following could be a result of attempting to compile and run the program?

a. BaBb, B1, B2, B2
b. B1B2, null, null, Bb
c. , Ba, Bb, Bb
d. BaBbB1B2, null, null, null
e. Ba, B1, B2, Bb
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 11**

```
class I {
  private String name;
  public String name() {return name;}
  public I(String s) {name = s;}
}
class J {
  public static void m1(I i) {i = null;}
  public static void main (String[] args) {
    I i = new I("X");          // 1
    m1(i);                     // 2
    System.out.print(i.name()); // 3
}}
```

Which of the following is a true statement?

a. The object created a line 1 is eligible for garbage collection after line 2.

b. A NullPointerException is generated at line 3.

c. The program compiles, runs and prints X.

d. The program fails to compile.

e. None of the above

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

2 e Indeterminate. Since we don't know what method m2 might be doing, we can not know if the objects are eligible for garbage collection. Suppose that method m2 is declared inside of a class that also contains 10 instance variables (instance variables are non-static member fields) that are references to instances of class A. The argument that appears in the method invocation expression m2(q1) is a reference to an instance of class Q. Suppose that m2 saves each argument value in one of the ten instance variables or in an element of an array of type Q[]. When the loop in method m1 runs to completion, each instance of class Q would still be referenced by a one of the ten instance variables. Since the instance variables would continue to reference each instance of class Q when line 3 is executed, none of the instances would be eligible for garbage collection at that point. A second possibility is that method m2 does not save the reference values. In that case, all of the instances that were created inside the loop would be eligible for garbage collection when line 3 is executed.

3 c 9 With each pass through the loop, q1 references a new object, and the old object becomes eligible for garbage collection. When the processing of line 2 begins, the last object referenced by q1 is not eligible for garbage collection.

4 g None of the above Please note that this question asks which object is NOT eligible for garbage collection after method m1 returns. The objects referenced by i1, i2 and i3 form a ring such that each object is referenced by another. Even so, nothing outside of method J.m1 references any of those objects. When method J.m1 returns, the ring becomes an island of isolated objects that are not reachable by any part of the user program. A key point to remember is that an object that is referenced by another object can be eligible for garbage collection if the two objects form an island of isolated objects.

5 c C Please note that this question asks which objects are NOT eligible for garbage collection when method m2 begins to execute? All three references, i1, i2 and i3, refer to object named C; so C is not eligible for garbage collection when method m2 begins to execute. The objects named A and B have references to each other, but no other objects refer to A and B. The objects A and B form an island of islolated objects and are eligible for garbage collection.

6 a 1 Please note that this question asks which objects are NOT eligible for garbage collection after method m1 returns. After method m1 returns, the array a1 created on line 1 is not eligible for garbage collection. Method m1 sets all elements of the array to null; so the objects created on lines 2, 3 and 4 are eligible for garbage collection when method m1 returns.

7 g None of the above After method m1 returns, none of the objects are eligible for garbage collection. Method m1 sets the parameter variable a1 to null, but that does not change the reference a1 in the J.main method. Since array a1 continues to reference all three objects, none of the three are eligible for garbage collection.

8 a c d e A1A2A3A1 A1A2A3A1A2A3 A1A2A3A1A1A2A3 A1A2A3A1A3A2A1 The three instances of class A form an isolated ring where each instance references the next instance and the third references the first instance. Four iterations of the for loop are processed. Inside the body of the for statement, the invocation of the print method contains the argument expression a0 = a0.other(). On the first iteration, the reference variable a0 references the instance named A3. The value returned by the method named other is a reference to the instance named A1. The reference is assigned to the reference variable a0 and is also the value produced by the expression a0 = a0.other(). That reference value is passed as an argument to the print method, and the print method invokes the A.toString method. With each iteration of the loop, the reference moves to the next object in the loop and the name of the object is printed. After four iterations, the loop ends and the method m1 returns. The invocation of the System.gc method serves as a suggestion that the garbage collector should be allowed to run. The system could ignore the suggestion, so there is no guarantee that the eligible arguments will be garbage collected. If they are collected, there is no guarantee which will be collected first. The only guarantee is that the finalize method will be invoked on each particular instance before the resources that had been allocated to that instance are reclaimed.

9 c d , Ba, B1 B2, Ba, B1 Class H declares two static member variables named ba and i. The type of i is int, and the value is initialized to 1. The type of ba is B. The declaration of ba contains the class instance creation expression new B("Ba"). The constructor of class B assigns the argument value to the instance variable called name. Inside the main method of class H, the method invocation expression m1(ba) invokes method m1. The argument is the static member variable ba. The body of method m1 contains a return statement with the expression b = new B("B" + i++). The assignment expression contains the class instance creation expression new B("B" + i++) which creates a new instance of the class B. For this first invocation of method m1, the argument appearing in the class instance creation expression is the String value B1. The reference to the new String is assigned to the parameter variable b, but that assignment does not change the value of the member variable ba. The value of the assignment expression is the reference to the new instance of class B with the name B1, and that reference value is returned by the method m1. The returned value is assigned to the local variable x. The next statement inside the main method is another invocation of method m1. The argument appearing in the method invocation expression m1(x) is the local reference variable x. The method invocation does not change the value of x. The value returned by this second invocation of m1 is a reference to a new instance of class B that has the name B2. The returned reference value is not assigned to a variable, so the instance named B2 is eligible for garbage collection. There is no guarantee that the garbage collector will run before the print statement is invoked. If it does run, then the instance named B2 could be finalized causing the name to be printed.

10 e Ba, B1, B2, Bb Class J declares two static member variables named bc and i. The type of i is int, and the value is initialized to 1. The type of bc is B. Inside the main method of class J, the method invocation expression m1(new B("Ba")) invokes method m1. The argument is the class instance creation expression new B("Ba"). The constructor of class B assigns the argument value to the instance variable called name, so a new instance of class B named Ba is created. The reference to the new instance of class B is the argument that is passed to method m1. The body of method m1 contains two statements. The first contains the assignment expression bc = b that assigns the value of the method parameter b to the static member variable bc, so bc now references the instance of class B named Ba. The second statement in the body of m1 is a return statement with the class instance creation expression new B("B" + i++). For this first invocation of method m1, the argument appearing in the class instance creation expression is the String value B1. The reference to the new String is returned by method m1. The returned value is assigned to the local variable x. Inside the main method, the declaration of the local variable y contains another invocation of method m1. The argument appearing in the method invocation expression m1(new B("Bb")) is the class instance creation expression new B("Bb"), so the argument value for the invocation of method m1 is a reference to a new instance of class B named Bb. Inside the body of method m1, the reference to the new instance of class B named Bb is assigned to the static member variable Bc. At that point, the instance of class B named Ba becomes eligible for garbage collection. Method m1 returns a reference to a new instance of class B named B2. There is no guarantee that the garbage collector will run before the print statement is invoked. If it does run, then the instance named Ba could be finalized causing the name to be printed.

11 c  The program compiles, runs and prints X.  The parameter i of method m1 is a copy of the local variable i of method J.main. Setting the parameter variable i of method m1 to null does not change the local variable i of method J.main.

**Packages**

**Question 1**

```
package com.dan.chisholm;
public class A {
  public void m1() {System.out.print("A.m1, ");}
  protected void m2() {System.out.print("A.m2, ");}
  private void m3() {System.out.print("A.m3, ");}
  void m4() {System.out.print("A.m4, ");}
}
class B {
  public static void main(String[] args) {
    A a = new A();
    a.m1();  // 1
    a.m2();  // 2
    a.m3();  // 3
    a.m4();  // 4
}}
```

Assume that the code appears in a single file named A.java. What is the result of attempting to compile and run the program?

a. Prints: A.m1, A.m2, A.m3, A.m4,
b. Compile-time error at 1.
c. Compile-time error at 2.
d. Compile-time error at 3.
e. Compile-time error at 4.
f. None of the above

---

**Question 2**

```
// Class A is declared in a file named A.java.
```

```
package com.dan.chisholm;
public class A {
  public void m1() {System.out.print("A.m1, ");}
  protected void m2() {System.out.print("A.m2, ");}
  private void m3() {System.out.print("A.m3, ");}
  void m4() {System.out.print("A.m4, ");}
}
// Class D is declared in a file named D.java.
package com.dan.chisholm.other;
import com.dan.chisholm.A;
public class D {
  public static void main(String[] args) {
    A a = new A();
    a.m1();  // 1
    a.m2();  // 2
    a.m3();  // 3
    a.m4();  // 4
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A.m1, A.m2, A.m3, A.m4,
b. Compile-time error at 1.
c. Compile-time error at 2.
d. Compile-time error at 3.
e. Compile-time error at 4.

---

**Question 3**

```
// Class A is declared in a file named A.java.
package com.dan.chisholm;
public class A {
   public void m1() {System.out.print("A.m1, ");}
   protected void m2() {System.out.print("A.m2, ");}
   private void m3() {System.out.print("A.m3, ");}
```

```
   void m4() {System.out.print("A.m4, ");}
}
// Class C is declared in a file named C.java.
package com.dan.chisholm.other;
import com.dan.chisholm.A;
public class C extends A {
  public static void main(String[] args) {
    C c = new C();
    c.m1();  // 1
    c.m2();  // 2
    c.m3();  // 3
    c.m4();  // 4
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A.m1, A.m2, A.m3, A.m4,

b. Compile-time error at 1.

c. Compile-time error at 2.

d. Compile-time error at 3.

e. Compile-time error at 4.

1.d .Compile-time error at 3. .Both class A and B are declared in the same package, so class B has access to the public, protected, and package access methods of class A.

2.c d e .Compile-time error at 2. Compile-time error at 3. Compile-time error at 4. .Classes A and D are not declared in the same package, so class D does not have access to package access method, m4. Since class D does not extend class A, class D does not have access to the protected method, m2, of class A.

3.d e .Compile-time error at 3. Compile-time error at 4. .Class A and C are not declared in the same package; therefore, class C does not have access to package access method, m4. Since class C extends class A, class C does have access to the protected method, m2, of class A.

**Interfaces**

**Question 1**

```
interface A {
  void m1();           // 1
  public void m2();    // 2
  protected void m3();  // 3
```

```
  private void m4();    // 4
}
```

Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3
d. 4

---

**Question 2**

Which of the following statements is not true?

a. An interface that is declared within the body of a class or interface is known as a nested interface.
b. A constant can be a member of an interface.
c. A class declaration can be a member of an interface.
d. If an interface is named in the implements clause of a class, then the class must implement all of the methods declared within the interface.
e. None of the above.

---

**Question 3**

Which of the following statements are true?

a. An interface declaration can be a member of an interface.
b. A method declared within an interface must have a body represented by empty curly braces.
c. An interface can implement another interface.
d. An abstract class that implements an interface must implement all abstract methods declared within the interface.
e. An abstract method declaration can be a member of an interface.

---

**Question 4**

Which of the following are modifiers that can be applied to an interface that is a member of a directly enclosing interface?

a. Abstract
b. implements
c. Final
d. Private
e. protected
f. public

---

**Question 5**

Which of the following are modifiers that can be applied to an interface that is a member of a directly enclosing class?

a. abstract
b. extends
c. final
d. private
e. protected
f. Public

---

**Question 6**

Which of the following is a modifier that can be applied to an interface that is a member of a directly enclosing class or interface?

a. Static
b. synchronized
c. Transient
d. Volatile
e. implements
f. None of the above.

---

**Question 7**

Suppose that an interface, I1, is not a member of an enclosing class or interface. Which of the following modifiers can be applied to interface I1?

a. Abstract
b. Final
c. Private
d. Protected
e. Public

---

**Question 8**

Suppose that an interface, I1, is not a member of an enclosing class or interface. Which of the following modifiers can be applied to interface I1?

a. Abstract
b. Public
c. Static
d. synchronized
e. Transient
f. Volatile

---

**Question 9**

Which of the following are modifiers that can be applied to a field declaration within an interface?

a. abstract
b. Const
c. Final
d. private
e. protected
f. Public

---

**Question 10**

Which of the following is a modifier that can be applied to a field declaration within an interface?

a. static
b. Synchronized
c. Transient
d. Volatile
e. None of the above.

---

**Question 11**

Which of the following modifiers can be applied to a class that is declared within an enclosing interface?

a. Public
b. Protected
c. Private
d. Abstract
e. Static
f. Final

---

**Question 12**

```
interface A {
  int a = 1;                    // 1
  public int b = 2;             // 2
  public static int c = 3;      // 3
  public static final int d = 4;    // 4
}
```

Which field declaration results in a compile-time error?

a. 1
b. 2
c. 3
d. 4
e. None of the above

---

**Question 13**

```
interface A {
  protected int e = 5;             // 1
  private int f = 6;               // 2
  volatile int g = 7;              // 3
  transient int h = 8;             // 4
  public static final int d = 9;   // 5
}
```

Which of the field declarations does not result in a compile-time error?

a. 1
b. 2
c. 3
d. 4
e. 5
f. None of the above

---

**Question 14**

Which of the following are modifiers that can be applied to a method declaration within an interface?

a. abstract
b. Final
c. private

d. protected
e. Public

---

**Question 15**

Which of the following is a modifier that can be applied to a method declaration within an interface?

a. Static
b. synchronized
c. transient
d. volatile
e. Native
f. None of the above

---

**Question 16**

```
interface A {void m1();}                    // 1
class B implements A {public void m1() {}}    // 2
class C implements A {protected void m1() {}} // 3
class D implements A {private void m1() {}}   // 4
class E implements A {void m1() {}}          // 5
```

Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3
d. 4
e. 5

---

**Question 17**

```
interface A {
  void m1();              // 1
  public void m2();       // 2
  protected void m3();    // 3
  private void m4();      // 4
  abstract void m5();     // 5
}
```

Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3
d. 4
e. 5

---

**Question 18**

```
interface A {
  final void m1();        // 1
  synchronized void m2(); // 2
  native void m3();       // 3
  abstract void m4();     // 4
  public void m5();       // 5
}
```

Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3
d. 4
e. 5

---

**Question 19**

```
interface A {void main(String[] args);}              // 1
interface B {public void main(String[] args);}       // 2
interface C {public static void main(String[] args);} // 3
interface D {protected void main(String[] args);}    // 4
interface E {private void main(String[] args);}      // 5
```

Which interface declarations generate a Compile-time error?

a. 1
b. 2
c. 3
d. 4
e. 5

---

**Question 20**

```
interface F {abstract void main(String[] args);}     // 1
interface G {synchronized void main(String[] args);} // 2
interface H {final void main(String[] args);}        // 3
interface I {native void main(String[] args);}       // 4
```

Which interface declaration does not generate a compile-time error?

a. 1
b. 2
c. 3
d. 4
e. None of the above

---

**Question 21**

```
interface A {String s1 = "A"; String m1();}
interface B implements A {String s1 = "B"; String m1();}
class C implements B {
  public String m1() {return s1;}
  public static void main(String[] args) {
    A a = new C(); System.out.print(a.m1());
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A
b. Prints: B
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 22**

```
interface A {int i = 1; int m1();}
interface B extends A {int i = 10; int m1();}
class C implements B {
  public int m1() {return ++i;}
  public static void main(String[] args) {
    System.out.print(new C().m1());
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 2
b. Prints: 11
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 23**

```
interface Z {void m1();}  // 1
class A implements Z {void m1() {}} // 2
class B implements Z {public void m1() {}} // 3
abstract class C implements Z {public abstract void m1();} // 4
```

A Compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. None of the above

---

**Question 24**

```
interface Z {void m1();}  // 1
class D implements Z {public final void m1() {}}    // 2
class E implements Z {public synchronized void m1() {}} // 3
class G implements Z {public native void m1();}     // 4
```

A Compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. None of the above

---

**Question 25**

```
interface I10 {String name = "I10"; String s10 = "I10.s10";}
interface I20 {String name = "I20"; String s20 = "I20.s20";}
class C10 implements I10, I20 {           // 1
  public static void main(String[] args) {
    System.out.print(s10+",");            // 2
    System.out.print(s20+",");            // 3
    System.out.print(name);               // 4
}}
```

What is the result of attempting to compile and run the program?

a. Prints: I10.s10,I20.s20,I10
b. Prints: I10.s10,I20.s20,I20
c. Prints: I10.s10,I20.s20,
d. Prints: I10.s10,I20.s20,null
e. Compile-time error at line 1
f. Compile-time error at line 2
g. Compile-time error at line 3
h. Compile-time error at line 4
i. Run-time error
j. None of the above

---

**Question 26**

```
interface I10 {String name = "I10"; String s10 = "I10.s10";}
interface I20 {String name = "I20"; String s20 = "I20.s20";}
class C20 implements I10, I20 {         // 1
  public static void main(String[] args) {
    System.out.print(I10.s10+",");      // 2
    System.out.print(I20.s20+",");      // 3
    System.out.print(I20.name);         // 4
}}
```

What is the result of attempting to compile and run the program?

a. Prints: I10.s10,I20.s20,I10
b. Prints: I10.s10,I20.s20,I20
c. Prints: I10.s10,I20.s20,
d. Prints: I10.s10,I20.s20,null
e. Compile-time error at line 1
f. Compile-time error at line 2
g. Compile-time error at line 3
h. Compile-time error at line 4
i. Run-time error
j. None of the above

1 c d 3 4 Methods declared within an interface are implicitly public. If no access modifier is included in the method declaration; then, the declaration is implicitly public. An attempt to declare the method using a weaker access privilege, private or protected, results in a compile-time error.

2 d If an interface is named in the implements clause of a class, then the class must implement all of the methods declared within the interface. This question asks which answer option is not true. Some true statements are as follows. An interface can be declared within an enclosing class or interface. The members of an interface can be constants, abstract method declarations, class declarations or interface declarations. If an interface is named in the implements clause of a class, then the class must implement all of the methods declared within the interface or the class must be declared abstract. The untrue answer option did not mention that an abstract class is not required to implement any of the methods declared in an interface that is named in the implements clause of the class declaration.

3 a e An interface declaration can be a member of an interface. An abstract method declaration can be a member of an interface. An interface can be declared within an enclosing class or interface. The members of an interface can be constants, abstract method declarations, class declarations, or interface declarations. The body of a method declared within an interface is a semicolon. An interface can extend another interface, but can not implement an interface. An abstract class that has an interface, I1, in its implements clause is not required to implement any of the methods declared within I1.

4 a f abstract public All interfaces are implicitly abstract. The explicit application of the abstract modifier to an interface declaration is redundant and is strongly discouraged. The declaration of an interface within the body of an enclosing class or interface is called a member type declaration. Every member type declaration appearing within the body of a directly enclosing interface is implicitly static and public. Use of the access modifiers, private or protected, is contradictory and results in a compile-time error. In contrast, the modifiers, private and protected, are applicable to a member type declaration appearing within the body of a directly enclosing class. The modifier, final, is never applicable to an interface. The keyword, implements, is not a modifier.

5 a d e f abstract private protected public All interfaces are implicitly abstract. The explicit application of the modifier, abstract, to an interface is redundant and is strongly discouraged. The declaration of an interface within the body of an enclosing class or interface is called a member type declaration. The private, protected and static modifiers are applicable to a member type declaration that appears in the body of a directly enclosing class. In contrast, the modifiers, private and protected, are not applicable to a member type declaration appearing within the body of a directly enclosing interface. The modifier, final, is never applicable to an interface. The keyword, extends, is not a modifier.

6 a static A member interface is always implicitly static. The modifier, static, can not be applied to an interface that is not a member interface. The modifier, synchronized, is applicable to a concrete implementation of a method, but is not applicable to any interface. The modifiers, volatile and transient, are only applicable to variables that are members of a class. The keyword, implements, is not a modifier.

7 a e abstract public The modifier, abstract, is applicable to an interface declaration, but its use is strongly discouraged; because every interface is implicitly abstract. An interface can not be final. The modifiers, private and protected, are applicable only to an interface declaration that is a member of a directly enclosing class declaration. If an interface is not a member of a directly enclosing class, or if the interface is a member of a directly enclosing interface; then, the modifiers, private and protected, are not applicable. If an interface is declare public, then the compiler will generate an error if the class is not stored in a file that has the same name as the interface plus the extension .java.

8 a b abstract public The modifier, abstract, is applicable to an interface declaration, but its use is strongly discouraged; because every interface is implicitly abstract. If an interface is declare public, then the compiler will generate an error if the class is not stored in a file that has the same name as the interface plus the extension .java. The modifier, static, is applicable to a member interface, but not to an interface that is not nested. The modifier, synchronized, is applicable only to concrete implementations of methods. The modifiers, transient and volatile, are applicable only to variables.

9 c f final public The modifier, abstract, is not applicable to a variable. All field declarations within an interface are implicitly public, static and final. Use of those modifiers is redundant but legal. Although const is a Java keyword, it is not currently used by the Java programming language. An interface member can never be private or protected.

10 a static All field declarations within an interface are implicitly public, static and final. Use of these modifiers is redundant but legal. A field that is declared final can not also be declared volatile; so a field of an interface can not be declared volatile. The modifier, synchronized, is never applicable to a field.

11 a d e f public abstract static final A class that is declared within an enclosing interface is implicitly public and static; so the access modifiers, protected and private, are not applicable.

12 e None of the above All field declarations within an interface are implicitly public, static and final. Use of these modifiers is redundant but legal. No other modifiers can be applied to a field declaration within an interface.

13 e 5 All field declarations within an interface are implicitly public, static and final. Use of these modifiers is redundant but legal. No other modifiers can be applied to a field declaration within an interface.

14 a e abstract public All methods declared within an interface are implicitly abstract and public. Although the abstract and public modifiers can legally be applied to a method declaration in an interface, the usage is redundant and is discouraged. An abstract method can not also be declared private, static, final, native or synchronized; so the same restriction applies to methods declared within an interface.

15 f None of the above All methods declared within an interface are implicitly abstract and public. Although the abstract and public modifiers can legally be applied to a method declaration in an interface, the usage is redundant and is discouraged. An abstract method can not also be declared private, static, final, native or synchronized; so the same restriction applies to methods declared within an interface. Transient and volatile are not method modifiers.

16 c d e 3 4 5 Methods declared within an interface are implicitly public even if the modifier, public, is omitted from the declaration. Within the body of a class declaration, an attempt to implement the method using a weaker access privilege, private, protected or package access, results in a compile-time error.

17 c d 3 4 All methods declared within an interface are implicitly abstract and public. Although the abstract and public modifiers can legally be applied to a method declaration in an interface, the usage is redundant and is discouraged. Since all methods declared within an interface are implicitly public, a weaker access level can not be declared.

18 a b c 1 2 3 All methods declared within an interface are implicitly abstract and public. Although the abstract and public modifiers can legally be applied to a method declaration in an interface, the usage is redundant and is discouraged. The final, synchronized and native modifiers can not appear in the declaration of an abstract method, and can not be applied to an abstract method declared within an interface.

19 c d e 3 4 5 All methods declared within an interface are implicitly abstract and public. Although the abstract and public modifiers can legally be applied to a method declaration in an interface, the usage is redundant and is discouraged. Since all methods declared within an interface are implicitly public, a weaker access level can not be declared.
20 a 1 All methods declared within an interface are implicitly abstract. The final, synchronized and native modifiers can not appear in the declaration of an abstract method, and can not be applied to an abstract method declared within an interface.
21 c Compile-time error In the declaration of interface B, the keyword, extends, has been replaced by the keyword, implements.
22 c Compile-time error Fields declared within an interface are implicitly public, final, and static. A compile-time error is generated in response to the attempt to increment the value of i.
23 b 2 All methods declared within an interface are implicitly abstract and public. Although the abstract and public modifiers can legally be applied to a method declaration in an interface, the usage is redundant and is discouraged. Methods declared within an interface are implicitly public even if the modifier, public, is omitted from the declaration. Within the body of a class declaration, an attempt to implement the method using a weaker access privilege, private, protected or package access, results in a compile-time error. An abstract class that implements an interface is free to override any of the inherited method declarations with another abstract method declaration.
24 e None of the above All methods declared within an interface are implicitly abstract and public. Although the abstract and public modifiers can legally be applied to a method declaration within an interface, the usage is redundant and is discouraged. The modifiers, final, synchronized and native, can not appear in the declaration of an abstract method, but they can be added to an implementation of an abstract method.
25 h Compile-time error at line 4 Class C10 inherits ambiguous declarations of the name field. As long as the field is not referenced as a member of class C10; then, no compile-time error occurs. Line 4 generates the compile-time error, because it is the first to access the name field as a member of class C10.
26 b Prints: I10.s10,I20.s20,I20 Class C20 inherits ambiguous declarations of the name field. As long as the field is not referenced as a member of class C20; then, no compile-time error occurs. Although line 4 may appear to generate the compile-time error it does not, because name is accessed directly as a member of interface I20. Therefore, the compiler does not encounter an ambiguity.

**Main Method**

**Question 1**

```
class GRC10 {
  public static void main (String[] s) {
    System.out.print(s[1] + s[2] + s[3]);
}}
java GRC10 A B C D E F
```

What is the result of attempting to compile and run the program using the specified command line?

a. Prints: ABC
b. Prints: BCD

c. Prints: CDE
d. Prints: A B C
e. Prints: B C D
f. Prints: C D E
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 2**

class GRC4 {public static void main(String[] args) {}} // 1
class GRC5 {public static void main(String []args) {}} // 2
class GRC6 {public static void main(String args[]) {}} // 3

What is the result of attempting to compile and run the above programs?

a. Compile-time error at line 1.
b. Compile-time error at line 2.
c. Compile-time error at line 3.
d. An attempt to run GRC4 from the command line fails.
e. An attempt to run GRC5 from the command line fails.
f. An attempt to run GRC6 from the command line fails.
g. None of the above

---

**Question 3**

class GRC7 {public void main(String[] args) {}} // 1
class GRC8 {public void main(String []args) {}} // 2
class GRC9 {public void main(String args[]) {}} // 3

What is the result of attempting to compile and run the above programs?

a. Compile-time error at line 1.
b. Compile-time error at line 2.
c. Compile-time error at line 3.
d. An attempt to run GRC7 from the command line fails.
e. An attempt to run GRC8 from the command line fails.
f. An attempt to run GRC9 from the command line fails.

---

**Question 4**

```
class GRC1 {public static void main(String[] args) {}}    // 1
class GRC2 {protected static void main(String[] args) {}} // 2
class GRC3 {private static void main(String[] args) {}}   // 3
```

What is the result of attempting to compile each of the three class declarations and invoke each main method from the command line?

a. Compile-time error at line 1.
b. Compile-time error at line 2.
c. Compile-time error at line 3.
d. An attempt to run GRC1 from the command line fails.
e. An attempt to run GRC2 from the command line fails.
f. An attempt to run GRC3 from the command line fails.

1 b Prints: BCD The index for the first element of an array is zero so the first argument printed by this program is the second argument on the command line following the name of the class.
2 g None of the above Section 12.1.4 of the Java Language Specification requires that the main method must accept a single argument that is an array of components of type String. In each of the three class declarations, the single argument is indeed an array of components of type String. Please note that the square brackets within an array declaration may appear as part of the type or part of the declarator (i.e. array name).
3 d e f An attempt to run GRC7 from the command line fails. An attempt to run GRC8 from the command line fails. An attempt to run GRC9 from the command line fails. Section 12.1.4 of the JLS requires the main method to be declared static. In this example, each of the three main methods are not declared static. The result is an error at run-time.
4 e f An attempt to run GRC2 from the command line fails. An attempt to run GRC3 from the command line fails. Section 12.1.4 of the JLS requires the main method to be declared public. The main methods of GRC2 and GRC3 are not declared public and can not be invoked from the command line using a JVM that is compliant with section 12.1.4. Not every JVM enforces the rule. Even so, for the purposes of the SCJP exam, the main method should be declared as required by the JLS.

**Keywords**

**Question 1**

Which of these words belongs to the set of Java keywords?

a. qualified
b. record
c. repeat
d. restricted
e. label
f. to
g. type
h. until
i. value
j. virtual
k. xor
l. None of the above

---

**Question 2**

Which of these words belong to the set of Java keywords?

a. Exit
b. Strictfp
c. Enum
d. Super
e. Abort
f. Event
g. Goto
h. Native

i. Exception

---

**Question 3**

Which of these words belong to the set of Java keywords?

a. Double
b. Goto
c. Min
d. Extern
e. New
f. Signed
g. Finally
h. Const
i. And
j. Array
k. Do

---

**Question 4**

Which of these words belong to the set of Java keywords?

a. Declare
b. Global
c. Const
d. preserve
e. Continue
f. Float
g. Extends
h. Select

i. Break
j. Dim
k. instanceOf

---

**Question 5**

Which of these words belong to the set of Java keywords?

a. next
b. catch
c. function
d. instanceof
e. Mod
f. Const
g. Or
h. Boolean
i. Goto
j. Import
k. transient

---

**Question 6**

Which of these words belong to the set of Java keywords?

a. Byte
b. Short
c. Int
d. Long
e. Decimal
f. int64

g. Float
h. Single
i. Double
j. Boolean
k. Char
l. Unsigned
m. Array
n. String

---

**Question 7**

Which of these words belongs to the set of Java keywords?

a. clause
b. loop
c. expression
d. overrides
e. statement
f. assertion
g. validate
h. exception
i. cast
j. None of the above

---

**Question 8**

Which of these words belong to the set of Java keywords?

a. transient
b. Serializable

c. Runnable
d. Run
e. volatile
f. Externalizable
g. Cloneable

---

**Question 9**

Which of these words belong to the set of Java keywords?

a. virtual
b. goto
c. ifdef
d. typedef
e. friend
f. struct
g. Implements
h. Union
i. const

---

**Question 10**

Which of these lists contains at least one word that is not a Java keyword?

a. abstract, default, if, private, this
b. do, implements, protected, boolean, throw
c. case, extends, int, short, try
d. import, break, double, exception, throws
e. byte, else, instanceof, return, transient
f. None of the above

---

**Question 11**

Which of these lists contains at least one word that is not a Java keyword?

a. interface, static, void, catch, final
b. char, strictfp, finally, long, volatile
c. native, super, class, float, while
d. const, for, new, switch, import
e. continue, finalize, goto, package, synchronized
f. None of the above

1 l None of the above All of these are keywords of the Pascal programming language, but none are Java keywords.
2 b d g h strictfp super goto native
3 b e g h k goto new finally const do
4 c e g i const continue extends break All of the letters of all Java keywords are lower case. The word instanceof is a Java keyword, but instanceOf is not.
5 b d f i j k catch instanceof const goto import transient
6 a b c d g i j k byte short int long float double boolean char
7 j None of the above
8 a e transient volatile Serializable, Runnable, Externalizable, and Cloneable are all interfaces. Thread.run is a method. The keywords transient and volatile are field modifiers.
9 b g i goto implements const The words virtual, ifdef, typedef, friend, struct and union are all words related to the C programming language. Although the words const and goto are also related to the C programming language, they are also Java keywords.
10 d import, break, double, exception, throws The word exception is not a Java keyword. The words import, break, double and throws are Java keywords.
11 e continue, finalize, goto, package, synchronized The word finalize is the name of a method of the Object class: It is not a keyword. The words continue, goto, package and synchronized are all Java keywords.

**Initialization**

**Question 1**

```
class GFM11{
  public static void main (String[] args) {
    int x,y,z;
    System.out.println(x+y+z);
}}
```

What is the result of attempting to compile and run the program?

a. Prints nothing.
b. Prints an undefined value.
c. Prints: null
d. Prints: 0
e. Run-time error
f. Compile-time error
g. None of the above

---

**Question 2**

```
class GFM12 {
  static int x;                          // 1
  public static void main(String[] args) {  // 2
    int y;                               // 3
    System.out.println("x="+x);          // 4
    System.out.println("y="+y);          // 5
}}
```

What is the result of attempting to compile and run the program?

a. Compile-time error at line 1.
b. Compile-time error at line 2.
c. Compile-time error at line 3.
d. Compile-time error at line 4.
e. Compile-time error at line 5.
f. Run-time error
g. None of the above

---

**Question 3**

```
class GFM13 {
  static byte a; static short b; static char c;
```

```
  static int d; static long e; static String s;
  public static void main(String[] args) {
    System.out.println(a+b+c+d+e+s);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 00000null
b. Prints: 00000
c. Prints: 0null
d. Prints: 0
e. Prints: null
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 4**

```
class GFM14 {
  static byte a; static short b; static char c;
  static int d; static long e; static String s;
  public static void main(String[] args) {
    System.out.println(a+","+b+","+(int)c+","+d+","+e+","+s);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0,0,0,0,null
b. Prints: 0,0,0,0,0,
c. Prints: 0,0, ,0,0,
d. Compile-time error
e. Run-time error
f. None of the above

---

**Question 5**

```
class GFM15 {
  static int a; static float b; static double c;
  static boolean d; static String s;
  public static void main(String[] args) {
    System.out.println(a+","+b+","+c+","+d+","+s);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0,0,false,null
b. Prints: 0,0,0,false,
c. Prints: 0,0.0,0.0,false,null
d. Prints: 0,0.0,0.0,false,
e. Prints: 0,0.0,0.0,true,null
f. Prints: 0,0.0,0.0,true,
g. Prints: 0,0,0,true,null
h. Prints: 0,0,0,true,
i. Compile-time error
j. Run-time error
k. None of the above

---

**Question 6**

```
class GFM16 {
  static int m1 (int i1, int i2) {
    int i3;
    if (i1 > 0) {i3 = i1 + i2;}
    return i3;
  }
  public static void main(String[] args) {
```

```
   System.out.println(m1(1,2));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0
b. Prints: 1
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 7**

```
class GFM17 {
  int x;                              // 1
  public static void main(String[] args) {  // 2
    int y = 0;                        // 3
    System.out.print(x+",");          // 4
    System.out.print(y);              // 5
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0
b. Compile-time error at line 1.
c. Compile-time error at line 1.
d. Compile-time error at line 2.
e. Compile-time error at line 3.
f. Compile-time error at line 4.
g. Compile-time error at line 5.
h. Run-time error
i. None of the above

1 f  Compile-time error  Variables declared inside of a block or method are called local variables; they are not automatically initialized. The compiler will generate an error as a result of the attempt to access the local variables before a value has been assigned.
2 e  Compile-time error at line 5.  The local variable y has not been initialized so the attempt to access the variable results in a compile-time error.
3 c  Prints: 0null  The numeric sum of variables a, b, c, d and e is zero. The zero is converted to a String and concatenated with s.
4 a  Prints: 0,0,0,0,0,null  The default value of type char is the null character. When it is cast to an int the value is interpreted as zero.
5 c  Prints: 0,0.0,0.0,false,null  Generally speaking, numeric type variables are initialized to zero. Primitive boolean variables are initialized to false. Reference type variables are initialized to null.
6 c  Compile-time error  Local variables are not initialized automatically, and must be initialized explicitly before attempting to access the value. The local variable i3 will not be initialized if i1 is less than or equal to zero; so the result is a compile-time error.
7 f  Compile-time error at line 4.  Anytime a field is accessed from within a static context it is very important to verify that the field is also static. If the field is instead an instance variable then the result is a Compile-time error.

**Range**

**Question 1**

```
class JJF1 {
  public static void main (String args[]) {
    System.out.print(Byte.MIN_VALUE+",");
    System.out.print(Byte.MAX_VALUE);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,255
b. Prints: 0,256
c. Prints: -127,128
d. Prints: -128,127
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 2**

```
class JJF2 {
  public static void main (String args[]) {
    System.out.print(Short.MIN_VALUE+",");
    System.out.print(Short.MAX_VALUE);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: -32767,32768
b. Prints: -32768,32767
c. Prints: 0,65535
d. Prints: 0,65536
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 3**

```
class JJF3 {
  public static void main(String args[]) {
    System.out.print(Integer.toBinaryString(Byte.MAX_VALUE)+",");
    System.out.print(Integer.toOctalString(Byte.MAX_VALUE)+",");
    System.out.print(Integer.toString(Byte.MAX_VALUE)+",");
    System.out.print(Integer.toHexString(Byte.MAX_VALUE));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 1111111,177,127,7f
b. Prints: 11111111,377,256,ff
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 4**

```
class JJF4 {
  public static void main(String args[]) {
    System.out.print(Long.toHexString(Byte.MAX_VALUE)+",");
    System.out.print(Long.toHexString(Character.MAX_VALUE)+",");
    System.out.print(Long.toHexString(Short.MAX_VALUE));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: f,ff,7f
b. Prints: f,ff,ff
c. Prints: 7f,ffff,7fff
d. Prints: ff,ffff,ffff
e. Prints: 7fff,ffffff,7fffff
f. Prints: ffff,ffffff,ffffff
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 5**

```
class JJF5 {
  public static void main(String args[]) {
    System.out.print(Integer.toHexString(Integer.MIN_VALUE)+",");
    System.out.print(Integer.toHexString(Integer.MAX_VALUE));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0000,ffff

b. Prints: 00000000,ffffffff
c. Prints: 7fff,8000
d. Prints: 8000,7fff
e. Prints: 7fffffff,80000000
f. Prints: 80000000,7fffffff
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 6**

Which of the following represent the full range of type char?

a. '\u0000' to '\u7fff'
b. '\u0000' to '\uffff'
c. 0 to 32767
d. 0 to 65535
e. -32768 to 32767
f. -65536 to 65535

---

**Question 7**

Which of the following represent the full range of type char?

a. -0x8000 to 0x7fff
b. 0x0000 to 0xffff
c. -0100000 to 077777
d. 0 to 0177777
e. 0 to 32767
f. 0 to 65535

g. -32768 to 32767

---

**Question 8**

```
class JJF6 {
  char c1 = 0xffff;  // 1
  char c2 = 0xfffff; // 2
  byte b1 = 0xffff;  // 3
  byte b2 = 0x7f;    // 4
  byte b3 = 0xff;    // 5
  byte b4 = -0x80;   // 6
}
```

Compile-time errors are generated at which lines?

a. 1

b. 2

c. 3

d. 4

e. 5

f. 6

1 d Prints: -128,127 A byte is an 8 bit signed value; so the minimum byte value is -(27) and the maximum value is (27 - 1).

2 b Prints: -32768,32767 A short is a 16 bit signed value; so the minimum short value is -(215) and the maximum value is (215 - 1).

3 a Prints: 1111111,177,127,7f A byte is an 8 bit signed value. The left most bit is the sign bit. The sign bit is set to zero for positive numbers and is set to one for negative numbers. The most positive byte value is represented as a sign bit that is set to zero and all of the other bits set to one. The Integer.toBinaryString method does not print leading zeros; so only seven bits are printed. An eight bit binary value is represented as three octal digits. The first of the three digits represents the left most two bits of the binary value. In this case, the left most two bits are zero and one. The second octal digit represents the next three bits of the binary value. The last of the octal digits represents the right most three bits of binary value. Note that the Integer.toOctalString method does not print a leading zero as is required for an octal literal value. An eight bit binary value is represented as two hexadecimal digits. The left hex digit represents the left most four bits of the binary value. The right hex digit represents the right most four bits of the binary value.

4 c Prints: 7f,ffff,7fff A byte is an 8 bit signed value. A char is a 16 bit unsigned value. A short is a 16 bit signed value. The left most bit of a signed value is the sign bit. The sign bit is zero for positive numbers and one for negative numbers. The maximum byte value in hexadecimal format is 7f and in decimal format is 127. The minimum byte value in hexadecimal format is 80 and in decimal format is -128. The byte value of decimal -1 is ff in hexadecimal.

5 f  Prints: 80000000,7fffffff  An int is a 32 bit signed value. The left most bit is the sign bit. The sign bit is zero for positive numbers and one for negative numbers.
6 b  d  '\u0000' to '\uffff'  0 to 65535  A char is a 16 bit unsigned value; so none of the char values are negative and the minimum value is zero. The maximum value is 216 - 1.
7 b  d  f  0x0000 to 0xffff  0 to 0177777  0 to 65535  A char is a 16 bit unsigned value; so none of the char values are negative and the minimum value is zero. The maximum value is 216 - 1.
8 b  c  e  2  3  5  The maximum char value is 0xffff. The maximum byte value is 127 = 0x7f. The hex value 0xff is of type int, and the decimal value is 255. The maximum positive value of type byte is 127. The value 255 is beyond the range of type byte; so 255 can not be assigned to type byte without an explicit cast. The assignment expression b3 = (byte)0xff would assign the value -1 to variable b3.

**Literals**

**Question 1**

```
class MCZ11 {
  public static void main (String[] args) {
    char a = '\c';  // 1
    char b = '\r';  // 2
    char c = '\'';  // 3
    char d = '\b';  // 4
    char e = '\";  // 5
}}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. 5
f. None of the above

---

**Question 2**

```
class MCZ27 {
  public static void main (String[] args) {
    char a = '\f';  // 1
    char b = '\n';  // 2
    char c = '\r';  // 3
    char d = '\l';  // 4
    char e = '\\';  // 5
}}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. 5
f. None of the above

---

**Question 3**

```
class MCZ28 {
  public static void main (String[] args) {
    char a = '\b';  // 1
    char b = '\d';  // 2
    char c = '\f';  // 3
    char d = '\t';  // 4
    char e = '\'';  // 5
}}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3

d. 4

e. 5

f. None of the above

---

**Question 4**

```
class MCZ29 {
  public static void main (String[] args) {
    char a = '\a';  // 1
    char b = '\b';  // 2
    char c = '\f';  // 3
    char d = '\n';  // 4
    char e = '\r';  // 5
}}
```

A compile-time error is generated at which line?

a. 1

b. 2

c. 3

d. 4

e. 5

f. None of the above

---

**Question 5**

```
class MCZ30 {
  public static void main (String[] args) {
    char a = '\t';  // 1
    char b = '\\';  // 2
    char c = '\?';  // 3
    char d = '\"';  // 4
    char e = '\';  // 5
```

}}

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. 5
f. None of the above

---

**Question 6**

```
class MCZ31 {
  public static void main (String[] args) {
    char a = '\t';  // 1
    char b = '\\';  // 2
    char c = '\"';  // 3
    char d = '\';  // 4
    char e = '\?';  // 5
}}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. 5
f. None of the above

---

**Question 7**

```
class MCZ13 {
  public static void main (String[] args) {
    String s = null;
    System.out.print(s);
}}
```

What is the result of attempting to compile and run the program?

a. Prints nothing.
b. Prints: null
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 8**

```
class MCZ15 {
  public static void main (String[] args) {
    float a = 1.1e1f;  // 1
    float b = 1e-1F;   // 2
    float c = .1e1f;   // 3
    double d = .1d;    // 4
    double e = 1D;     // 5
}}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. 5
f. None of the above

---

**Question 9**

```
                    class MCZ16 {
  public static void main (String[] args) {
    float a = 1;          // 1
    float b = 1L;          // 2
    float c = 1F;          // 3
    float d = 1.0;         // 4
}}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. None of the above

---

**Question 10**

```
class MCZ17 {
  public static void main (String[] args) {
    String a = "\n";      // 1
    String b = "\r";      // 2
    String c = "\u000a";  // 3  \u000a = new line
    String d = "\u000d";  // 4  \u000d = return
}}
```

Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3
d. 4

---

**Question 11**

```
class MCZ18 {
  public static void main (String[] args) {
    String a = "abcd";          // 1
    String b = "'\u0041'";      // 2
    String c = "\u0041";        // 3
    String d = "\uD7AF";        // 4
    System.out.print(a+b+c+d);  // 5
}}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. 5
f. None of the above

---

**Question 12**

```
class MCZ20 {
  public static void main (String[] args) {
    // Insert code here.
}}
```

Which of the following lines can be inserted at the specified location without generating a compile-time error?

a. String a = 'a';
b. String b = 'abc';
c. String c = '\u0041';

d. String d = '\uabcd';

e. None of the above

---

**Question 13**

```
class MCZ21 {
  public static void main (String[] args) {
    // Insert code here.
}}
```

Which of the following lines can be inserted at the specified location without generating a compile-time error?

a. char a = a;

b. char b = abc;

c. char c = \u0041;

d. char d = \uabcd;

e. None of the above

---

**Question 14**

```
class MCZ24 {
  public static void main (String[] args) {
    char a = 061;      // 1
    char b = '\61';    // 2
    char c = '\061';   // 3
    char d = 0x0031;   // 4
    char e = '\u0031'; // 5
    System.out.print(""+a+b+c+d+e);
}}
```

A compile-time error is generated at which line?

a. 1

b. 2

c. 3

d. 4

e. 5

f. None of the above

---

**Question 15**

```
class MCZ25 {
  public static void main (String[] args) {
    char a = '\7';   // 1
    char b = '\61';  // 2
    char c = '\062'; // 3
    char d = '\x7';  // 4
    char e = '\x41'; // 5
    System.out.print(""+a+b+c+d+e);
}}
```

Compile-time errors are generated at which lines?

a. 1

b. 2

c. 3

d. 4

e. 5

---

**Question 16**

```
class MCZ19 {
  public static void main (String[] args) {
    String a1 = null;       // 1
    String b1 = 'null';     // 2
    String c1 = "null";     // 3
```

```
    String d1 = "'null'";    // 4
}}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. None of the above

---

**Question 17**

```
class MCZ22 {
  public static void main (String[] args) {
    // Insert code here.
}}
```

Which of the following lines will generate a compile-time error if inserted at the specified location?

a. char a = 0x0041;
b. char b = '\u0041';
c. char c = 0101;
d. char d = -1;
e. char e = (char)-1;
f. None of the above

---

**Question 18**

```
class MCZ23 {
  public static void main (String[] args) {
    // Insert code here.
}}
```

Which of the following lines can be inserted at the specified location without generating a compile-time error?

a. boolean b1 = true;

b. boolean b2 = TRUE;

c. boolean b3 = 'true';

d. boolean b4 = "TRUE";

e. boolean b5 = 0;

f. None of the above

1 a 1 The escape sequences are as follows: '\b' (backspace), '\f' (formfeed), '\n' (newline), '\r' (carriage return), '\t' (horizontal tab), '\\' (backslash), '\"' (double quote), '\'' (single quote). Yes, you must memorize the escape sequences! Just remember "big farms need red tractors".

2 d 4 The escape sequences are as follows: '\b' (backspace), '\f' (formfeed), '\n' (newline), '\r' (carriage return), '\t' (horizontal tab), '\\' (backslash), '\"' (double quote), '\'' (single quote). Yes, you must memorize the escape sequences! Just remember "big farms need red tractors".

3 b 2 The escape sequences are as follows: '\b' (backspace), '\f' (formfeed), '\n' (newline), '\r' (carriage return), '\t' (horizontal tab), '\\' (backslash), '\"' (double quote), '\'' (single quote). Yes, you must memorize the escape sequences! Just remember "big farms need red tractors".

4 a 1 The escape sequences are as follows: '\b' (backspace), '\f' (formfeed), '\n' (newline), '\r' (carriage return), '\t' (horizontal tab), '\\' (backslash), '\"' (double quote), '\'' (single quote). Yes, you must memorize the escape sequences! Just remember "big farms need red tractors".

5 c 3 The escape sequences are as follows: '\b' (backspace), '\f' (formfeed), '\n' (newline), '\r' (carriage return), '\t' (horizontal tab), '\\' (backslash), '\"' (double quote), '\'' (single quote). Yes, you must memorize the escape sequences! Just remember "big farms need red tractors".

6 e 5 The escape sequences are as follows: '\b' (backspace), '\f' (formfeed), '\n' (newline), '\r' (carriage return), '\t' (horizontal tab), '\\' (backslash), '\"' (double quote), '\'' (single quote). Yes, you must memorize the escape sequences! Just remember "big farms need red tractors".

7 b Prints: null The System.out.print method prints the word null if the argument is a String reference that is null.

8 f None of the above Floating-point literals are covered in section 3.10.2 of the JLS. A floating-point literal can begin with either a digit or a decimal point. Optionally, it can have a fractional part, an exponent part and a floating point suffix--f, F, d, or D.

9 d 4 The literal 1.0 is a double and can not be used to initialize a float without an explicit cast.

10 c d 3 4 The compiler interprets \u000a as a line terminator. The escape sequence \n should be used instead. Similarly, \u000d is interpreted as a line terminator. The escape sequence \r should be used instead.

11 f None of the above All of the declarations are legal. String b is a single quote followed by the letter A followed by another single quote. String c is the letter A. String d is the Unicode character that is represented by the hexadecimal value D7AF. String literals are covered in section 3.10.5 of the JLS.

12 e None of the above String literals are declared using double quotes, but all of the declarations here use single quotes.

13 e None of the above Unicode char literals are declared using single quotes, but none of the declarations here use single quotes. The declaration of char b, is also problematic, because it contains more than one char.

14 f None of the above All of the declarations are legal. The first three ( 061, '\61', '\061' ) are declared in octal format. The fourth (0x0031) is declared as a hexadecimal literal. The fifth ('\u0031') is a Unicode escape sequence.

15 d e 4 5 All of the escape sequences used in this question are defined for the C programming language. Those that are not also Java escape sequences result in a compile-time error. Java does not accept the hexadecimal escape sequences of the C programming language. However, Java does accept Unicode escapes (JLS 3.3).
16 b 2 The reference a1 is set to null. String b1 generates a compile-time error, because String literals must be enclosed by double quotes. String c1 is the word null. String d1 is a single quote followed by the word null followed by another single quote. String literals are covered in section 3.10.5 of the JLS.
17 d char d = -1; The assignment of -1 to char d generates a compile-time error, because the primitive char type is unsigned. A negative int can not be assigned to a char without an explicit cast. If the literal value -1 were cast to type char then the result would be \uffff.
18 a boolean b1 = true; There are two primitive boolean values: true and false. Both must be written with lower case letters. Although the C programming language accepts zero as a boolean value, the Java programming language does not.

**java.lang.Math**

**Question 1**

```
class SRC102 {
  public static void main (String[] args) {
    int i1 = Math.round(0.5f);
    int i2 = Math.round(1.5d);
    System.out.print(i1 + "," + i2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,1
b. Prints: 0,2
c. Prints: 1,1
d. Prints: 1,2
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 2**

```
class SRC103 {
```

```
  public static void main (String[] args) {
    int i1 = Math.round(0.5f);
    int i2 = Math.round(1.5f);
    System.out.print(i1 + "," + i2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,1
b. Prints: 0,2
c. Prints: 1,1
d. Prints: 1,2
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 3**

```
class SRC104 {
  public static void main (String[] args) {
    System.out.print(Math.round(Float.NaN));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: NaN
b. Prints: 0.0
c. Prints: 0
d. Compile-time error
e. Run-time error
f. None of the above

---

**Question 4**

```
class SRC105 {
  public static void main(String[] args) {
    double d1 = Math.ceil(0.5);
    double d2 = Math.ceil(1.5);
    System.out.print(d1 + "," + d2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0.0,1.0
b. Prints: 0.0,2.0
c. Prints: 1.0,1.0
d. Prints: 1.0,2.0
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 5**

```
class SRC106 {
  public static void main(String[] args) {
    double d1 = Math.floor(0.5);
    double d2 = Math.floor(1.5);
    System.out.print(d1 + "," + d2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0.0,1.0
b. Prints: 0.0,2.0
c. Prints: 1.0,1.0

d. Prints: 1.0,2.0
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 6**

```
class SRC107 {
  public static void main (String[] args) {
    int a = -1; long b = -2;
    float c = -3.0f; double d = -4.0;
    a = Math.abs(a); b = Math.abs(b);
    c = Math.abs(c); d = Math.abs(d);
    System.out.print(a+b+c+d);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 10.0
b. Compile-time error
c. Run-time error
d. None of the above

---

**Question 7**

```
class SRC109 {
  public static void main (String[] args) {
    short x3 = 0; short x4 = 1;
    short b1 = Math.min(x3,x4); // 1
    int c1 = Math.min(0,1);     // 2
    long d1 = Math.min(0L,+1L); // 3
    System.out.print(b1+","+c1+","+d1);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0,0
b. Compile-time error at 1
c. Compile-time error at 2
d. Compile-time error at 3
e. Run-time error
f. None of the above

---

**Question 8**

```
class SRC110 {
  public static void main (String[] args) {
    byte x1 = 0; byte x2 = 1;
    byte a1 = Math.min(x1,x2);  // 1
    int c1 = Math.min(0,1);     // 2
    long d1 = Math.min(0L,+1L); // 3
    System.out.print(a1+","+c1+","+d1);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0,0
b. Compile-time error at 1
c. Compile-time error at 2
d. Compile-time error at 3
e. Run-time error
f. None of the above

---

**Question 9**

```
class SRC111 {
```

```
  static String m(byte i) {return "byte";}
  static String m(short i) {return "short";}
  static String m(char i) {return "char";}
  static String m(int i) {return "int";}
  static String m(double i) {return "double";}
  public static void main (String[] args) {
    byte b = 0; short s = 0; char c = 0;
    System.out.print(m(Math.min(b,b))+",");
    System.out.print(m(Math.min(s,s))+",");
    System.out.print(m(Math.min(c,c)));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: byte,byte,byte
b. Prints: short,short,short
c. Prints: int,int,int
d. Prints: byte,short,int
e. Prints: short,short,int
f. Prints: byte,short,char
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 10**

```
class SRC112 {
  static String m(byte i) {return "byte";}
  static String m(int i) {return "int";}
  static String m(long i) {return "long";}
  static String m(double i) {return "double";}
  public static void main (String[] args) {
    byte b = 0;
    System.out.print(m(Math.min(b,b))+",");
```

```
    System.out.print(m(Math.min(b,1))+",");
    System.out.print(m(Math.min(b,1L)));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: byte,byte,byte
b. Prints: byte,int,long
c. Prints: int,int,int
d. Prints: int,int,long
e. Prints: long,long,long
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 11**

```
class SRC113 {
  static String m(byte i) {return "byte";}
  static String m(short i) {return "short";}
  static String m(char i) {return "char";}
  static String m(int i) {return "int";}
  static String m(long i) {return "long";}
  static String m(float i) {return "float";}
  static String m(double i) {return "double";}
  public static void main (String[] args) {
    byte b = 0; short s = 0; char c = 0;
    System.out.print(m(Math.min(b,1L))+",");
    System.out.print(m(Math.min(s,1.0f))+",");
    System.out.print(m(Math.min(c,1.0)));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: int,int,int
b. Prints: byte,short,char
c. Prints: long,float,double
d. Prints: double,double,double
e. Prints: long,long,long
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 12**

```
class SRC114 {
  static String m(int i) {return "int";}
  static String m(long i) {return "long";}
  static String m(float i) {return "float";}
  static String m(double i) {return "double";}
  public static void main (String[] args) {
    System.out.print(m(Math.random()));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: int
b. Prints: long
c. Prints: float
d. Prints: double
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 13**

```
class SRC115 {
  static String m(int i) {return "int";}
  static String m(long i) {return "long";}
  static String m(float i) {return "float";}
  static String m(double i) {return "double";}
  public static void main (String[] args) {
    long seed = 1L;
    System.out.print(m(Math.random(seed)));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: int
b. Prints: long
c. Prints: float
d. Prints: double
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 14**

```
class SRC116 {
  static String m(int i) {return "int";}
  static String m(long i) {return "long";}
  static String m(float i) {return "float";}
  static String m(double i) {return "double";}
  public static void main (String[] args) {
    System.out.print(m(Math.sin(0.0f))+",");
    System.out.print(m(Math.cos(0.0f))+",");
    System.out.print(m(Math.tan(0.0f)));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: int,int,int
b. Prints: long,long,long
c. Prints: float,float,float
d. Prints: double,double,double
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 15**

```
class SRC117 {
  public static void main (String[] args) {
    double d1 = Math.random();
    boolean b1 = (d1 < 0.0), b2 = (d1 <= 0.0), b3 = (d1 == 0.0);
    boolean b4 = (d1 >= 0.0), b5 = (d1 < 1.0), b6 = (d1 <= 1.0);
    boolean b7 = (d1 == 1.0), b8 = (d1 >= 1.0), b9 = (d1 > 1.0);
}}
```

Which of the boolean variables will never be initialized to the value true?

a. b1
b. b2
c. b3
d. b4
e. b5
f. b6
g. b7
h. b8
i. b9

---

**Question 16**

Which method of the java.lang.Math class is not static?

a. abs
b. ceil
c. floor
d. max
e. min
f. random
g. round
h. sin
i. cos
j. tan
k. sqrt
l. None of the above

---

**Question 17**

Some of the following method names are overloaded and some are not. Which of the following method names are overloaded such that for each of the following four primitive types int, long, float and double there is at least one corresponding method that returns that type?

a. Abs
b. ceil
c. floor
d. max
e. Min
f. random
g. round
h. sin
i. cos
j. tan
k. sqrt

---

**Question 18**

What is the result of attempting to compile and run the program?

a. Prints: DDDDDD
b. Prints: LLLLLL
c. Prints: DLLLDD
d. Prints: DIDLDD
e. Prints: DLDLDD
f. Prints: DDDLDD
g. Prints: DIDIDD
h. None of the above

---

**Question 19**

```
class SRC119 {
  static String m1(int i) {return "I";}
  static String m1(long i) {return "L";}
  static String m1(float i) {return "F";}
  static String m1(double i) {return "D";}
  public static void main (String[] args) {
    System.out.print(m1(Math.floor(1.0f)) + m1(Math.floor(1.0d)));
    System.out.print(m1(Math.ceil(1.0f)) + m1(Math.ceil(1.0d)));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: IIII
b. Prints: ILIL

c. Prints: LLLL

d. Prints: FDFD

e. Prints: DDDD

f. None of the above

1 e Compile-time error There are two versions of the Math.round method. One accepts an argument of type float; the return type is int. The other accepts an argument of type double; the return type is long. A compile-time error is generated here due to the attempt to assign the long return value to the int variable, i2.

2 d Prints: 1,2 The Math.round method returns the floor of the argument value plus 0.5. If the argument is of type float, then the return type is int. If the argument is of type double, then the return type is long.

3 c Prints: 0 If NaN is the argument passed to Math.round, then the return value is zero. If the argument type is float, then the return type is int. If the argument type is double, then the return type is long.

4 d Prints: 1.0,2.0 The Math.ceil method name is not overloaded. The Math.ceil method accepts an argument of type double and returns a double. The returned value is the smallest whole number that is greater than or equal to the argument.

5 a Prints: 0.0,1.0 The Math.floor method name is not overloaded. The Math.floor method accepts an argument of type double and returns a double. The returned value is the largest whole number that is smaller than or equal to the argument.

6 a Prints: 10.0 The Math class declares four versions of the abs method; each declares one parameter. The parameter can be of type int, long, float or double. The return type is the same as the argument type. At run-time, the argument might not match the parameter type; so the argument might require an implicit conversion to an acceptable type. If the argument is of type byte, short or char, then it will be promoted to type int.

7 b Compile-time error at 1 At line 1, the invocation of the Math.min method produces a return value of type int. The variable b1 is of type short; so the assignment expression results in a compile-time error. The Math class declares four versions of the min method; each declares a pair of parameters of the same type. The parameter pair can be of type int, long, float or double. The return type is the same as the argument types. At run-time, the arguments might not match the declared parameter types; so one argument or both might require an implicit conversion to an acceptable type. If both arguments are of type byte, short or char, then both will be promoted to type int. If only one argument is of type byte, short or char, then it will be promoted to the type of the other argument. If both arguments are of type int, long, float or double but the types differ, then a primitive widening conversion will be applied to one of the two arguments.

8 b Compile-time error at 1 At line 1, the invocation of the Math.min method produces a return value of type int. The variable a1 is of type byte; so the assignment expression results in a compile-time error. The Math class declares four versions of the min method; each declares a pair of parameters of the same type. The parameter pair can be of type int, long, float or double. The return type is the same as the argument types. At run-time, the arguments might not match the declared parameter types; so one argument or both might require an implicit conversion to an acceptable type. If both arguments are of type byte, short or char, then both will be promoted to type int. If only one argument is of type byte, short or char, then it will be promoted to the type of the other argument. If both arguments are of type int, long, float or double but the types differ, then a primitive widening conversion will be applied to one of the two arguments.

9 c Prints: int,int,int The Math class declares four versions of the min method; each declares a pair of parameters of the same type. The parameter pair can be of type int, long, float or double. The return type is the same as the argument types. At run-time, the arguments might not match the declared parameter types; so one argument or both might require an implicit conversion to an acceptable type. If both arguments are of type byte, short or char, then both will be promoted to type int. If only one argument is of type byte, short or char, then it will be promoted to the type of the other argument. If both arguments are of type int, long, float or double but the types differ, then a primitive widening conversion will be applied to one of the two arguments.

10 d Prints: int,int,long The Math class declares four versions of the min method; each declares a pair of parameters of the same type. The parameter pair can be of type int, long, float or double. The return type is the same as the argument types. At run-time, the arguments might not match the declared parameter types; so

one argument or both might require an implicit conversion to an acceptable type. If both arguments are of type byte, short or char, then both will be promoted to type int. If only one argument is of type byte, short or char, then it will be promoted to the type of the other argument. If both arguments are of type int, long, float or double but the types differ, then a primitive widening conversion will be applied to one of the two arguments.
11 c Prints: long,float,double The Math class declares four versions of the min method; each declares a pair of parameters of the same type. The parameter pair can be of type int, long, float or double. The return type is the same as the argument types. At run-time, the arguments might not match the declared parameter types; so one argument or both might require an implicit conversion to an acceptable type. If both arguments are of type byte, short or char, then both will be promoted to type int. If only one argument is of type byte, short or char, then it will be promoted to the type of the other argument. If both arguments are of type int, long, float or double but the types differ, then a primitive widening conversion will be applied to one of the two arguments.
12 d Prints: double The Math.random method returns a value of type double. The range of values is greater than or equal to zero and less than one.
13 e Compile-time error The Math.random method does not accept a seed value. If your application requires a seed, then use java.util.Random.
14 d Prints: double,double,double The Math.sin, Math.cos and Math.tan methods return a value of type double.
15 a g h i b1 b7 b8 b9 The Math.random method returns a value that is equal to or greater than zero and less than 1.0.
16 l None of the above All methods of the java.lang.Math class are static.
17 a d e abs max min
18 f Prints: DDDLDD The round method does not return either of the two floating point primitive types, float or double. The ceil, sin and sqrt methods return only a value of type double. The abs and max methods are able to return an int, long, float or double depending on the argument type.
19 e Prints: DDDD The floor and ceil methods are not overloaded. There is only one version of each method. The parameter type is double, and the return type is double

Question 1

```
class SRC120 {
  static String m1(int i) {return "I";}
  static String m1(long i) {return "L";}
  static String m1(float i) {return "F";}
  static String m1(double i) {return "D";}
  public static void main (String[] args) {
    System.out.print(m1(Math.abs(1.0f)) + m1(Math.abs(1.0d)));
    System.out.print(m1(Math.sqrt(1.0f)) + m1(Math.sqrt(1.0d)));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: IIII
b. Prints: ILIL
c. Prints: LLLL
d. Prints: FDFD
e. Prints: FDDD

f. Prints: DDFD
g. Prints: DDDD
h. None of the above

---

Question 2

```
class SRC121 {
  static String m1(byte i) {return "B";}
  static String m1(char i) {return "C";}
  static String m1(int i) {return "I";}
  static String m1(long i) {return "L";}
  static String m1(float i) {return "F";}
  static String m1(double i) {return "D";}
  public static void main (String[] args) {
    System.out.print(m1(Math.min((byte)1,(byte)2)));
    System.out.print(m1(Math.min('A','B')));
    System.out.print(m1(Math.min(1,2)));
    System.out.print(m1(Math.min((long)1,2)));
    System.out.print(m1(Math.min(1.0f,1)));
    System.out.print(m1(Math.min(1.0,1)));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: DDDDDD
b. Prints: FFFFFD
c. Prints: FFFDFD
d. Prints: IIILFD
e. Prints: IILLDD
f. Prints: IIILDD
g. Prints: BCILFD
h. Prints: CCILFD
i. None of the above

---

Question 3

```
class SRC122 {
 static String m1(int i) {return "I";}
 static String m1(long i) {return "L";}
 static String m1(float i) {return "F";}
 static String m1(double i) {return "D";}
 public static void main (String[] args) {
   System.out.print(m1(Math.abs(1.0f)) + m1(Math.abs(1.0)));
   System.out.print(m1(Math.round(1.0f)) + m1(Math.round(1.0)));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: IIII
b. Prints: LLLL
c. Prints: FFFF
d. Prints: DDDD
e. Prints: FDFD
f. Prints: ILIL
g. Prints: FDIL
h. Prints: ILFD
i. None of the above

---

Question 4

Which of the following methods of the java.lang.Math class accepts an argument of type float or double, but has a return type of type int or long respectively.

a. abs
b. ceil
c. floor
d. max
e. min
f. random
g. round
h. sin
i. cos

j. tan
k. sqrt
l. None of the above

---

Question 5
Which of the following methods of the java.lang.Math class declares a non-primitive argument type?

a. abs
b. ceil
c. floor
d. max
e. min
f. random
g. round
h. sin
i. cos
j. tan
k. sqrt
l. None of the above

---

Question 6
Which of the following methods of the java.lang.Math class declares a non-primitive return type?

a. abs
b. ceil
c. floor
d. max
e. min
f. random
g. round
h. sin
i. cos
j. tan

k. sqrt
l. None of the above

---

Question 7
Which of the following statements is true in terms of the java.lang.Math.round method?

a. The returned value is of a floating-point primitive type.
b. The returned value is always of type int.
c. The returned value is always of type long.
d. The returned value is of type long only if the argument is of type long.
e. The returned value is of type long only if the argument is of type double.
f. None of the above

---

Question 8
Which of the following statements is true in terms of the value returned by the java.lang.Math.round method?

a. Rounds the argument to the nearest whole number. If the result is equally close to two whole numbers, then the even one is returned.
b. Rounds the argument to the nearest whole number. If the result is equally close to two whole numbers, then the odd one is returned.
c. Adds 0.5 to the argument and takes the floor of the result.
d. Adds 0.5 to the argument and takes the ceil of the result.
e. None of the above

---

Question 9
Suppose that the java.lang.Math.round method is invoked with an argument of type float, and the result exceeds the range of type int. Which of the following occurs?

a. The result is implicitly widened to type long.
b. The returned value is Float.NaN.
c. A run-time exception is thrown.
d. The bits that overflow are ignored and no exception is thrown.
e. The returned value is zero but no exception is thrown.

f. None of the above

---

Question 10
Which of the following statements is true in terms of the java.lang.Math.sqrt method?

a. It is not overloaded.
b. Returns a float if the argument type is float.
c. If the argument is negative, then the returned value is the square root of the absolute value of the argument.
d. Throws an ArithmeticException if the argument is negative.
e. None of the above

---

Question 11
Which of the following statements are true in terms of the java.lang.Math.sin method?

a. It is not overloaded.
b. Returns a float if the argument type is float.
c. The argument is an angle measured in radians.
d. The argument is an angle measured in degrees.
e. Throws an ArithmeticException if the argument is greater than 360.

---

Question 12
Which of the following statements is true in terms of the java.lang.Math.random method?

a. The random method name is overloaded.
b. The argument type is a double that represents a seed value.
c. Throws an ArithmeticException if the seed value is negative.
d. The returned value is always greater than zero and less than or equal to one.
e. None of the above

---

Question 13
Which of the following statements is true in terms of the java.lang.Math.floor method?

a. Four overloaded versions of floor exist.
b. An ArithmeticException is declared in the throws clause.
c. The type of the return value depends on the argument type.
d. The returned value is always of an integral primitive type.
e. Returns the largest whole number that is less than or equal to the argument value.
f. Returns the smallest whole number that is greater than or equal to the argument value.
g. None of the above

---

Question 14
Which of the following statements is true in terms of the java.lang.Math.ceil method?

a. Four overloaded versions of ceil exist.
b. An ArithmeticException is declared in the throws clause.
c. The type of the return value depends on the argument type.
d. The returned value is always of an integral primitive type.
e. Returns the largest whole number that is less than or equal to the argument value.
f. Returns the smallest whole number that is greater than or equal to the argument value.
g. None of the above

---

Question 15
Which of the following statements are true in terms of the java.lang.Math.abs method?

a. Four overloaded versions of abs exist.
b. An ArithmeticException is declared in the throws clause.
c. The type of the return value depends on the argument type.
d. The returned value is always of a floating-point primitive type.
e. If the argument is a negative integral value, then the returned value is always positive.
f. If the argument is positive, then the argument is returned.

---

Question 16
The java.lang.Math.abs method can return which of the following?

a. Negative infinity
b. Positive infinity
c. NaN
d. Short.MIN_VALUE
e. Integer.MIN_VALUE
f. Long.MIN_VALUE
g. Negative zero
h. Positive zero

---

Question 17

```
class SRC123 {
  public static void main (String[] args) {
    System.out.print(Math.floor(-3.6) + "," + Math.ceil(-3.6)+ ",");
    System.out.print(Math.floor(3.6) + "," + Math.ceil(3.6));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: -3.0,-4.0,3.0,4.0
b. Prints: -3.0,-4.0,4.0,3.0
c. Prints: -4.0,-3.0,3.0,4.0
d. Prints: -4.0,-3.0,4.0,3.0
e. Prints: -4.0,-4.0,3.0,3.0
f. Compile-time error
g. Run-time error
h. None of the above

---

Question 18

```
class SRC124 {
```

```
  public static void main (String[] args) {
    System.out.print(Math.round(-3.6) + "," + Math.round(-3.4)+ ",");
    System.out.print(Math.round(3.4) + "," + Math.round(3.6));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: -3.0,-4.0,3.0,4.0
b. Prints: -3.0,-4.0,4.0,3.0
c. Prints: -4.0,-3.0,3.0,4.0
d. Prints: -4.0,-3.0,4.0,3.0
e. Prints: -4.0,-4.0,3.0,3.0
f. Compile-time error
g. Run-time error
h. None of the above

---

Question 19

```
class SRC125 {
  public static void main (String[] args) {
    System.out.print(Math.round(-3.6) + Math.round(3.6) + ",");
    System.out.print(Math.round(-3.4) + Math.round(3.4));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0
b. Prints: 0,-1
c. Prints: -1,0
d. Prints: -1,-1
e. Prints: 0,1
f. Prints: 1,0
g. Prints: 1,1
h. Compile-time error
i. Run-time error
j. None of the above

1 e Prints: FDDD The method name abs is overloaded. There are versions that accept one argument of type int, long, float or double. The type of the return value is the same as the argument type. The sqrt method is not overloaded. The parameter type is double and the return type is double.

2 d Prints: IIILFD There are four overloaded versions of the Math.min method with versions that declare one parameter of type int, long, float or double. The return type is the same as the parameter type. Arguments of type byte, char and short are promoted to type int.

3 g Prints: FDIL The method name abs is overloaded. There are versions that accept one argument of type int, long, float or double. The type of the return value is the same as the argument type. There are two versions of the Math.round method. One accepts an argument of type float; the return type is int. The other accepts an argument of type double; the return type is long.

4 g round

5 l None of the above

6 l None of the above

7 e The returned value is of type long only if the argument is of type double.

8 c Adds 0.5 to the argument and takes the floor of the result.

9 f None of the above If the java.lang.Math.round method is invoked with an argument of the float type and the result exceeds the range of type int, then the result is Integer.MAX_VALUE.

10 a It is not overloaded. The argument and returned value are both of type double. If the argument is negative, then the returned value is NaN.

11 a c It is not overloaded. The argument is an angle measured in radians. The argument is of type double and represents an angle measured in radians. The returned value is of type double. The method does not throw any exceptions.

12 e None of the above The random method is not overloaded, does not declare any parameters and does not throw any exceptions. The returned value is greater than or equal to zero and is less than but not equal to one.

13 e Returns the largest whole number that is less than or equal to the argument value. The floor method is not overloaded, and declares only one parameter of type double. The return value is always of type double. The floor method does not declare any exceptions.

14 f Returns the smallest whole number that is greater than or equal to the argument value. The ceil method is not overloaded, and declares only one parameter of type double. The return value is always of type double. The ceil method does not declare any exceptions.

15 a c f Four overloaded versions of abs exist. The type of the return value depends on the argument type. If the argument is positive, then the argument is returned. The method name abs is overloaded. There are versions that accept one argument of type int, long, float or double. The type of the return value is the same as the argument type. No exceptions are declared. If the argument is a negative integral value, then the returned value is the two's complement of the argument. The magnitude of Integer.MIN_VALUE is one greater than the magnitude of Integer.MAX_VALUE; therefore, the absolute value of Integer.MIN_VALUE exceeds the range of Integer.MAX_VALUE. Due to the limited range of type int, the two's complement of Integer.MIN_VALUE is Integer.MIN_VALUE. For that reason, the Math.abs method returns Integer.MIN_VALUE when the argument is Integer.MIN_VALUE.

16 b c e f h Positive infinity NaN Integer.MIN_VALUE Long.MIN_VALUE Positive zero The method name abs is overloaded. There are versions that accept one argument of type int, long, float or double. The type of the return value is the same as the argument type. The result is positive infinity if the argument is negative infinity. The result is positive zero if the argument is negative zero. The result is NaN if the argument is NaN. If the argument is a negative integral value, then the returned value is the two's complement of the argument. The magnitude of Integer.MIN_VALUE is one greater than the magnitude of Integer.MAX_VALUE; therefore, the absolute value of Integer.MIN_VALUE exceeds the range of Integer.MAX_VALUE. Due to the limited range of type int, the two's complement of Integer.MIN_VALUE is Integer.MIN_VALUE. For that reason, the Math.abs method returns Integer.MIN_VALUE when the argument is Integer.MIN_VALUE. The negation of Short.MIN_VALUE is well within the range of type int; so Short.MIN_VALUE is never returned by the Math.abs method.

17 c Prints: -4.0,-3.0,3.0,4.0 The Math.floor method returns a primitive of type double that is equal to the largest integral value that is smaller than or equal to the argument. For example, -4.0 is the floor of -3.6, because -4.0 is the largest floating-point value that is equal to an integral value that is closer to negative infinity than -3.6. Similarly, the Math.ceil method returns a primitive of type double that is equal to an integral value, and is smaller than any other such value that is larger than the argument. For example, -3.0 is the ceil of -3.6. The term "larger" refers to valuess that are closer to positive infinity. For example, -3 is larger than -4, because -3 is closer to positive infinity.
18 h None of the above The math.round method name is overloaded. If the argument is of type double, then the return value is of type long. If the argument is of type float, then the return value is of type int. Both return types are integral primitive types; so a decimal point should not be included in the output. The actual output is -4,-3,3,4.
19 a Prints: 0,0 The math.round method adds 0.5 to the argument, and then takes the floor of the sum. The result of Math.round with an argument of -3.6 is -4. The result of Math.round with an argument of 3.6 is 4.0. The sum of the two results is zero.

**Strings**

**Question 1**

Which of the following are not methods of the java.lang.String class?

a. Append
b. Concat
c. Delete
d. Insert
e. Replace
f. Substring
g. ValueOf

---

**Question 2**

Which of these methods modify the contents of the String instance on which it is invoked?

a. Append
b. Concat
c. Delete
d. Insert

e. Replace
f. substring
g. valueOf
h. None of the above.

---

**Question 3**

Which of these methods is a static member of the java.lang.String class?

a. append
b. concat
c. delete
d. insert
e. replace
f. substring
g. ValueOf
h. None of the above.

---

**Question 4**

```
class MWC106 {
  static void m1(String s) {
    s = s.trim(); s = s.concat("D");
  }
  public static void main(String[] s) {
    String s1 = "A", s2 = " B ", s3 = "C";
    m1(s2);
    System.out.print(s1 + s2 + s3);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: ABC
b. Prints: A B C
c. Prints: ABCD
d. Prints: ABDC
e. Prints: A B CD
f. Prints: A B DC
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 5**

```
class MWC107 {
  public static void main(String[] s) {
    String s1 = "A", s2 = " B ", s3 = "C";
    s2.trim(); s3.concat("D");
    System.out.print(s1 + s2 + s3);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: ABC
b. Prints: A B C
c. Prints: ABCD
d. Prints: ABDC
e. Prints: A B CD
f. Prints: A B DC
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 6**

```
class MWC108 {
  public static void main(String[] s) {
    String s1 = "A", s2 = " B ", s3 = "C";
    s2.trim(); s3.append("D");
    System.out.print(s1 + s2 + s3);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: ABC
b. Prints: A B C
c. Prints: ABCD
d. Prints: ABDC
e. Prints: A B CD
f. Prints: A B DC
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 7**

```
class MWC110 {
  static void m1(String s1, String s2) {
    s1.insert(1,"D"); s1 = s1 + s2;
  }
  public static void main(String[] s) {
    String s1 = "A", s2 = "B";
    m1(s1,s2);
    System.out.print(s1 + s2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AB
b. Prints: ABB
c. Prints: ADB
d. Prints: ADBB
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 8**

```
class MWC111 {
  static void m1(String s1) {
    s1.replace('A','Y'); System.out.print(s1);
  }
  static void m2(String s1) {
    s1 = s1.replace('A','Z'); System.out.print(s1);
  }
  public static void main(String[] s) {
    String s1 = "A"; m1(s1); m2(s1);
    System.out.print(s1);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AAA
b. Prints: YZA
c. Prints: YAA
d. Prints: AZA
e. Prints: AZZ
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 9**

```
class MWC112 {
  static void m1(String s1) {
    s1 = s1.toUpperCase(); System.out.print(s1);
  }
  static void m2(String s1) {
    s1.toLowerCase(); System.out.print(s1);
  }
  public static void main(String[] s) {
    String s1 = "Ab";
    m1(s1); m2(s1);
    System.out.print(s1);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AbAbAb
b. Prints: ABabab
c. Prints: ABabAb
d. Prints: ABAbAb
e. Prints: Ababab
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 10**

```
class MWC113 {
  public static void main(String[] s) {
    String s1 = "1", s2 = "2", s3 = s1 + s2;
    s1 += s2; s3 += s1;
    System.out.println(s3);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 3
b. Prints: 6
c. Prints: 33
d. Prints: 1212
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 11**

```
class MWC114 {
  public static void main(String[] s) {
    String s1 = new String("ABCDEFG"), s2 = new String("EFGHIJ");
    String s3 = s1.substring(4,7), s4 = s2.substring(0,3);
    System.out.println((s3 == s4) + "," + (s3 + s4).equals(s4 + s3));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false
d. Prints: true,true
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 12**

```
class MWC115 {
```

```
  public static void main(String[] s) {
    String s1 = new String("ABCDEFG"), s2 = s1.substring(0,3);
    String s3 = s1.substring(4,6); char c1 = s1.charAt(3);
    System.out.println(s2.concat(String.valueOf(c1)).concat(s3));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: "ABCDEFG"
b. Prints: "ABCEFG"
c. Prints: "ABCDDEFG"
d. Prints: "ABCDEF"
e. Prints: "ABCEF"
f. Prints: "ABCDDEF"
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 13**

```
class MWC118 {
  public static void main(String[] args) {
    byte[] b = {'a', 'b', 'c'};
    char[] c = {'a', 'b', 'c'};
    String s = "abc";
    StringBuffer sb = new StringBuffer("abc");
    byte b2 = 'a';
    char c2 = 'a';
    String s1 = new String(b);   // 1
    String s2 = new String(c);   // 2
    String s3 = new String(s);   // 3
    String s4 = new String(sb);  // 4
    String s5 = new String(b2);  // 5
    String s6 = new String(c2);  // 6
```

}}

Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3
d. 4
e. 5
f. 6

---

**Question 14**

```
class MWC101 {
  public static void main(String[] args) {
    String s1 = "A", s2 = "a", s3 = "b";
    s1.toLowerCase(); s3.replace('b','a');
    System.out.print((s1.equals(s2)) + "," + (s2.equals(s3)));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false
d. Prints: true,true
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 15**

```
class MWC102 {
  public static void main (String[] args) {
    String s1 = "ABCDE";
    System.out.print(s1.substring(1,2)+s1.substring(3));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AABC
b. Prints: ACDE
c. Prints: ABABC
d. Prints: ABCDE
e. Prints: BABCD
f. Prints: BDE
g. Prints: BCABCD
h. Prints: BCDE
i. Compile-time error
j. Run-time error
k. None of the above

---

**Question 16**

```
class MWC104 {
  public static void main (String[] args) {
    char[] c = {'a','b','c','d'};
    String s1 = new String(c);
    boolean b = true;
    for (int i = 0; i < s1.length; i++) {
      b &= (c[i] == s1.charAt(i));
    }
    System.out.print(b);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false
b. Prints: true
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 17**

```
class MWC109 {
  public static void main(String args[]) {
    String a = "A", b = "B", c = a+b, d = a+b;
    System.out.print(((a+b)==(a+b)) + ",");
    System.out.print((c==d) + ",");
    System.out.print(c.equals(d));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false
b. Prints: false,false,true
c. Prints: false,true,false
d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true
g. Prints: true,true,false
h. Prints: true,true,true

---

**Question 18**

```
class MWC116 {
  public static void main(String[] args) {
    String s1 = " B C D E ";
```

```
   System.out.print('A' + s1.trim() + 'F');
}}
```

What is the result of attempting to compile and run the program?

a. Prints: ABCDEF
b. Prints: A B C D E F
c. Prints: A BCDEF
d. Prints: ABCDE F
e. Prints: AB C D EF
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 19**

```
class MWC117 {
  public static void main (String[] args) {
    System.out.print(String.valueOf(1) + String.valueOf(2));
    String s1 = "S1";
    String s2 = s1.toString();
    System.out.print("," + (s1==s2));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 3,false
b. Prints: 3,true
c. Prints: 12,false
d. Prints: 12,true
e. Compile-time error
f. Run-time error
g. None of the above

1.a c d .append delete insert .The StringBuffer class has methods named append, delete and insert, but the String class does not. A typical trick question will attempt to invoke StringBuffer methods on a String instance.

2.h .None of the above. .Strings are immutable. No method of the of a String class will change the contents of a String instance. Some String methods such as concat, replace, substring and trim will return a new String with the desired modifications. The StringBuffer methods append, delete and insert are not members of the java.lang.String class. A typical trick question will attempt to invoke StringBuffer methods on a String instance.

3.g .valueOf .The valueOf method is static. It returns a String representation of the argument. The StringBuffer methods append, delete and insert are not members of the java.lang.String class. A typical trick question will attempt to invoke StringBuffer methods on a String instance.

4.b .Prints: A B C .The String instance referenced by s2 is passed to the m1 method by passing the value of the reference. The reference value used in method m1 is a local copy of the reference. If the local copy used in method m1 is changed, then the original reference variable in the main method remains unchanged.

5.b .Prints: A B C .Strings are immutable. No method will change the contents of a String instance. Some String methods such as concat, replace, substring and trim will return a new String instance with the desired modifications. In this case, the String instances returned by the methods trim and concat are ignored.

6.g .Compile-time error .The StringBuffer class has an append method, but the String class does not. A compile-time error is generated due to the attempt to invoke the append method on an instance of type String.

7.e .Compile-time error .The StringBuffer class has an insert method, but the String class does not. A compile-time error is generated due to the attempt to invoke the insert method on an instance of type String.

8.d .Prints: AZA .Instances of type String are immutable. In method m1, the replace method returns a new instance of type String that contains the value Y, but the String instance referenced by s1 remains unchanged. The original value, A, is printed in method m1. In method m2, the replace method returns a new instance of type String that contains the value Z, and a reference to the new instance is assigned to reference variable s1. The new value, Z, is printed in method m2. In the main method, a copy of the reference value contained by the reference variable s1 is passed as an argument to methods m1 and m2. Since String instances are immutable, methods m1 and m2 can not change the original String instance that is declared in the main method. Since references are passed by value, methods m1 and m2 can not change the reference variable declared in the main method. Regardless of anything that happens in methods m1 and m2, the reference variable s1 that is declared in the main method will continue to reference the original String instance that contains the value A.

9.d .Prints: ABAbAb .Instances of type String are immutable. In method m1, the toUpperCase method returns a new instance of type String that contains the value AB, and a reference to the new instance is assigned to reference variable s1. The new value, AB, is printed in method m1. In method m2, the toLowerCase method returns a new instance of type String that contains the value ab, but the String instance referenced by s1 remains unchanged. The original value, Ab, is printed in method m2. In the main method, a copy of the reference value contained by the reference variable s1 is passed as an argument to methods m1 and m2. Since String instances are immutable, methods m1 and m2 can not change the original String instance that is declared in the main method. Since references are passed by value, methods m1 and m2 can not change the reference variable declared in the main method. Regardless of anything that happens in methods m1 and m2, the reference variable s1 that is declared in the main method will continue to reference the original String instance that contains the value Ab.

10.d .Prints: 1212 .The reference variable s3 is initialized with a reference to an instance of type String containing the value "12". The expression s1 += s2 is equivalent to the expression s1 = s1 + s2. Further simplification produces s1 = "1" + "2" = "12". The expression s3 += s1 is equivalent to the expression s3 = "12" + "12" = "1212".

11.b .Prints: false,true .The reference variable s1 is initialized with a reference to an instance of type String containing the value "ABCDEFG". The reference variable s2 is initialized with a reference to an instance of type String containing the value "EFGHIJ". The expression s3 = s1.substring(4,7) initializes the

reference variable s3 with a reference to a unique instance of type String containing the value "EFG". The expression s4 = s2.substring(0,3) initializes the reference variable s4 with a reference to a unique instance of type String containing the value "EFG". The expression s3 == s4 compares two unique reference values and produces the value false even though s3 and s4 reference two String instances that contain the same value, "EFG". The expression s3 + s4 produces a unique instance of type String containing the value "EFGEFG". Similarly, the expression s4 + s3 produces a unique instance of type String containing the value "EFGEFG". The expression (s3 + s4).equals(s4 + s3) compares the contents of two unique instances of type String that contain the value "EFGEFG". The result of the comparison is true.

12.d .Prints: "ABCDEF" .The reference variable s1 is initialized with a reference to an instance of type String containing the value "ABCDEFG". The expression s2 = s1.substring(0,3) initializes the reference variable s2 with a reference to a unique instance of type String containing the value "ABC". The expression s3 = s1.substring(4,6) creates a unique instance of type String containing the value "EF". The expression c1 = s1.charAt(3) initializes the primitive variable c1 with the value 'D'. The expression String.valueOf(c1) invokes the static valueOf method with an argument of type primitive char and value 'D'. The valueOf method creates a new instance of type String. The value contained by the new instance is "D". The expression s2.concat(String.valueOf(c1)) invokes the concat method on the instance of type String referenced by the variable s2. The instance referenced by s2 contains the value "ABC". The value contained by the argument is "D". The result of the concatenation operation is a new instance of type String containing the value "ABCD". The expression s2.concat(String.valueOf(c1)).concat(s3) invokes the concat method on the previously created instance containing the value "ABCD". The instance referenced by the argument s3 contains the value "EF". The result of the concatenation operation is a new instance of type String containing the value "ABCDEF".

13.e f .5 6 .The overloaded contructors for the String class accept a parameter of type String or StringBuffer or an array of byte or char. There is no constructor that will accept a primitive byte or char. Please note that if you want to convert a primitive to a String then you can use the static String.valueOf method.

14.a .Prints: false,false .String instances are immutable. String methods such as String.toLowerCase and String.replace create and return a new String instance. The instance on which the method has been invoked remains unchanged. In the program, the equals method is invoked on the original instances of s1 and s2--not the new instances.

15.f .Prints: BDE .The substring method returns a new String instance that is a substring of the original String instance. The single parameter form of the substring method creates a new String that begins at the index specified by the argument value. The two parameter form creates a substring that starts at the index of the first parameter and ends at the index of the second parameter. The character at the start index is included in the substring, but the character at the end index is not.

16.c .Compile-time error .A compile-time error is generated due to the attempt to access the length method of the String class as though it were a variable.

17.b .Prints: false,false,true .At run-time, the expression a+b is evaluated four times. Each evaluation produces a new String instance containing the value "AB". Each of the four instances has a unique reference. If any two of the four instances appear as the operands of the equality operator, then the result is always false. The left and right operands of the equality expression (a+b)==(a+b) reference unique String instances. Since the two references are not the same, the equality expression produces the value false. Similarly, the expression c==d produces the value false. Since the contents of the String instances referenced by c and d are the same, the method invocation expression c.equals(d) produces the value true.

18.e .Prints: AB C D EF .The trim method creates a new String instance with the leading and trailing white space removed.

19.d .Prints: 12,true .The valueOf method returns a String representation of the input parameter. The input parameter may be of type Object, char[], or any primitive type. The toString method returns a reference to the existing String instance. It does not create a new instance of the String.

**Stringbuffer**

**Question 1**

```
class MWC204 {
  static void m1(StringBuffer s1) {
    s1 = s1.append("B"); System.out.print(s1);
  }
  static void m2(StringBuffer s1) {
    s1.append("C"); System.out.print(s1);
  }
  public static void main(String[] s) {
    StringBuffer s1 = new StringBuffer("A");
    m1(s1); m2(s1);
    System.out.print(s1);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AAA
b. Prints: ABAA
c. Prints: ABACA
d. Prints: ABABAB
e. Prints: ABABCAB
f. Prints: ABABCABC
g. Prints: ABCABCABC
h. Compile-time error
i. Run-time error

j. None of the above

---

**Question 2**

```
class MWC205 {
  static void m1(StringBuffer s1) {
    s1.append("B"); System.out.print(s1);
  }
  static void m2(StringBuffer s1) {
    s1 = new StringBuffer("C"); System.out.print(s1);
  }
  public static void main(String[] s) {
    StringBuffer s1 = new StringBuffer("A");
    m1(s1); m2(s1);
    System.out.print(s1);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AAA
b. Prints: ABCA
c. Prints: ABCAB
d. Prints: ABCABC
e. Prints: ABCAC
f. Prints: ABABCABC
g. Compile-time error
h. Run-time error
i. None of the above

---

**Question 3**

```
class MWC200 {
  public static void main (String[] args) {
```

```
   String s1 = "ABC";
   StringBuffer s2 = new StringBuffer(s1);
   System.out.print(s2.equals(s1) + "," + s1.equals(s2));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false
d. Prints: true,true
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 4**

```
class MWC201 {
  public static void main (String[] args) {
    String s1 = new String("ABC"), s2 = new String("ABC");
    StringBuffer sb1 = new StringBuffer(s1);
    StringBuffer sb2 = new StringBuffer(s2);
    System.out.print(s1.equals(s2) + "," + sb1.equals(sb2));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false
d. Prints: true,true
e. Compile-time error
f. Run-time error

g. None of the above

---

**Question 5**

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false
d. Prints: true,true
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 6**

```
class MWC203 {
  public static void main (String[] args) {
    String s1 = new String("ABC"), s2 = new String("ABC");
    StringBuffer sb1 = new StringBuffer(s1);
    StringBuffer sb2 = new StringBuffer(s2);
    boolean b1 = s1.hashCode() == s2.hashCode();
    boolean b2 = sb1.hashCode() == sb2.hashCode();
    System.out.print(b1 + "," + b2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true

c. Prints: true,false

d. Prints: true,true

e. Compile-time error

f. Run-time error

g. None of the above

1 f Prints: ABABCABC Instances of type StringBuffer are not immutable. In method m1, the method invocation expression s1.append("B") appends the String literal "B" to the StringBuffer instance referenced by variable s1. The append method returns a reference to the same StringBuffer instance on which it is invoked; so the assignment expression s1 = s1.append("B") does not assign a different reference value to variable s1. The new value, AB, is printed in method m1. In method m2, the method invocation expression s1.append("C") appends the String literal "C" to the StringBuffer instance referenced by variable s1. The new value, ABC, is printed in method m2. In the main method, a copy of the reference value contained by the reference variable s1 is passed as an argument to methods m1 and m2. Since StringBuffer instances are not immutable, methods m1 and m2 are able to change the original StringBuffer instance that is declared in the main method. The new value, ABC, is printed in the main method.

2 c Prints: ABCAB Instances of type StringBuffer are not immutable. In method m1, the method invocation expression s1.append("B") appends the String literal "B" to the StringBuffer instance referenced by the parameter variable s1. The new value, AB, is printed in method m1. The reference variable s1 declared in the main method refers to the same modified StringBuffer instance. In method m2, the class instance creation expression new StringBuffer("C") creates a new instance of type StringBuffer containing the value C. The assignment expression s1 = new StringBuffer("C") assigns a reference to the new StringBuffer instance to the method parameter variable s1. The value C is printed in method m1. The method local reference variable s1 in the main method remains unchanged by the assignment expression contained in method m2. In the main method, a copy of the reference value contained by the reference variable s1 is passed as an argument to methods m1 and m2. Since StringBuffer instances are not immutable, method m1 is able to change the original instance of the StringBuffer declared in the main method. Since references are passed by value, method m2 can not change the reference variable declared in the main method. Regardless of anything that happens in method m2, the reference variable s1 that is declared in the main method will continue to reference the original StringBuffer instance. Since the content of the original instance was modified by method m1, the new value, AB, is printed in the main method.

3 a Prints: false,false StringBuffer.equals does not override the Object.equals method; so StringBuffer.equals just compares reference values. Since the reference variables s1 and s2 refer to different objects, the equals method of the StringBuffer instance s2 returns the value false. The String.equals method does override the Object.equals, method. The String.equals method returns false anytime the argument is not an instance of type String. The method invocation expression s1.equals(s2) produces the value false, because the argument is an instance of type StringBuffer.

4 c Prints: true,false String.equals overrides Object.equals. The String.equals method compares the contents of the String instances--not the references. Since the contents of s1 and s2 are the same, the method invocation expression s1.equals(s2) produces the value true. The StringBuffer.equals method does not override Object.equals. The StringBuffer.equals method compares the reference values--not the contents of the StringBuffer instances. Since the reference values sb1 and sb2 are different, the method invocation expression sb1.equals(sb2) produces the value false.

5 a Prints: false,false StringBuffer.equals does not override Object.equals. The StringBuffer.equals method compares the reference values--not the contents of the StringBuffer instances. The expressions sb1==sb2 and sb1.equals(sb2) produce the same results.

6 c Prints: true,false The StringBuffer class does not override the equals and hashCode methods of the Object class. The Object.equals method does not return the value true unless the argument is a reference to the same object on which the method is invoked. For example, the method invocation expression obj1.equals(obj2) only produces the value true when obj1 == obj2 is also true. The Object.hashCode method tends to return distinct hashcode values for distinct objects regardless of the internal contents of the object. Suppose that the reference variables sb1 and sb2 are of type StringBuffer. The expression sb1.hashCode()

== sb2.hashCode() will not produce the value true unless the expression sb1 == sb2 is also true. The String class does override the equals and hashCode methods of the Object class. The String.hashCode method returns a hashcode value that is computed based on the contents of the String object. Suppose that the reference variables s1 and s2 are of type String. The expression s1.hashCode() == s2.hashCode() must produce the value true anytime the method invocation expression s1.equals(s2) produces the value true.

**Wrapper**

**Question 1**

```
class A {
  public static void main (String args[]) {
    byte primitiveByte = 1;           // 1
    Byte b1 = new Byte(primitiveByte); // 2
    Byte b2 = new Byte(1);            // 3
    System.out.print(b1.byteValue() + b2.byteValue());
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 2
b. Prints: 11
c. Compile-time error at 1
d. Compile-time error at 2
e. Compile-time error at 3
f. Run-time error
g. None of the above

---

**Question 2**

```
class A {
  public static void main (String args[]) {
    byte primitiveByte = 1;
    char primitiveChar = 'b'-'a';
```

```
   int primitiveInt = 1;
   short primitiveShort = 1;
   String s = "1";
   Integer i1 = new Integer(primitiveByte);
   Integer i2 = new Integer(primitiveChar);
   Integer i3 = new Integer(primitiveShort);
   Integer i4 = new Integer(primitiveInt);
   Integer i5 = new Integer(s);
   int p1 = i1.intValue() + i2.intValue() +
         i3.intValue() + i4.intValue() +
         i5.intValue();
   System.out.print(p1);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 5
b. Prints: 5.0
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 3**

```
class A {
  public static void main (String args[]) {
    byte primitiveByte = 1;
    char primitiveChar = 1;
    double primitiveDouble = 1;
    String s = "1";
    Double d1 = new Double(primitiveByte);
    Double d2 = new Double(primitiveChar);
    Double d3 = new Double(primitiveDouble);
    Double d4 = new Double(s);
    double d5 = d1.doubleValue() + d2.doubleValue() +
```

```
            d3.doubleValue() + d4.doubleValue();
    System.out.print(d5);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 4
b. Prints: 4.0
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 4**

```
class A {
  public static void main (String args[]) {
    byte primitiveByte = 1;
    int primitiveInt = 1;
    long primitiveLong = 1L;
    float primitiveFloat = 1f;
    String s = "1";
    Long i1 = new Long(primitiveByte);
    Long i2 = new Long(primitiveInt);
    Long i3 = new Long(primitiveLong);
    Long i4 = new Long(primitiveFloat);
    Long i5 = new Long(s);
    int i6 = i1.intValue() + i2.intValue() +
            i3.intValue() + i4.intValue() +
            i5.intValue();
    System.out.print(i6);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 5
b. Prints: 5.0
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 5**

```
class C {
  public static void main (String[] args) {
    Float f1 = new Float(1.0);     // 1
    Float f2 = new Float("1.0");   // 2
    Float f3 = new Float("1.0f");  // 3
    Float f4 = new Float("1e1f");  // 4
    Float f5 = new Float(".1e1d"); // 5
}}
```

What is the result of attempting to compile and run the program?

a. Compile-time error at 1
b. Compile-time error at 2
c. Compile-time error at 3
d. Compile-time error at 4
e. Compile-time error at 5
f. Run-time error at 1
g. Run-time error at 2
h. Run-time error at 3
i. Run-time error at 4
j. Run-time error at 5
k. None of the above

---

**Question 6**

```
class A {
  public static void main(String[] args) {
    Boolean b1 = new Boolean(true);    // 1
    Boolean b2 = new Boolean(false);   // 2
    Boolean b3 = new Boolean(TRUE);    // 3
    Boolean b4 = new Boolean(FALSE);   // 4
    Boolean b5 = new Boolean("TrUe");  // 5
    Boolean b6 = new Boolean("fAlSe"); // 6
}}
```

Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3
d. 4
e. 5
f. 6

---

**Question 7**

Which of the following class instance creation expressions would generate a compile-time error?

a. new Short(1)
b. new Short('1')
c. new Short('b' - 'a')
d. new Short((short)1 - (short)2)
e. new Short((byte)1)
f. new Short((short)1)

---

**Question 8**

```
class B {
  public static void main (String args[]) {
    Byte b1 = new Byte(1);    // 1
    Byte b2 = new Byte('2');  // 2
    Byte b3 = new Byte("3");  // 3
    byte i1 = b1.byteValue()+b2.byteValue()+b3.byteValue(); // 4
    System.out.print(i1);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 6
b. Compile-time error at 1
c. Compile-time error at 2
d. Compile-time error at 3
e. Compile-time error at 4
f. Run-time error

---

**Question 9**

```
class A {
  public static void main (String args[]) {
    int primitiveInt = 1;
    long primitiveLong = 1L;
    float primitiveFloat = 1.0f;
    String s = "1";
    Integer i1 = new Integer(primitiveInt);
    Integer i2 = new Integer(primitiveLong);
    Integer i3 = new Integer(primitiveFloat);
    Integer i4 = new Integer(s);
    int i5 = i1.intValue() + i2.intValue() +
            i3.intValue() + i4.intValue();
    System.out.print(i5);
```

}}

What is the result of attempting to compile and run the program?

a. Prints: 4
b. Prints: 4.0
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 10**

```
class A {
  public static void main (String args[]) {
    Double d1 = new Double(1.0);
    Double d2 = new Double(d1);
    System.out.print(d1.equals(d2));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false
b. Prints: true
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 11**

```
class B {
  public static void main (String args[]) {
    Long i1 = new Long(1);
    Long i2 = new Long(i1);
```

```
   System.out.print((i1==i2) + "," + i1.equals(i2));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false
d. Prints: true,true
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 12**

```
class F {
  public static void main (String[] args) {
    Float f1 = new Float('B' - 'A');  // 1
    Float f2 = new Float(010);        // 2
    Float f3 = new Float(0x10);       // 3
    Float f4 = new Float(.1e1);       // 4
    Float f5 = new Float(1.0);        // 5
    Float f6 = new Float(1.0d);       // 6
    System.out.print(f1+","+f2+","+f3+","+f4+","+f5+","+f6);
}}
```

What is the result of attempting to compile and run the program?

a. Compile-time error at 1
b. Compile-time error at 2
c. Compile-time error at 3
d. Compile-time error at 4
e. Compile-time error at 5

f. Compile-time error at 6
g. Run-time error at 1
h. Run-time error at 2
i. Run-time error at 3
j. Run-time error at 4
k. Run-time error at 5
l. Run-time error at 6
m. Prints: 1.0,10.0,10.0,1.0,1.0,1.0
n. Prints: 1.0,8.0,16.0,1.0,1.0,1.0
o. None of the above

---

**Question 13**

```
class B {
  public static void main(String[] args) {
    Boolean b1 = new Boolean(true);
    Boolean b2 = new Boolean(true);
    Boolean b3 = new Boolean("TrUe");
    Boolean b4 = new Boolean("tRuE");
    System.out.print((b1==b2) + ",");
    System.out.print((b1.booleanValue()==b2.booleanValue()) + ",");
    System.out.println(b3.equals(b4));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false
b. Prints: false,false,true
c. Prints: false,true,false
d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true

g. Prints: true,true,false
h. Prints: true,true,true
i. None of the above

---

**Question 14**

Which of the following class instance creation expressions would generate a compile-time error?

a. new Short("1")
b. new Short("-1")
c. new Short("1.0")
d. new Short("0x1")
e. new Short("011")
f. None of the above

---

**Question 15**

```
class D {
  public static void main (String args[]) {
    Byte a = new Byte("1");
    byte b = a.byteValue();
    short c = a.shortValue();
    char d = a.charValue();
    int e = a.intValue();
    long f = a.longValue();
    float g = a.floatValue();
    double h = a.doubleValue();
    System.out.print(b+c+d+e+f+g+h);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 7

b. Prints: 7.0
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 16**

```
class A {
  public static void main (String args[]) {
    Integer i1 = new Integer(1);
    Integer i2 = new Integer(i1);
    System.out.print(i1.equals(i2));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false
b. Prints: true
c. Compile-time error
d. Run-time error
e. None of the above

---

**Question 17**

```
class B {
  public static void main (String args[]) {
    Double a = new Double(0xFFFF);
    byte b = a.byteValue();
    short c = a.shortValue();
    int e = a.intValue();
    long f = a.longValue();
    float g = a.floatValue();
    double h = a.doubleValue();
```

```
   System.out.print(b+","+c+","+ (e+f+g+h == 4 * 0xFFFF));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0xFFFF,0xFFFF,false
b. Prints: 0xFFFF,0xFFFF,true
c. Prints: -1,-1,false
d. Prints: -1,-1,true
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 18**

```
class C {
  public static void main (String args[]) {
    Long a = new Long(1);
    byte b = a.byteValue();
    short s = a.shortValue();
    char c = a.charValue();
    int d = a.intValue();
    long e = a.longValue();
    float f = a.floatValue();
    double g = a.doubleValue();
    System.out.print(b+s+c+d+e+f+g);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 7
b. Prints: 7L
c. Prints: 7.0

d. Compile-time error

e. Run-time error

f. None of the above

e Compile-time error at 3 The Byte class has only two constructors: one accepts a primitive byte; the other accepts a String. The argument that appears in the class instance creation expression new Byte(1) is of type int, but the compiler will not implicitly narrow the int to a byte. At line 1, the assignment expression primitiveByte = 1 causes the compiler to implicitly narrow a literal of type int to type byte. The compiler will implicitly do a narrowing conversion for an assignment expression if the right hand operand is a compile-time constant of type byte, short, char or int and the value falls within the range of the variable on the left and the variable is of type byte, short, or char. For that reason, a constant int expression that is equal to 1 may be assigned to a byte without an explicit cast. The same is not true for the parameters of a method or constructor. The designers of the Java programming language felt that implicit narrowing conversions of method and constructor arguments would add unnecessary complexities to the process of resolving overloaded method calls.

2 a Prints: 5 The Integer class has only two constructors: one accepts a primitive int, and the other accepts a String. If the argument is of type byte, short or char, then it is implicitly promoted to type int.

3 b Prints: 4.0 The Double constructor is overloaded: one accepts an argument of type primitive double; the other accepts an argument of type String.

4 c Compile-time error Long has two constructors: one accepts an argument of type primitive long; the other accepts an argument of type String. The class instance creation expression new Long(primitiveFloat) generates a compile-time error, because the compiler will not implicitly apply a primitive narrowing conversion to the argument.

5 k None of the above The Float constructor is overloaded: one version accepts a primitive of type float; one accepts a primitive of type double; one accepts a String representation of a floating-point literal.

6 c d 3 4 The boolean literals true and false are written with lower case letters; upper case letters produce a compile-time error. A String representation of a boolean literal is acceptable in both upper and lower case letters.

7 a b c d new Short(1) new Short('1') new Short('b' - 'a') new Short((short)1 - (short)2) The Short class has only two constructors: one accepts a primitive short; the other accepts a String. The argument that appears in the class instance creation expression new Short(1) is of type int, but the compiler will not implicitly narrow the int to a short. The argument that appears in the class instance creation expression new Short('1') is of type char, but the compiler will not implicitly convert the char to a short. The argument that appears in the class instance creation expression new Short('b' - 'a') is of type int, but the compiler will not implicitly narrow the int to a short. The argument that appears in the class instance creation expression new Short((short)1 - (short)2) is of type int, but the compiler will not implicitly narrow the int to a short. If both operands of a binary arithmetic expression are of type byte, char or short; then both are implicitly widened to type int, and the result of the expression is of type int.

8 b c e Compile-time error at 1 Compile-time error at 2 Compile-time error at 4 The Byte class has only two constructors: one accepts a primitive byte; the other accepts a String. The argument that appears in the class instance creation expression new Byte(1) is of type int, but the compiler will not implicitly narrow it to type byte. The argument that appears in the class instance creation expression new Byte('2') is of type char, but the compiler will not implicitly narrow it to type byte. At line 4, the result of the additive expression is of type int, but the variable is of type byte. The assignment expression generates a compile-time error.

9 c Compile-time error The Integer class has only two constructors: one accepts a primitive int, and the other accepts a String. The class instance creation expression new Integer(primitiveLong) generates a compile-time error, because the compiler will not apply an implicit narrowing conversion to the argument. For the same reason, the class instance creation expression new Integer(primitiveFloat) generates a compile-time error.

10 c Compile-time error The Double constructor is overloaded: one accepts an argument of type primitive double; the other accepts an argument of type String. There is no constructor that accepts a reference to an instance of type Double.

11 e Compile-time error Long has two constructors: one accepts an argument of type primitive long; the other accepts an argument of type String. The class instance creation expression new Long(i1) generates a compile-time error, because there is no constructor that accepts a reference to an instance of type Long.
12 n Prints: 1.0,8.0,16.0,1.0,1.0,1.0 The Float constructor is overloaded: one version accepts a primitive of type float; one accepts a primitive of type double; one accepts a String representation of a floating-point literal. All numeric values can be promoted to type double; so all numeric values are accepted by the float constructor.
13 d Prints: false,true,true Four instances of type Boolean containing the value true are created. The equality expression b1==b2 evaluates to false, because the unique instances of type Boolean have different reference values. The instance method Boolean.booleanValue returns a copy of the primitive boolean value that is contained by the instance. Since the boolean values of b1 and b2 are the same, the result of the equality expression b1.booleanValue()==b2.booleanValue() is true. The equals method compares the values of the primitives that are wrapped by the Boolean instances. Since the wrapped primitive values are the same, the result of the method invocation expression b3.equals(b4) is true.
14 f None of the above None of the String arguments would generate a compile-time error, but two would generate a run-time error. The argument 1.0 would generate a run-time error, because a floating-point value is not acceptable. The argument 0x1 would generate a run-time error, because a hexadecimal integer literal is not acceptable. The argument must be a decimal integer literal. The leading 0 of the argument 011 is ignored and the argument is interpreted as a decimal integer literal.
15 c Compile-time error A compile-time error is generated, because the Byte class does not have a charValue method. The Byte class extends the Number class and implements all six of the methods declared in Number.
16 c Compile-time error The Integer class has only two constructors: one accepts a primitive int, and the other accepts a String. There is no constructor that accepts an argument that is an instance of type Integer.
17 d Prints: -1,-1,true Double is a subclass of the abstract class Number, and implements all of the methods of Number such as byteValue, shortValue, floatValue, etc. In this case, the Double instance contains the value 0xFFFF. When that value is converted to type byte the result is 0xFF which is also the two's complement representation of the byte value -1. Similarly, 0xFFFF is the two's complement representation of the short value -1. Please note there is no Double.charValue method.
18 d Compile-time error Long is a subclass of the abstract class Number, and Long implements all of the methods of Number: byteValue, shortValue, intValue, longValue, floatValue and doubleValue. The attempt to invoke the charValue method on an instance of Long generates a compile-time error, because there is no charValue method.

**Question 1**

Which of the instance creation expressions produce a run-time error?

a. new Float('A')
b. new Float("A")
c. new Float(1L)
d. new Float("1L")

e. new Float(0x10)

f. new Float("0x10")

g. new Float("010")

---

**Question 2**

```
class C {
  public static void main(String[] args) {
    Boolean b1 = Boolean.valueOf(true);
    Boolean b2 = Boolean.valueOf(true);
    Boolean b3 = Boolean.valueOf("TrUe");
    Boolean b4 = Boolean.valueOf("tRuE");
    System.out.print((b1==b2) + ",");
    System.out.print((b1.booleanValue()==b2.booleanValue()) + ",");
    System.out.println(b3.equals(b4));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false

b. Prints: false,false,true

c. Prints: false,true,false

d. Prints: false,true,true

e. Prints: true,false,false

f. Prints: true,false,true

g. Prints: true,true,false

h. Prints: true,true,true

i. Compile-time error

j. Run-time error

k. None of the above

---

**Question 3**

Which of the following class instance creation expressions would generate a run-time error?

a. new Short("1")
b. new Short("-1")
c. new Short("+1")
d. new Short("1.0")
e. new Short("0x1")
f. new Short("011")

---

**Question 4**

```
class E {
  public static void main (String[] args) {
    Byte b1 = new Byte("1"), b2 = new Byte("1");
    System.out.print((b1==b2)+","+(b1.equals(b2)));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false
d. Prints: true,true
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 5**

```
class B {
  public static void main (String args[]) {
    Integer a = new Integer(256);
```

```
    byte b = a.byteValue();
    short c = a.shortValue();
    int e = a.intValue();
    long f = a.longValue();
    float g = a.floatValue();
    double h = a.doubleValue();
    System.out.print(b+","+c+","+ (e+f+g+h == 4 * 256));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 0,0,false
b. Prints: 0,0,true
c. Prints: 0,-1,false
d. Prints: 0,-1,true
e. Prints: -1,0,false
f. Prints: -1,0,true
g. Prints: -1,-1,false
h. Prints: -1,-1,true
i. Compile-time error
j. Run-time error
k. None of the above

---

**Question 6**

```
class D {
  static boolean m(double v) {
    return(v != v == Double.isNaN(v));
  }
  public static void main (String args[]) {
    double d1 = Double.NaN;
    double d2 = Double.POSITIVE_INFINITY;
    double d3 = Double.MAX_VALUE;
    System.out.print(m(d1) + "," + m(d2) + "," + m(d3));
```

}}

What is the result of attempting to compile and run the program?

a. Prints: false,false,false
b. Prints: false,false,true
c. Prints: false,true,false
d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true
g. Prints: true,true,false
h. Prints: true,true,true
i. Compile-time error
j. Run-time error
k. None of the above

---

**Question 7**

```
class F {
  static String m(long i) {return "long";}
  static String m(Long i) {return "Long";}
  static String m(double i) {return "double";}
  static String m(Double i) {return "Double";}
  static String m(String i) {return "String";}
  public static void main (String[] args) {
    System.out.print(m(Long.parseLong("1")));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: long
b. Prints: Long
c. Prints: double

d. Prints: Double
e. Prints: String
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 8**

```
class E {
  public static void main (String args[]) {
    String s1 = Float.toString(1.0);  // 1
    String s2 = Float.toString(1.0f); // 2
    String s3 = Float.toString(0xf);  // 3
    String s4 = Float.toString(010);  // 4
    String s5 = Float.toString('A');  // 5
}}
```

What is the result of attempting to compile and run the program?

a. Compile-time error at 1
b. Compile-time error at 2
c. Compile-time error at 3
d. Compile-time error at 4
e. Compile-time error at 5
f. Run-time error at 1
g. Run-time error at 2
h. Run-time error at 3
i. Run-time error at 4
j. Run-time error at 5
k. None of the above

---

**Question 9**

```
class D {
  public static void main (String[] args) {
    Boolean b1 = new Boolean("trUE");          // 1
    Boolean b2 = new Boolean("What's This?");  // 2
    Boolean b3 = new Boolean(null);            // 3
    System.out.print(b1 + "," + b2 + "," + b3);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false
b. Prints: false,false,true
c. Prints: false,true,false
d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true
g. Prints: true,true,false
h. Prints: true,true,true
i. Compile-time error
j. Run-time error
k. None of the above

---

**Question 10**

```
class F {
  public static void main (String[] args) {
    Short s1 = new Short("1"), s2 = new Short("1");
    int a1 = s1.hashCode(), b1 = s2.hashCode();
    System.out.print((s1==s2)+","+(s1.equals(s2))+","+(a1==b1));
}}
```

What is the result of attempting to compile and run the program?

a. false,false,false
b. false,false,true
c. false,true,false
d. false,true,true
e. true,false,false
f. true,false,true
g. true,true,false
h. true,true,true
i. Compile-time error
j. Run-time error
k. None of the above

---

**Question 11**

```
class E {
  public static void main (String[] args) {
    Byte b1 = new Byte("1"), b2 = new Byte("1");
    int a = b1.hashCode(), b = b2.hashCode();
    System.out.print((b1.equals(b2))+","+(a==b));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false
d. Prints: true,true
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 12**

```
class E {
  static String m(int i) {return "int primitive";}
  static String m(Integer i) {return "Integer instance";}
  static String m(double i) {return "double primitive";}
  static String m(Double i) {return "Double instance";}
  static String m(String i) {return "String instance";}
  public static void main (String[] args) {
    System.out.print(m(Integer.parseInt("1")));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: int primitive
b. Prints: double primitive
c. Prints: Double instance
d. Prints: String instance
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 13**

Which of the following methods are static members of the java.lang.Double class?

a. DoubleValue
b. FloatValue
c. IntValue
d. LongValue
e. ParseDouble
f. toString(double)
g. ValueOf

---

**Question 14**

```
class F {
  static String m(long i) {return "long";}
  static String m(Long i) {return "Long";}
  static String m(double i) {return "double";}
  static String m(Double i) {return "Double";}
  static String m(String i) {return "String";}
  public static void main (String[] args) {
    System.out.print(m(Long.parseLong("1L")));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: long
b. Prints: Long
c. Prints: double
d. Prints: Double
e. Prints: String
f. Compile-time error
g. Run-time error
h. None of the above

---

**Question 15**

```
class E {
  static String m(float f) {return "f";}
  static String m(Float f) {return "F";}
  public static void main (String[] args) {
    System.out.print(m(Float.parseFloat("1")));
    System.out.print(m(Float.floatValue()));
    System.out.print(m(Float.valueOf("1")));
```

}}

What is the result of attempting to compile and run the program?

a. Prints: fff
b. Prints: ffF
c. Prints: fFf
d. Prints: Fff
e. Prints: Fff
f. Prints: FfF
g. Prints: FFf
h. Prints: FFF
i. Compile-time error
j. Run-time error
k. None of the above

---

**Question 16**

```
class D {
  public static void main (String[] args) {
    Boolean b1 = Boolean.valueOf("trUE");          // 1
    Boolean b2 = Boolean.valueOf("Even more true"); // 2
    Boolean b3 = Boolean.valueOf(null);            // 3
    System.out.print((b1==b2) + ",");
    System.out.print((b2==b3) + ",");
    System.out.println(b3==b1);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false
b. Prints: false,false,true
c. Prints: false,true,false

d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true
g. Prints: true,true,false
h. Prints: true,true,true
i. Compile-time error
j. Run-time error
k. None of the above

---

**Question 17**

```
class E {
  static String m(short s) {return "p";}
  static String m(Short s) {return "S";}
  public static void main (String[] args) {
    Short s1 = new Short("1");
    System.out.print(m(s1.shortValue())+",");
    System.out.print(m(Short.parseShort("1"))+",");
    System.out.print(m(Short.valueOf("1")));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: p,p,p
b. Prints: p,p,S
c. Prints: p,S,p
d. Prints: p,S,S
e. Prints: S,p,p
f. Prints: S,p,S
g. Prints: S,S,p
h. Prints: S,S,S
i. Compile-time error

j. Run-time error

k. None of the above

---

**Question 18**

```
class F {
  public static void main (String[] args) {
    System.out.print(Byte.MIN_VALUE+","+Byte.MAX_VALUE);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: -128,127

b. Prints: -127,128

c. Prints: 0,255

d. Compile-time error

e. Run-time error

f. None of the above

1 b d f new Float("A") new Float("1L") new Float("0x10") The Float constructor is overloaded: one version accepts a primitive of type float; one accepts a primitive of type double; one accepts a String representation of a floating-point literal. The primitive char literal 'A' is converted to a float, and is accepted by the constructor that declares a parameter of type float. The String literals "NaN" and "Infinity" are accepted by the Float constructor. A sign (+ or -) is optional. The API specification states that any other String must represent a floating-point value; however, a little experimentation proves that a String is acceptable if it can be parsed as a decimal integer value. The leading 0 of an octal value is ignored, and the String is parsed as a decimal value. A String representation of a hexadecimal value is not acceptable. The String "A" does not represent a floating-point literal value; therefore, a NumberFormatException is thrown. Arguments of type String can not contain an integer type suffix, L or l. A floating-point suffix, F, f, D or d, is acceptable, but the suffix has no impact on the result.

2 h Prints: true,true,true The Boolean class contains two public static final Boolean instances: Boolean.FALSE wraps the primitive boolean value false; Boolean.TRUE wraps the primitive boolean value true. Depending on the value of the argument, the Boolean.valueOf method returns a reference to either Boolean.FALSE or Boolean.TRUE. Reference variables b1 and b2 are both initialized with a reference value returned by the method invocation expression Boolean.valueOf(true); so the equality expression b1==b2 is true. Please note that the valueOf method that accepts an argument of type primitive boolean was introduced in the 1.4 version of Java.

3 c d e new Short("+1") new Short("1.0") new Short("0x1") The Short class has only two constructors: one accepts a primitive short; the other accepts a String. A String argument must represent an integral primitive type. A leading minus sign can be added to indicate a negative value. A leading plus sign generates a run-time error. The constructor is not able to determine the radix of the String value by examing a prefix such as 0 or 0x. The 0 prefix used to identify octal

values is accepted, but the String is parsed as a decimal value. The prefix 0x generates a run-time error. A run-time error is generated if the String argument is not formatted as a decimal integer. A floating-point format results in a run-time error.

4 b Prints: false,true The expression b1==b2 compares the references of two instances of Byte. The result is false, because the instances are distinct. The expression b1.equals(b2) compares the contents of two instances of Byte. The result is true, because the two instances contain the same value.

5 k None of the above The binary representation of 256 is one bit that is set to one followed by eight bits that are set to zero. When 256 is converted to an eight bit byte value, the bit that is set to one is lost and only the bits that are set to zero remain. When 256 is converted to a short, no information is lost; so the value remains 256.

6 h Prints: true,true,true NaN is the only value that is not equal to itself. The Double.isNaN method returns the result of the expression (v != v).

7 a Prints: long The Long.parseLong method returns a primitive long.

8 a Compile-time error at 1 The Float.toString method is overloaded: one declares no parameters and returns the value wrapped by the Float instance; the other accepts a primitive of type float. The literal, 1.0, is of type double and can not be implicitly narrowed to type float.

9 e Prints: true,false,false The Boolean constructor is overloaded: one version accepts a primitive boolean argument; the other accepts a String. If the String value is the word true, then the new Boolean instance will contain the value true. Both upper and lower case letters are acceptable. If the String contains any word other than true or if the reference is null, then the new instance will contain the value false.

10 d false,true,true The equality expression s1==s2 compares the reference values of two distinct instances of type Short. Since the instance are distinct, the equality expression is false. The expression s1.equals(s2) compares the values of two instances of type Short. Since both instances contain the value 1, the returned value is true. The expression a1==b1 compares the hash codes of two instances of Short. The result is true, because the two instances contain the same value.

11 d Prints: true,true The expression b1.equals(b2) compares the values of two instances of type Byte. Since both instances contain the value 1, the return value is true. The expression a==b compares the hash codes of two instances of Byte. The result is true, because the two instances contain the same value.

12 a Prints: int primitive The Integer.parseInt method returns a primitive of type int.

13 e f g parseDouble toString(double) valueOf

14 g Run-time error The Long.parseLong method accepts a String parameter that represents a numeric value. Long.parseLong assumes that the input String represents a decimal value unless a second parameter is provided to specify the radix. All of the characters in the String must be digits of the specified radix. The parseLong method does not determine the type of the numeric value based on a suffix such as l, L, f, F, d, or D. Use of any such suffix generates a NumberFormatException at run-time.

15 i Compile-time error The Float.floatValue method is not static; it must be invoked on an instance of type Float.

16 c Prints: false,true,false The Boolean.valueOf method is overloaded: one version accepts a primitive boolean argument; the other accepts a String. If the String value is the word true, then the new Boolean instance will contain the value true. Both upper and lower case letters are acceptable. If the String contains any word other than true or if the String reference is null, then the new instance will contain the value false.

17 b Prints: p,p,S The Short.shortValue method and the Short.parseShort method both return primitives of type short. The Short.valueOf method returns an instance of type Short.

18 a Prints: -128,127

**Question 1**

```
class D {
  public static void main (String args[]) {
```

```
   boolean b1 = Integer.MIN_VALUE == 0x80000000;
   boolean b2 = Integer.MAX_VALUE == 0x7FFFFFFF;
   System.out.print(b1 + "," + b2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false
d. Prints: true,true
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 2**

Which methods of the java.lang.Double class return a primitive value?

a. doubleValue
b. floatValue
c. intValue
d. longValue
e. parseDouble
f. toString(double)
g. valueOf

---

**Question 3**

```
class G {
  public static void main (String[] args) {
    Long l1 = new Long("1"), l2 = new Long("1");
```

```
    int h1 = l1.hashCode(), h2 = l2.hashCode();
    StringBuffer s1 = new StringBuffer("1");
    StringBuffer s2 = new StringBuffer("1");
    int h3 = s1.hashCode(), h4 = s2.hashCode();
    System.out.print((l1.equals(l2) & (h1==h2)) + ",");
    System.out.print(s1.equals(s2) | (h3==h4));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false
d. Prints: true,true
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 4**

```
class F {
  static String m(float f) {return "f";}
  static String m(Float f) {return "F";}
  public static void main (String[] args) {
    Float f1 = new Float(1);
    System.out.print(m(f1.parseFloat("1")));
    System.out.print(m(f1.floatValue()));
    System.out.print(m(f1.valueOf("1")));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: fff
b. Prints: ffF

c. Prints: fFf
d. Prints: fFF
e. Prints: Fff
f. Prints: FfF
g. Prints: FFf
h. Prints: FFF
i. Compile-time error
j. Run-time error
k. None of the above

---

**Question 5**

```
class E {
  static String m1(boolean b) {return "b";}
  static String m1(Boolean b) {return "B";}
  public static void main(String[] args) {
    Boolean b1 = new Boolean(true);
    System.out.print(m1(Boolean.valueOf(null)));
    System.out.print(m1(b1.booleanValue()));
    System.out.println(m1(Boolean.TRUE));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: bbb
b. Prints: bbB
c. Prints: bBb
d. Prints: bBB
e. Prints: Bbb
f. Prints: BbB
g. Prints: BBb
h. Prints: BBB

i. Compile-time error

j. Run-time error

k. None of the above

---

**Question 6**

```
class C {
  public static void main (String args[]) {
    String s1 = "1", s2 = "2";
    Byte b1 = Byte.parseByte(s1);
    Byte b2 = Byte.parseByte(s2);
    System.out.print(b1.byteValue() + b2.byteValue());
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 3

b. Prints: 12

c. Compile-time error

d. Run-time error

e. None of the above

---

**Question 7**

```
class A {
  public static void main (String[] args) {
    String s = "11";
    int i1 = Integer.parseInt(s);
    System.out.print(new Integer(i1).equals(new Integer(i1)) + ",");
    System.out.print(new Integer(i1).equals(new Integer(s))  + ",");
    System.out.print(new Integer(i1) == new Integer(i1));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false
b. Prints: false,false,true
c. Prints: false,true,false
d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true
g. Prints: true,true,false
h. Prints: true,true,true
i. Compile-time error
j. Run-time error
k. None of the above

---

**Question 8**

Which method of the java.lang.Double class returns an instance of type Double?

a. doubleValue
b. floatValue
c. intValue
d. longValue
e. parseDouble
f. toString(double)
g. valueOf
h. None of the above

---

**Question 9**

Which of the method invocation expressions would produce a run-time error?

a. Long.parseLong("1")
b. Long.parseLong("1L")
c. Long.parseLong("010")
d. Long.parseLong("0x10")
e. Long.parseLong("1.0")

---

**Question 10**

```
class B {
  public static void main (String[] args) {
    byte b1 = 11;
    System.out.print(new Integer(b1).equals(new Integer(b1)) + ",");
    System.out.print(new Integer(b1).equals(new Short(b1)) + ",");
    System.out.print(new Integer(b1).equals(new Byte(b1)));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false
b. Prints: false,false,true
c. Prints: false,true,false
d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true
g. Prints: true,true,false
h. Prints: true,true,true
i. Compile-time error
j. Run-time error
k. None of the above

---

**Question 11**

Which methods of the java.lang.Double class declare a parameter of type String?

a. doubleValue
b. floatValue
c. intValue
d. longValue
e. parseDouble
f. valueOf

---

**Question 12**

Which of the method invocation expressions would produce a run-time error?

a. Long.parseLong("-1")
b. Long.parseLong("+1")
c. Long.parseLong("01")
d. Long.parseLong("1L")
e. Long.parseLong("1.0")

---

**Question 13**

Which of the following methods of the java.lang.Integer class returns a primitive int type?

a. intValue
b. parseInt
c. valueOf

---

**Question 14**

Which methods of the java.lang.Double class declare a NumberFormatException in the throws clause?

a. doubleValue
b. floatValue
c. intValue
d. longValue
e. parseDouble
f. toString(double)
g. valueOf

---

**Question 15**

```
class F {
  public static void main (String[] args) {
    Integer i1 = new Integer("1");
    Integer i2 = new Integer("1");
    int h1 = i1.hashCode(), h2 = i2.hashCode();
    StringBuffer sb1 = new StringBuffer("1");
    StringBuffer sb2 = new StringBuffer("1");
    int h3 = sb1.hashCode(), h4 = sb2.hashCode();
    System.out.print((h1==h2)+","+(h3==h4));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false
d. Prints: true,true
e. Compile-time error
f. Run-time error
g. None of the above

---

**Question 16**

```
class F {
  public static void main (String args[]) {
    Double d1 = new Double("0x10");         // 1
    double d2 = Double.parseDouble("0x10"); // 2
    Double d3 = Double.valueOf("0x10");     // 3
    System.out.print(d1+","+d2+","+d3);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: 10,10,10
b. Prints: 16,16,16
c. Compile-time error at line 1
d. Compile-time error at line 2
e. Compile-time error at line 3
f. Run-time error
g. None of the above

---

**Question 17**

```
class G {
  public static void main (String[] args) {
    Integer i1 = new Integer("1"), i2 = new Integer("1");
    StringBuffer sb1 = new StringBuffer("1");
    StringBuffer sb2 = new StringBuffer("1");
    System.out.print(sb1.equals(sb2)+","+i1.equals(i2));
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false
b. Prints: false,true
c. Prints: true,false

d. Prints: true,true

e. Compile-time error

f. Run-time error

g. None of the above

---

**Question 18**

```
class H {
  public static void main (String[] args) {
    int i1, i2; Integer i3, i4;
    i1=new Integer(Integer.parseInt("1")).intValue();     // 1
    i3=new Integer(Integer.parseInt("1")).intValue();     // 2
    i2=Integer.parseInt(Integer.valueOf("1").toString()); // 3
    i4=Integer.parseInt(Integer.valueOf("1").intValue()); // 4
}}
```

What is the result of attempting to compile and run the program?

a. Compile-time error at 1.

b. Compile-time error at 2.

c. Compile-time error at 3.

d. Compile-time error at 4.

e. Compile-time error

f. Run-time error

1 d Prints: true,true An int is a 32-bit signed integral value that is stored in two's complement format. The left most bit is the sign bit. The sign bit is set to one for negative numbers and is set to zero for positive numbers.

2 a b c d e doubleValue floatValue intValue longValue parseDouble

3 c Prints: true,false Long.equals overrides Object.equals. The Long.equals method compares the data values contained in the Long instances. The StringBuffer.equals method does not override Object.equals. The StringBuffer.equals method compares the reference values of the instances. Two distinct instances of StringBuffer will not have the same reference values; so the equals method returns false. The StringBuffer.hashCode method typically returns a value that is based on the internal address of the StringBuffer instance. Two instances of StringBuffer will not have the same hash code.

4 b Prints: ffF

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

6 c Compile-time error The Byte.parseByte method returns a primitive byte. A compile-time error is generated as a result of the attempt to assign a primitive byte to a Byte reference variable.

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

8 g valueOf

9 b d e Long.parseLong("1L") Long.parseLong("0x10") Long.parseLong("1.0") Long.parseLong is overloaded: one version accepts a String argument that represents an integral value; the other accepts both a String argument and an argument of type int. The int argument represents the radix (i.e. base) of the String argument. The Long.parseLong method is not able to determine the type of the String value by examing a suffix such as L. Any such suffix results in a run-time error. The Long.parseLong method is not able to determine the radix of the String value by examing a prefix such as 0 or 0x. The 0 prefix used to identify octal values is accepted, but the String is parsed as a decimal value. The prefix 0x generates a run-time error. The Long.parseLong method generates a run-time error if the String argument is not formatted as a decimal integer. A floating-point format results in a run-time error.

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

11 e f parseDouble valueOf

12 b d e Long.parseLong("+1") Long.parseLong("1L") Long.parseLong("1.0") Long.parseLong is overloaded: one version accepts a String argument that represents an integral value; the other accepts both a String argument and an argument of type int. The int argument represents the radix (i.e. base) of the decimal integral value represented by the String argument. The Long.parseLong method is not able to determine the type of the String value by examing a suffix such as L. Any such suffix results in a run-time error. The Long.parseLong method is not able to determine the radix of the String value by examing a prefix such as 0 or 0x. The 0 prefix used to identify octal values is accepted, but the String is parsed as a decimal value. The prefix 0x generates a run-time error. A leading minus sign (-) can be added to indicate a negative value. A leading plus sign (+) results in a run-time error. The Long.parseLong method generates a run-time error if the String argument is not formatted as a decimal integer. A floating-point format results in a run-time error.

13 a b intValue parseInt Integer.valueOf returns an instance of the Integer wrapper class.

14 e g parseDouble valueOf

15 c  Prints: true,false  Integer.hashCode overrides Object.hashCode. The Integer.hashCode method calculates the hash code based on the value contained in the Integer instance. The StringBuffer.hashCode method does not override Object.hashCode. The StringBuffer.hashCode method typically returns a value that is based on the internal address of the StringBuffer instance. Two instances of StringBuffer will not have the same hash code.
16 f  Run-time error  An argument of type String must represent a floating-point value. The argument "0x10" represents a hexadecimal integer literal; so a NumberFormatException would be thrown at run-time.
17 b  Prints: false,true  Integer.equals overrides Object.equals. The Integer.equals method compares the data values contained in the Integer instances. The StringBuffer.equals method does not override Object.equals. The StringBuffer.equals method compares the reference values of the instances. Two distinct instances of StringBuffer will not have the same reference values; so the equals method will return false.
18 b d  Compile-time error at 2.  Compile-time error at 4.  The Integer.parseInt method is overloaded: one version accepts one argument of type String; the other accepts one String and an int. Both versions of Integer.parseInt return a primitive int. The Integer.intValue method returns the value of the Integer instance as a primitive int. The Integer.valueOf method is overloaded: one version accepts one argument of type String; the other accepts one String and an int. Both versions of Integer.valueOf return an instance of Integer.

# Section 6: Overloading, Overriding, Runtime Type and Object Orientation

# 6.1Encapsulation:

# Question 1

Which of the following are true statements?

a. Encapsulation is a form of data hiding.
b. A tightly encapsulated class is always immutable.
c. Encapsulation is always used to make programs run faster.
d. Encapsulation helps to protect data from corruption.
e. Encapsulation allows for changes to the internal design of a class while the public interface remains unchanged.

---

# Question 2

Which of the following are true statements?

a. A top-level class can not be called "tightly encapsulated" unless it is declared private.

b. Encapsulation enhances the maintainability of the code.
c. A tightly encapsulated class allows fast public access to member fields.
d. A tightly encapsulated class allows access to data only through accessor and mutator methods.
e. Encapsulation usually reduces the size of the code.
f. A tightly encapsulated class might have mutator methods that validate data before it is loaded into the internal data model.

---

## Question 3

A class can not be called "tightly encapsulated" unless which of the following is true?

a. The class is declared final.
b. All local variables are declared private.
c. All method parameters are declared final.
d. No method returns a reference to any object that is referenced by an internal data member.
e. None of the above

---

## Question 4

A class can not be called "tightly encapsulated" unless which of the following is true?

a. All of the methods are declared private.
b. All of the methods are synchronized.
c. All local variables are declared final.
d. The class is a direct subclass of Object.
e. Accessor methods are used to prevent fields from being set with invalid data.
f. None of the above

---

## Question 5

A class can not be called "tightly encapsulated" unless which of the following are true?

a. The data members can not be directly manipulated by external code.
b. The class is declared final.
c. It has no public mutator methods.
d. The superclass is tightly encapsulated.

---

# Question 6

A class can not be called "tightly encapsulated" unless which of the following is true?

a. The class is a nested class.
b. The constructors are declared private.
c. The mutator methods are declared private.
d. The class implements the Encapsulated interface.
e. None of the above

---

# Question 7

A class can not be called "tightly encapsulated" unless which of the following is true?

a. All member fields are declared final.
b. The class is not anonymous.
c. The internal data model can be read and modified only through accessor and mutator methods.
d. The class is an inner class.
e. None of the above

---

# Question 8

```
class GFC500 {private String name;}
class GFC501 {
  private String name;
  private void setName(String name) {this.name = name;}
```

```
  private String getName() {return name;}
}
class GFC502 {
  private String name;
  public void setName(String name) {this.name = name;}
  public String getName() {return name;}
}
```

Which class is not tightly encapsulated?

a. GFC501
b. GFC502
c. GFC503
d. None of the above

---

# Question 9

```
class GFC505 extends GFC504 {
  public void setName(String name) {this.name = name;}
  public String getName() {return name;}
}
class GFC504 extends GFC503 {
  private void setName(String name) {this.name = name;}
  private String getName() {return name;}
}
class GFC503 {String name;}
```

Which class is tightly encapsulated?

a. GFC503
b. GFC504
c. GFC505
d. None of the above

---

# Question 10

```
class GFC506 {private String name;}
class GFC507 extends GFC506 {
  String name;
  public void setName(String name) {this.name = name;}
  public String getName() {return name;}
}
class GFC508 extends GFC506 {
  private String name;
  public GFC508(String name) {setName(name);}
  public void setName(String name) {this.name = name;}
  public String getName() {return name;}
}
```

Which class is not tightly encapsulated?

a. GFC506
b. GFC507
c. GFC508
d. None of the above

---

**No. Answer Remark**

**1 a d e Encapsulation is a form of data hiding. Encapsulation helps to protect data from corruption. Encapsulation allows for changes to the internal design of a class while the public interface remains unchanged. A tightly encapsulated class does not allow direct public access to the internal data model. Instead, access is permitted only through accessor (i.e. get) and mutator (i.e. set) methods. The additional time required to work through the accessor and mutator methods typically slows execution speed. Encapsulation is a form of data hiding. A tightly encapsulated class does not allow public access to any data member that can be changed in any way; so encapsulation helps to protect internal data from the possibility of corruption from external influences. The mutator methods can impose contraints on the argument values. If an argument falls outside of the acceptable range, then a mutator method could throw an IllegalArgumentException. The internal design of a tightly encapsulated class can change while the public interface remains unchanged. An immutable class is always tightly encapsulated, but not every tightly encapsulation class is immutable.**

**2 b d f Encapsulation enhances the maintainability of the code. A tightly encapsulated class allows access to data only through accessor and mutator methods. A tightly encapsulated class might have mutator methods that validate data before it is loaded into the internal data model. The data members of a tightly encapsulated class are declared private; so changes to the data model are less likely to impact external code. Access to internal data can be provided by public accessor (i.e. get) and mutator (i.e. set) methods. The mutator methods can be used to validate the data before it is loaded into the internal data model. The use of accessor and mutator methods is likely to increase the size of the code and slow execution speed.**

**3 e None of the above If a class A has a method that returns a reference to an internal, mutable object; then external code can use the reference to modify the internal state of class A. Therefore, class A can not be considered tightly encapsulated. However, the methods of a tightly encapsulated class may return a reference to an immutable object or a reference to a copy or clone of an internal object.**

**4 f None of the above One answer option reads as follows: "Accessor methods are used to prevent fields from being set with invalid data." The answer would be correct if the word "Accessor" were replaced by the word "Mutator". Accessor methods are used to read data members; mutator methods are used to set data members. The mutator methods can validate the parameter values before the values are used to change the state of the internal data model.**

**5 a d The data members can not be directly manipulated by external code. The superclass is tightly encapsulated. If a class A is not tightly encapsulated, then no subclass of A is tightly encapsulated.**

**6 e None of the above A tightly encapsulated class may have public mutator methods.**

**7 c The internal data model can be read and modified only through accessor and mutator methods. A class is not tightly encapsulated if the internal data model can be read and/or modified without working through accessor (i.e. get) and mutator (i.e. set) methods.**

**8 d None of the above All three classes are tightly encapsulated, because the data members are private. A tightly encapsulated class can have public accessor and mutator methods, but it is not required to have those methods.**

**9 d None of the above Class GFC503 is not tightly encapsulated; so no subclass of GFC503 is tightly encapsulated.**

**10 b GFC507 Class GFC507 has a public field; so it is not tightly encapsulated.**

# 6.2Inheritance:

# Question 1

Which of the following statements are true?

a. A constructor can invoke another constructor of the same class using the alternate constructor invocation, "this(argumentListopt);".
b. A constructor can invoke itself using the alternate constructor invocation, "this(argumentListopt);".
c. The alternate constructor invocation, "this(argumentListopt);", can legally appear anywhere in the constructor body.
d. A constructor can invoke the constructor of the direct superclass using the superclass constructor invocation, "super(argumentListopt);".
e. The number of constructor invocations that may appear in any constructor body can equal but not exceed the number of alternate constructors declared in the same class.
f. A constructor is not permitted to throw an exception.

---

# Question 2

Suppose that the superclass constructor invocation, "super(argumentListopt);", appears explicitly in a subclass constructor. If a compile-time error is to be avoided then the arguments for the superclass constructor invocation, "super(argumentListopt);", can not refer to which of the following?

a. Static variables declared in this class or any superclass.
b. Instance variables declared in this class or any superclass.
c. Static methods declared in this class or any superclass.
d. Instance methods declared in this class or any superclass.
e. The keyword this.
f. The keyword super.

---

# Question 3

```
class A {void m1(String s1) {}}
class B extends A {
  void m1(String s1) {}  // 1
  void m1(boolean b) {}  // 2
  void m1(byte b) throws Exception {}  // 3
  String m1(short s) {return new String();} //4
  private void m1(char c) {} // 5
  protected void m1(int i) {} // 6
}
```

What is the result of attempting to compile the program?

a. Compile-time error at line 1
b. Compile-time error at line 2
c. Compile-time error at line 3
d. Compile-time error at line 4
e. Compile-time error at line 5
f. Compile-time error at line 6
g. None of the above

---

# Question 4

```
class A {void m1() {System.out.print("A.m1");}}
class B extends A {
  void m1() {System.out.print("B.m1");}
  static void m1(String s) {System.out.print(s+",");}
}
class C {
  public static void main (String[] args) {B.m1("main"); new B().m1();}
}
```

What is the result of attempting to compile and run the program?

a. Prints: main,B.m1
b. Compile-time error
c. Run-time error
d. None of the above

---

# Question 5

Which of the following are true statements?

a. The relationship between a class and its superclass is an example of a "has-a" relationship.
b. The relationship between a class and its superclass is an example of an "is-a" relationship.
c. The relationship between a class and an object referenced by a field within the class is an example of a "has-a" relationship.
d. The relationship between a class and an object referenced by a field within the class is an example of an "is-a" relationship.

---

# Question 6

```
class A {String s1 = "A.s1"; String s2 = "A.s2";}
class B extends A {
  String s1 = "B.s1";
  public static void main(String args[]) {
    B x = new B(); A y = (A)x;
    System.out.println(x.s1+" "+x.s2+" "+y.s1+" "+y.s2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: B.s1 A.s2 B.s1 A.s2
b. Prints: B.s1 A.s2 A.s1 A.s2
c. Prints: A.s1 A.s2 B.s1 A.s2
d. Prints: A.s1 A.s2 A.s1 A.s2
e. Run-time error
f. Compile-time error
g. None of the above

---

# Question 7

```
class C {
  void printS1() {System.out.print("C.printS1 ");}
  static void printS2() {System.out.print("C.printS2 ");}
}
class D extends C {
  void printS1(){System.out.print("D.printS1 ");}
  void printS2() {System.out.print("D.printS2 ");}
  public static void main (String args[]) {
    C c = new D(); c.printS1(); c.printS2();
}}
```

What is the result of attempting to compile and run the program?

a. Prints: C.printS1 C.printS2
b. Prints: C.printS1 D.printS2
c. Prints: D.printS1 C.printS2
d. Prints: D.printS1 D.printS2
e. Run-time error
f. Compile-time error
g. None of the above

---

# Question 8

```
class E {
  void printS1(){System.out.print("E.printS1 ");}
  static void printS2() {System.out.print("E.printS2");}
}
class F extends E {
  void printS1(){System.out.print("F.printS1 ");}
  static void printS2() {System.out.print("F.printS2");}
  public static void main (String args[]) {
    E x = new F(); x.printS1(); x.printS2();
}}
```

What is the result of attempting to compile and run the program?

a. Prints: E.printS1 E.printS2
b. Prints: E.printS1 F.printS2
c. Prints: F.printS1 E.printS2
d. Prints: F.printS1 F.printS2
e. Run-time error
f. Compile-time error
g. None of the above

---

# Question 9

```
class P {
  static void printS1(){System.out.print("P.printS1 ");}
  void printS2() {System.out.print("P.printS2 ");}
  void printS1S2(){printS1();printS2();}
}
class Q extends P {
  static void printS1(){System.out.print("Q.printS1 ");}
  void printS2(){System.out.print("Q.printS2 ");}
  public static void main(String[] args) {
    new Q().printS1S2();
}}
```

What is the result of attempting to compile and run the program?

a. Prints: P.printS1 P.printS2

b. Prints: P.printS1 Q.printS2
c. Prints: Q.printS1 P.printS2
d. Prints: Q.printS1 Q.printS2
e. Run-time error
f. Compile-time error
g. None of the above

---

# Question 10

```
class R {
  private void printS1(){System.out.print("R.printS1 ");}
  protected void printS2() {System.out.print("R.printS2 ");}
  protected void printS1S2(){printS1();printS2();}
}
class S extends R {
  private void printS1(){System.out.print("S.printS1 ");}
  protected void printS2(){System.out.print("S.printS2 ");}
  public static void main(String[] args) {
    new S().printS1S2();
}}
```

What is the result of attempting to compile and run the program?

a. Prints: R.printS1 R.printS2
b. Prints: R.printS1 S.printS2
c. Prints: S.printS1 R.printS2
d. Prints: S.printS1 S.printS2
e. Run-time error
f. Compile-time error
g. None of the above

---

# Question 11

```
class T {
  private int i1, i2;
```

```
  void printI1I2() {System.out.print("T, i1="+i1+", i2="+i2);}
  T(int i1, int i2) {this.i1=i1; this.i2=i2;}
}
class U extends T {
  private int i1, i2;
  void printI1I2() {System.out.print("U, i1="+i1+", i2="+i2);}
  U(int i1, int i2) {this.i1=i1; this.i2=i2;}
  public static void main(String[] args) {
    T t = new U(1,2); t.printI1I2();
}}
```

What is the result of attempting to compile and run the program?

a. Prints: U, i1=1, i2=2
b. Prints: T, i1=1, i2=2
c. Prints: U, i1=null, i2=null
d. Prints: T, i1=null, i2=null
e. Run-time error
f. Compile-time error
g. None of the above

---

# Question 12

```
interface I {String s1 = "I";}
class A implements I {String s1 = "A";}
class B extends A {String s1 = "B";}
class C extends B {
  String s1 = "C";
  void printIt() {
    System.out.print(((A)this).s1 + ((B)this).s1 +
                     ((C)this).s1 + ((I)this).s1);
  }
  public static void main (String[] args) {new C().printIt();}
}
```

What is the result of attempting to compile and run the program?

a. Prints: ABCI

b. Run-time error
c. Compile-time error
d. None of the above

---

# Question 13

```
abstract class D {String s1 = "D"; String getS1() {return s1;}}
class E extends D {String s1 = "E"; String getS1() {return s1;}}
class F {
  public static void main (String[] s) {
    D x = new E(); System.out.print(x.s1 + x.getS1());
}}
```

What is the result of attempting to compile and run the program?

a. Prints: DD
b. Prints: DE
c. Prints: ED
d. Prints: EE
e. Run-time error
f. Compile-time error
g. None of the above

---

# Question 14

```
class A {static void m() {System.out.print("A");}}
class B extends A {static void m() {System.out.print("B");}}
class C extends B {static void m() {System.out.print("C");}}
class D {
  public static void main(String[] args) {
    C c = new C(); c.m(); B b = c; b.m(); A a = b; a.m();
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AAA
b. Prints: ABC
c. Prints: CBA
d. Prints: CCC
e. Compile-time error
f. Run-time error
g. None of the above

---

# Question 15

```
class A {String s1 = "A";}
class B extends A {String s1 = "B";}
class C extends B {String s1 = "C";}
class D {
  static void m1(A x) {System.out.print(x.s1);}
  static void m1(B x) {System.out.print(x.s1);}
  static void m1(C x) {System.out.print(x.s1);}
  public static void main(String[] args) {
    A a; B b; C c; a = b = c = new C(); m1(a); m1(b); m1(c);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AAA
b. Prints: ABC
c. Prints: CBA
d. Prints: CCC
e. Compile-time error
f. Run-time error
g. None of the above

---

# Question 16

```
class Leg{}
class Fur{}
abstract class Pet {
  public abstract void eat();
  public abstract void sleep();
}
class Dog extends Pet {
  Leg leftFront = new Leg(), rightFront = new Leg();
  Leg leftRear = new Leg(), rightRear = new Leg();
  Fur fur = new Fur();
  public Fur shed() {return fur;}
  public void eat() {}
  public void sleep() {}
}
class Cat extends Dog {
  public void ignoreOwner() {}
  public void climbTree() {}
}
```

Which of the following statements is not a true statement?

a. A Cat object inherits an instance of Fur and four instances of Leg from the Dog superclass.
b. A Cat object is able to sleep and eat.
c. A Cat object is able to climb a tree.
d. The relationship between Dog and Pet is an example of an appropriate use of inheritance.
e. The relationship between Cat and Dog is an example of an appropriate use of inheritance.
f. None of the above.

---

# Question 17

```
class A {
  A() {System.out.print("CA ");}
  static {System.out.print("SA ");}
}
class B extends A {
  B() {System.out.print("CB ");}
  static {System.out.print("SB ");}
  public static void main (String[] args) {B b = new B();}
}
```

What is the result of attempting to compile and run the above program?

a. Prints: SA CA SB CB
b. Prints: SA SB CA CB
c. Prints: SB SA CA CB
d. Prints: SB CB SA CA
e. Runtime Exception
f. Compiler Error
g. None of the above

---

# Question 18

```
class A {String s1="A";}
class B extends A {String s1="B";}
class C extends B {String s1="C";}
class D extends C {
  String s1="D";
  void m1() {
    System.out.print(this.s1  + ",");      // 1
    System.out.print(((C)this).s1  + ","); // 2
    System.out.print(((B)this).s1  + ","); // 3
    System.out.print(((A)this).s1);        // 4
  }
  public static void main (String[] args) {
    new D().m1(); // 5
}}
```

What is the result of attempting to compile and run the program?

a. Prints: D,D,D,D
b. Prints: D,C,B,A
c. Compile-time error at 1.
d. Compile-time error at 2.
e. Compile-time error at 3.
f. Compile-time error at 4.

g. Compile-time error at 5.
h. Run-time error
i. None of the above

---

# Question 19

```
class SuperA {String s1="SuperA";}
class SuperB {String s1="SuperB";}
class A extends SuperA {
  String s1="A";
  class B extends SuperB {  // 1
    String s1="B";
    void m1() {
      System.out.print(this.s1 + ",");   // 2
      System.out.print(super.s1 + ",");  // 3
      System.out.print(A.this.s1 + ","); // 4
      System.out.print(A.super.s1);      // 5
    }
  }
  public static void main (String[] args) {
    new A().new B().m1();  // 6
}}
```

What is the result of attempting to compile and run the program?

a. Prints: B,SuperB,B,SuperB
b. Prints: B,SuperB,A,SuperA
c. Compile-time error at 1.
d. Compile-time error at 2.
e. Compile-time error at 3.
f. Compile-time error at 4.
g. Compile-time error at 5.
h. Compile-time error at 6.
i. Run-time error
j. None of the above

---

# Question 20

```
class A {void m1() {System.out.print("A");}}
class B extends A {void m1(){System.out.print("B");}}
class C extends B {void m1() {System.out.print("C");}}
class D extends C {
  void m1() {System.out.print("D");}
  void m2() {
    m1();
    ((C)this).m1(); // 1
    ((B)this).m1(); // 2
    ((A)this).m1(); // 3
  }
  public static void main (String[] args) {
    new D().m2(); // 4
}}
```

What is the result of attempting to compile and run the program?

a. Prints: DCBA
b. Prints: DDDD
c. Compile-time error at 1.
d. Compile-time error at 2.
e. Compile-time error at 3.
f. Compile-time error at 4.
g. Run-time error
h. None of the above

---

# Question 21

```
class A {public void m1() {System.out.print("A1");}}
class B extends A {
  public void m1() {System.out.print("B1");}
  public void m2() {System.out.print("B2");}
}
class C {
  public static void main(String[] args) {
    A a1 = new B();
```

```
   a1.m1();      // 1
   a1.m2();      // 2
  ((B)a1).m1();  // 3
  ((B)a1).m2();  // 4
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A1B2B1B2
b. Prints: B1B2B1B2
c. Compile-time error at 1
d. Compile-time error at 2
e. Compile-time error at 3
f. Compile-time error at 4
g. Run-time error
h. None of the above

---

**No. Answer Remark**

**1 a d A constructor can invoke another constructor of the same class using the alternate constructor invocation, "this(argumentListopt);". A constructor can invoke the constructor of the direct superclass using the superclass constructor invocation, "super(argumentListopt);". If an alternate constructor invocation appears in the body of the constructor, then it must be the first statement. The same is true for a superclass constructor invocation. A compile-time error is generated if a constructor attempts to invoke itself either directly or indirectly.**

**2 b d e f Instance variables declared in this class or any superclass. Instance methods declared in this class or any superclass. The keyword this. The keyword super. If the superclass constructor invocation, "super(argumentListopt);", appears explicitly or implicitly, then it must be the first statement in the body of the constructor. Until the superclass constructor invocation runs to completion, no other statements are processed within the body of the constructor. The same is true of the constructors of any superclass. (Note: The primordial class, Object, does not have a superclass, so the constructors do not include a superclass constructor invocation statement.) Suppose class B is a subclass of A. The process of creating and initializing an instance of B includes the creation and initialization of an instance of the superclass A and an instance of the superclass Object. The superclass constructor invocation statement appearing in a constructor of B is invoked before the completion of the constructors of the superclasses A and Object. A superclass constructor invocation statement appearing in B can not refer to the non-static members of the superclasses, because the process of initializing those non-static superclass members is not complete when the superclass constructor invocation occurs in B. The same is true of the non-static members of B.**

**3 g None of the above If a superclass method is not private and is accessible to code in a subclass, then any subclass method that has the same signature as the superclass method must also have the same return type. In other words, if a superclass method is overridden or hidden in a**

**subclass, then the overriding or hiding subclass method must have the same return type as the superclass method. No such restriction applies to method overloading. If two methods share an overloaded method name but not the same parameter list, then the two methods need not have the same return type. Class A declares one method: The name is m1 and the single parameter is of type String. Class B extends A and declares six methods that overload the name m1. The method, B.m1(String s1), overrides the superclass method, A.m1(String s1). The overriding subclass method must have the same return type as the superclass method, but the methods that overload the name m1 are free to have different return types. An overriding subclass method is not permitted to throw any checked exception that is not listed or is not a subclass of any of those listed in the throws clause of the superclass method. No such restriction applies to method overloading. If two methods share an overloaded method name but not the same parameter list, then the two methods are free to have differing throws clauses.**

**4 a Prints: main,B.m1 Suppose a superclass method is not private and is accessible to code in a subclass. If the superclass method is declared static, then any subclass method sharing the same signature must also be declared static. Similarly, if the superclass method is not declared static, then any subclass method sharing the same signature must not be declared static. The rules governing method overloading are different. If a superclass method is declared static, then any subclass method that overloads the superclass method is free to be declared static or non-static. Similarly, if a method is declared non-static, then any overloading method is free to be declared static or non-static. Method B.m1() shares the same signature as the non-static superclass method A.m1(), so B.m1() must also be non-static. The method B.m1(String s) overloads the method name m1, but does not share the same signature with any superclass method; therefore, B.m1(String s) can be declared static even though the other methods of the same name are non-static.**

**5 b c The relationship between a class and its superclass is an example of an "is-a" relationship. The relationship between a class and an object referenced by a field within the class is an example of a "has-a" relationship. Inheritance is an example of an "is-a" relationship, because the subclass "is-a" specialized type of the superclass. The relationship between a class and an object referenced by a field declared within the class is an example of a "has-a" relationship, because the class "has-a" object.**

**6 b Prints: B.s1 A.s2 A.s1 A.s2 The variables of a subclass can hide the variables of a superclass or interface. The variable that is accessed is determined at compile-time based on the type of the reference--not the run-time type of the object. The two references x and y refer to the same instance of type B. The name x.s1 uses a reference of type B; so it refers to the variable s1 declared in class B. The name y.s1 uses a reference of type A; so it refers to the variable s1 declared in class A.**

**7 f Compile-time error Suppose a superclass method is not private and is accessible to code in a subclass. If the superclass method is declared static, then any subclass method sharing the same signature must also be declared static. Similarly, if the superclass method is declared non-static, then any subclass method sharing the same signature must also be declared non-static. The attempted declaration of the non-static method D.printS2 generates a compile-time error; because the superclass method, C.printS2, is static.**

**8 c Prints: F.printS1 E.printS2 A static method is selected based on the compile-time type of the reference--not the run-time type of the object. A non-static method is selected based on the run-time type of the object--not the compile-time type of the reference. Both method invocation expressions, x.printS1() and x.printS2(), use a reference of the superclass type, E, but the object is of the subclass type, F. The first of the two expressions invokes an instance method on an object of the subclass type; so the overriding subclass method is selected. The second invokes a static method using a reference of the superclass type; so the superclass method is selected.**

**9 b Prints: P.printS1 Q.printS2 Suppose a method m1 is invoked using the method invocation expression m1(). If m1 is a static member of the class where the invocation expression occurs, then that is the implementation of the method that is invoked at run-time regardless of the run-time type of the object. If m1 is non-static, then the selected implementation is determined at run-time based on the run-time type of the object. The program invokes method printS1S2 on an instance of class Q. The body of method printS1S2 contains two method invocation expressions, printS1() and printS2(). Since method printS1 is static, the implementation declared in class P is invoked. Since printS2 is non-static and the run-time type of the object is Q, the invoked method is the one declared in class Q.**

**10 b Prints: R.printS1 S.printS2 A private method of a superclass is not inherited by a subclass. Even if a subclass method has the same signature as a superclass method, the subclass method does not override the superclass method. Suppose a non-static method m1 is invoked using the method invocation expression m1(). If m1 is a private member of the class T where the invocation expression occurs, then the implementation in class T is selected at run-time regardless of the run-time type of the object. If the non-static method m1 is not private, then the selected implementation is determined at run-time based on the run-time type of the object. The program invokes the non-static method printS1S2 on an instance of class S, so the run-time type is S. The body of method R.printS1S2 contains two method invocation expressions, printS1() and printS2(). Since class R contains a private implementation of the instance method printS1, it is the implementation that is selected regardless of the run-time type of the object. Since printS2 is not private and not static, the selected implementation of printS2 depends on the run-time type of the object. The method printS1S2 is invoked on an instance of class S; so the run-time type of the object is S, and the implementation of printS2 declared in class S is selected.**

**11 f Compile-time error The two-parameter constructor of U does not explicitly invoke the two-parameter constructor of T; therefore, the constructor of U will try to invoke a no-parameter constructor of T, but none exists.**

**12 a Prints: ABCI Suppose that a class extends a superclass, X, or implements an interface, X. The field access expression ((X)this).hiddenField is used to access the hidden field, hiddenField, that is accessible within the superclass or interface, X.**

**13 b Prints: DE At run-time, the actual field that is accessed depends on the compile-time type of the reference--not the run-time type of the object. The compile-time type of the reference x in the name x.s1 is D; so the selected field is the one declared in class D. A non-static method is selected based on the run-time type of the object--not the compile-time type of the reference. The same reference variable x is used in the method invocation expression x.getS1(), and the compile-time type is still D. At run-time, the actual type of the object is E; so the selected method is the one declared in class E.**

**14 c Prints: CBA Class C extends B, and B extends A. The static method C.m hides method B.m, and B.m hides A.m. In the method invocation expression c.m(), the compile-time type of the reference c is C. A static method is invoked based on the compile-time type of the reference; so the method invocation expression c.m() invokes the method m declared in class C. The compile-time type of the reference b is B; so the method invocation expression b.m() invokes the method m declared in class B. The compile-time type of the reference a is A; so the method invocation expression a.m() invokes the method m declared in class A.**

**15 b Prints: ABC In all three cases, the object passed to method m1 is an instance of class C; however, the type of the reference is different for each method. Since fields are accessed based on the type of the reference, the value printed by each method is different even though the same instance is used for each method invocation.**

**16 e The relationship between Cat and Dog is an example of an appropriate use of inheritance. An appropriate inheritance relationship includes a subclass that "is-a" special kind of the superclass. The relationship between the Dog subclass and the Pet superclass is an example of an appropriate inheritance relationship, because a Dog "is-a" Pet. The relationship between the Cat subclass and the Dog superclass is not an example of an appropriate use of inheritance, because a Cat is not a special kind of a Dog. The goal of the OO paradigm is to develop software models that are accurate and reusable. If the software model is not accurate, then it probably is not reusable and the goals of the OO paradigm are not achieved. Code reuse and maintenance becomes increasingly difficult when inheritance is used to model inappropriate relationships. For example, suppose that somebody implements a herdSheep method in the Dog class. The Cat subclass would inherit the method and suddenly each instance of Cat would acquire the unwanted capability to make an attempt to herd sheep. It is difficult to imagine that a Cat would perform well in that role, so additional maintenance would be required to resolve the problem.**

**17 b Prints: SA SB CA CB The static initializer of the super class runs before the static initializer of the subclass. The body of the superclass constructor runs to completion before the body of the subclass constructor runs to completion.**

**18 b Prints: D,C,B,A A field of a superclass can be inherited by a subclass if the superclass field is not private and not hidden by a field declaration in the subclass and is accessible to code in the subclass. The field D.s1 hides C.s1, and C.s1 hides B.s1, and B.s1 hides A.s1. The keyword this serves as a reference to the object on which a method has been invoked. In the field access expression this.s1 appearing on line 1, the keyword this denotes a reference to the object of type D on which method m1 has been invoked. In the field access expression ((C)this).s1 appearing on line 2, the reference denoted by the keyword this is cast from type D to type C. The field that is accessed at run-time depends on the compile-time type of the reference; so the field access expression ((C)this).s1 refers the the variable s1 declared in class C.**

**19 b Prints: B,SuperB,A,SuperA The expression A.this.s1 is an example of a qualified this expression. It accesses the variable s1 declared in class A. The expression A.super.s1 is equivalent to ((SuperA)A.this).s1. It accesses the variable s1 declared in class SuperA.**

**20 b Prints: DDDD The instance method that is invoked depends on the run-time type of the object--not the compile-time type of the reference. In each case, the method m1 is invoked on an object of type D; so the implementation of m1 in type D is selected each time.**

**21 d Compile-time error at 2 The method invocation expression a1.m2() generates a compile-time error, because the named method, m2, is declared in class B, but the reference is of the superclass type, A. The reference a1 is of type A; so a1 is able to access only those methods that are declared in class A and subclass methods that override those of class A. Only one method, m1, is declared in A; so a reference of type A can be used to invoke A.m1 or an overriding implementation of m1 that is declared in a subclass of A. Class B extends A and overrides method m1. A reference of type A can be used to invoke method m1 on an instance of type B. Class B declares an additional method, m2, that does not override a method of class A; so a reference of type A can not invoke B.m2.**

# 6.3Nested Classes:

# Question 1

class A {

```
  private static String s1 = "s1";
  final String s2 = "s2";
  A () { new Z("s5","s6");}
  class Z {
    final String s3 = "s3";
    String s4 = "s4";
    Z (final String s5, String s6) {
      System.out.print(???);
  }}
  public static void main(String args[]) {new A();}
}
```

Which variable can not be substituted for ??? without causing a compile-time error?

a. s1
b. s2
c. s3
d. s4
e. s5
f. s6
g. None of the above

---

# Question 2

```
class B {
  private static String s1 = "s1";
  final String s2 = "s2";
  B () {new Z("s5","s6");}
  static class Z {
    final String s3 = "s3";
    static String s4 = "s4";
    Z (final String s5, String s6) {
      System.out.print(???);
  }}
  public static void main(String args[]) {new B();}
}
```

Which variable can not be substituted for ??? without causing a compile-time error?

a. s1
b. s2
c. s3
d. s4
e. s5
f. s6
g. None of the above

---

# Question 3

```
class C {
  private static String s1 = "s1";
  String s2 = "s2";
  C() {m1("s5","s6");}
  void m1(final String s5, String s6) {
    final String s3 = "s3"; String s4 = "s4";
    class Z {Z() {System.out.print(???);}}
    new Z();
  }
  public static void main(String args[]) {new C();}
}
```

Which variable names can be substituted for ??? without causing a compile-time error?

a. s1
b. s2
c. s3
d. s4
e. s5
f. s6

---

# Question 4

```
class D {
  D() {System.out.print("D");}
```

```
  class Z {Z(){System.out.print("Z");}}
  public static void main(String args[]) {
    new D.Z();
}}
```

What is the result of attempting to compile and run the program?

a. Prints: D
b. Prints: Z
c. Prints: DZ
d. Prints: ZD
e. Run-time error
f. Compile-time error
g. None of the above

---

# Question 5

```
class E {
  E() {System.out.print("E");}
  static class Z {Z(){System.out.print("Z");}}
  public static void main(String args[]) {
    new E.Z();
}}
```

What is the result of attempting to compile and run the program?

a. Prints: E
b. Prints: Z
c. Prints: EZ
d. Prints: ZE
e. Run-time error
f. Compile-time error
g. None of the above

---

# Question 6

```
class F {
  public void m1() {Z.m1();}      // 1
  private static class Y {
    private static void m1() {
      System.out.print("Y.m1 ");
  }}
  private static class Z {
    private static void m1(){
      System.out.print("Z.m1 ");
      Y.m1();                     // 2
  }}
  public static void main(String[] args) {
    new F().m1();
}}
```

What is the result of attempting to compile and run the program?

a. Compile-time error at line 1
b. Compile-time error at line 2
c. Run-time error at line 1
d. Run-time error at line 2
e. Prints: Z.m1 Y.m1
f. None of the above

---

# Question 7

```
class G {
  final String s1 = "G.s1";
  class Z {
    String s1;
    void m1() {System.out.println(???);}
  }
  public static void main(String args[]) {
    G g = new G(); g.new Z().m1();
}}
```

Which name or expression could be used in place of ??? to cause the program to print "G.s1"?

a. s1
b. G.s1
c. ((G)this).s1
d. G.this.s1
e. G.super.s1
f. None of the above

---

# Question 8

```
class Outer {
  static class StaticNested {
    static final int a = 25; // 1
    static final int b;      // 2
    static int c;            // 3
    int d;                   // 4
    static {b = 42;}         // 5
  }
  class NonStaticInner {
    static final int e = 25; // 6
    static final int f;      // 7
    static int g;            // 8
    int h;                   // 9
    static {f = 42;}         // 10
}}
```

Compile-time errors are generated at which lines?

a. 1
b. 2
c. 3
d. 4
e. 5
f. 6
g. 7
h. 8
i. 9

j. 10

---

# Question 9

```
class Red {
  private static final int a = 10;  // 1
  protected static int b = 20;      // 2
  int c = 30;                       // 3
  static class StaticNested {
    int d = a;                      // 4
    int e = b;                      // 5
    int f = c;                      // 6
  }
  class NonStaticInner {
    int g = a;                      // 7
    int h = b;                      // 8
    int i = c;                      // 9
}}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. 5
f. 6
g. 7
h. 8
i. 9

---

# Question 10

```
class Red {
  static class StaticNested {interface ABC {}}  // 1
  class NonStaticInner {interface DEF {}}       // 2
```

```
  interface GHI {}                    // 3
}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. None of the above

---

# Question 11

```
class A {
  private static int counter;
  public static int getCounter(){return counter++;}
  private static int innerCounter;
  public static int getInnerCounter(){return innerCounter++;}
  private String name;
  A() {name = "A" + getCounter();}
  class B {
    private String name;
    B() {
      name = "B" + getInnerCounter();
      System.out.print(A.this.name + name); // 1
  }}
  public static void main(String[] args) {
    new A().new B();  // 2
    A a1 = new A();
    a1.new B();       // 3
    a1.new B();       // 4
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A0B0A1B1A1B2
b. Prints: A0B0A1B1A2B2
c. Compile-time error at line 1
d. Compile-time error at line 2

e. Compile-time error at line 3
f. Compile-time error at line 4
g. Other compile-time error.
h. Run-time error
i. None of the above

---

# Question 12

```
class A {
  private static int counter;
  public static int getCounter(){return counter++;}
  private static int innerCounter;
  public static int getInnerCounter(){return innerCounter++;}
  private String name;
  A() {name = "A" + getCounter();}
  class B {
    private String name;
    B() {
      name = "B" + getInnerCounter();
      System.out.print(A.this.name + name); // 1
  }}
  void m1() {new A().new B();}  // 2
  void m2() {this.new B();}     // 3
  void m3() {new B();}          // 4
  public static void main(String[] args) {
    A a1 = new A();
    a1.m1(); a1.m2(); a1.m3();
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A0B0A1B1A1B2
b. Prints: A0B0A1B1A2B2
c. Prints: A1B0A0B1A0B2
d. Compile-time error at line 1
e. Compile-time error at line 2
f. Compile-time error at line 3
g. Compile-time error at line 4

h. Other compile-time error.
i. Run-time error
j. None of the above

---

# Question 13

```
class A {
  private static int counter;
  public static int getCounter(){return counter++;}
  private static int innerCounter;
  public static int getInnerCounter(){return innerCounter++;}
  private String name;
  A() {name = "A" + getCounter();}
  class B {
    private String name;
    B() {
      name = "B" + getInnerCounter();
      System.out.print(A.this.name + name);
  }}
  void m1() {new A().new B();}  // 1
  void m2() {new A.B();}        // 2
  void m3() {new B();}          // 3
  public static void main(String[] args) {
    A a1 = new A();
    a1.m1(); a1.m2(); a1.m3();
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A0B0A1B1A1B2
b. Prints: A0B0A1B1A2B2
c. Prints: A1B0A0B1A0B2
d. Compile-time error at line 1
e. Compile-time error at line 2
f. Compile-time error at line 3
g. Compile-time error at line 4
h. Other compile-time error.
i. Run-time error

j. None of the above

---

# Question 14

```
class A {
  private static int counter;
  public static int getCounter(){return counter++;}
  private static int innerCounter;
  public static int getInnerCounter(){return innerCounter++;}
  private String name;
  A() {name = "A" + getCounter();}
  class B {
    private String name;
    B() {
      name = "B" + getInnerCounter();
      System.out.print(A.this.name + name); // 1
  }}
  static void m1() {new A().new B();}  // 2
  static void m2() {this.new B();}     // 3
  static void m3() {new B();}          // 4
  public static void main(String[] args) {
    m1(); m2(); m3();
}}
```

What are the results of attempting to compile and run the program?

a. Prints: A0B0A1B1A1B2
b. Prints: A0B0A1B1A2B2
c. Prints: A1B0A0B1A0B2
d. Compile-time error at line 1
e. Compile-time error at line 2
f. Compile-time error at line 3
g. Compile-time error at line 4

**No. Answer Remark**

**1 g None of the above Please note that this question asks which variable can NOT be substituted. Class Z is a non-static member class of class A; therefore, all of the fields and methods of class A (static, non-static, and private) are available to class Z.**

**2 b s2 Please note that this question asks which variable can NOT be substituted. Class Z is a static member class of class B; therefore, all of the static fields and methods of the enclosing class B are available to Z. For example, from within the static member class Z, the static field s1 of the enclosing class B can be accessed using the simple name s1. The simple name s1 does not need to be qualified with a reference to an instance of the enclosing class B, because s1 is not associated with a particular instance of the enclosing class. In contrast, non-static fields and methods of the enclosing class B are associated with a particular instance of class B. From the static context of class Z, the instance fields and methods of the enclosing class B can be accessed only if the simple name of the instance field or method is qualified with a reference to a specific instance of class B. Suppose a reference variable r1 refers to an instance of the enclosing class B. Then the instance member s2 of the enclosing class instance referenced by r1 could be accessed using the expression r1.s2.**

**3 a b c e s1 s2 s3 s5 Class Z is a local class defined within the code block of method m1 of class C. All of the fields and methods of class C (static, non-static, and private) are available to Class Z. Additionally, the parameters of method m1 that are declared final are available to class Z. If a final local variable declaration appears before the declaration of class Z, then it is also available to class Z. Parameter s6 and variable s4 are not final and are therefore not available to class Z. There are at least two reasons why non-final variables are not available to a local inner class. The first reason is because the life cycle of a local variable might be shorter than that of a local class. A local variable lives on the stack and disappears as soon as the method is exited. A local class, however, might continue to exist even after the method has been exited. The second reason is due to multithreading issues. Suppose a local class extends the Thread class or implements the Runnable interface, and it is used to spawn a new Thread from within the method m1. If the local variables of method m1 were available to the local class, then a mechanism for synchronizing access to the variables would be required.**

**4 f Compile-time error An instance of class Z must be associated with an enclosing instance of class D. In a static context, an unqualified class instance creation expression of the form new ClassType(ArgumentListopt) ClassBodyopt can not be used to create an instance of an inner class. Instead, a qualified class instance creation expression of the form Reference.new Identifier(ArgumentListopt) ClassBodyopt is required to create an association between an instance of the enclosing class and the new instance of the inner class. The reference could be provided by a reference variable of the type of the enclosing class, or it could be provided by a class instance creation expression such as new D(). In a static context, the expression new D().new Z() can be used to create the new instance of the enclosing class D and the new instance of the inner class Z.**

**5 b Prints: Z Class Z is a static member class of class E. Static member classes are similar to ordinary top-level classes with the added advantage that all of the static fields and methods of the enclosing class (including those that are private) are available to the member class. The class instance creation expression new E.Z() creates an instance of the static nested class E.Z. The instance of the static nested class is not associated with any instance of the enclosing class, and no instance of the enclosing class is created. For that reason, only the letter "Z" is printed.**

**6 e Prints: Z.m1 Y.m1 The private fields and methods of the member classes are available to the enclosing class. The private fields and methods of the member classes are also available to other member classes.**

**7 d G.this.s1 The qualified this expression ClassName.this.name can be used to access shadowed variables declared within an enclosing class.**

**8 g h j 7 8 10 A non-static nested class is called an inner class. An inner class can not declare a static field unless it is a compile-time constant. Even though f is final, it does not have a value at compile time; so it causes a compilation error. Member variable g also causes a compile-time**

**error, because it is static. The static initializer of NonStaticInner causes a compile-time error, because inner classes (i.e. non-static nested classes) can not declare static initializers.**

**9 f 6 The non-static members of an enclosing class are not directly available to a static nested class. From within StaticNested, the non-static members of the enclosing class can not be referred to by a simple name. Instead, a qualified name is required. Suppose a reference variable r1 refers to an instance of the enclosing class Red. Then the instance member c of the enclosing class instance referenced by r1 could be accessed using the qualified name r1.c.**

**10 b 2 A member interface is implicitly static; therefore, it can not be declared as a member of a non-static nested class.**

**11 a Prints: A0B0A1B1A1B2 Class A is the enclosing class of the inner class B. An instance of class B must be associated with an enclosing instance of class A. In a static context such as the main method, instantiation of a named inner class requires the use of a qualified class instance creation expression of the form Reference.new Identifier(ArgumentListopt). The reference could be provided by a reference variable of the type of the enclosing class, or it could be provided by another class instance creation expression. At line 2, the qualified class instance creation expression new A().new B() first creates a new instance of the enclosing class A, then it creates an instance of B. The new instance of A is the first instance created; so the name is A0. The new instance of B is the first instance created; so the name is B0. At line 3, the qualified class instance creation expression a1.new B() creates an instance of B that is associated with a previously existing instance of class A that is referenced by variable a1. The instance of class A referenced by variable a1 is the second instance created so the name is A1. The new instance of B is the second instance created; so the name is B1. At line 4, a new instance of B is created and associated with the instance of A this is referenced by variable a1.**

**12 c Prints: A1B0A0B1A0B2 Class A is the enclosing class of the inner class B. An instance of class B must be associated with an enclosing instance of class A. In a static context, instantiation of a named inner class requires the use of a qualified class instance creation expression of the form Reference.new Identifier(ArgumentListopt). The reference could be provided by a reference variable of the type of the enclosing class, or it could be provided by another class instance creation expression. If the enclosing class is not an inner class, then the enclosing class could be instantiated with an unqualified class instance creation expression such as the following, new EnclosingClass(). The qualified class instance creation expression new A().new B() first creates a new instance of A, then it creates an instance of B. The new instance of A is the second instance created; so the name is A1. The new instance of B is the first instance created; so the name is B0. In the qualified class instance creation expression this.new B(), the keyword this, denotes a reference to the enclosing instance on which the method m2 has been invoked. A new instance of the enclosing class is not created; so the name of the enclosing instance is A0. The new instance of B is the second instance created; so the name is B1. Since method m3 is an instance method, the inner class B can be instantiated using the unqualified class instance creation expression new B(). The enclosing instance is the instance on which the method m3 has been invoked. It is the same instance that is referenced by the keyword this and the reference variable a1.**

**13 c Prints: A1B0A0B1A0B2 Class A is the enclosing class of the inner class B. An instance of class B must be associated with an enclosing instance of class A. In a static context, instantiation of a named inner class requires the use of a qualified class instance creation expression of the form Reference.new Identifier(ArgumentListopt). The reference could be provided by a reference variable of the type of the enclosing class, or it could be provided by another class instance creation expression. If the enclosing class is not an inner class, then the enclosing class could be instantiated with an unqualified class instance creation expression such as the following, new EnclosingClass(). At line 1, the qualified class instance creation expression new A().new B() first creates a new instance of A, then it creates an instance of B. The new instance of A is the second instance created; so the**

**name is A1. The new instance of B is the first instance created; so the name is B0. At line 2, a new instance of the named inner class B is created using the class instance creation expression new A.B(). The fully qualified name of class B is A.B. Since the class instance creation expression new A.B() appears within a method that is a member of class A, the use of the fully qualified name is unnecessary. Within method A.m2, the class instance creation expression new A.B() could be replaced by the expression new B() without changing the result. Using either expression, a new instance of class B is created without creating a new instance of class A. Instead, the new instance of class B is associated with the same instance of class A on which the method m2 has been invoked. It is the same instance of class A that is referenced by the keyword this and the reference variable a1. Since it was the first instance created, the name is A0. The new instance of B is the second instance created; so the name is B1. At line 3, a new instance of the named inner class B is created using the unqualified class instance creation expression new B(). The new instance of B is the third instance created; so the name is B2. The new instance of the inner class is associated with the same instance of the enclosing class on which the method m3 has been invoked. It is the same instance that is referenced by the keyword this and the reference variable a1. Since it was the first instance created, the name is A0.**

**14 f g Compile-time error at line 3 Compile-time error at line 4 Class A is the enclosing class of the inner class B. An instance of class B must be associated with an enclosing instance of class A. In a static context, instantiation of a named inner class requires the use of a qualified class instance creation expression of the form Reference.new Identifier(ArgumentListopt). The reference could be provided by a reference variable of the type of the enclosing class, or it could be provided by another class instance creation expression. If the enclosing class is not an inner class, then the enclosing class could be instantiated with an unqualified class instance creation expression such as the following, new EnclosingClass(). The qualified class instance creation expression new A().new B() first creates a new instance of A, then it creates an instance of B. The qualified class instance creation expression this.new B() generates a compile-time error, because the keyword this can not be used within a static method. Since methods m1, m2 and m3 are static methods, an instance of the named inner class B can not be created inside any of the three methods using an unqualified class instance creation expression of the form new ClassType(ArgumentListopt).**

# 6.4Overloading:

# Question 1

```
class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D extends C {
  void m1(D d) {System.out.print("D");}
  public static void main(String[] args) {
    A a1 = new A(); B b1 = new B(); C c1 = new C(); D d1 = new D();
    d1.m1(a1); d1.m1(b1); d1.m1(c1);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AAA
b. Prints: ABC
c. Prints: DDD
d. Prints: ABCD
e. Compile-time error
f. Run-time error
g. None of the above

---

# Question 2

```
class A {}  class B extends A {}  class C extends B {}
class D {
  void m1(A a) {System.out.print("A");}
  void m1(B b) {System.out.print("B");}
  void m1(C c) {System.out.print("C");}
  public static void main(String[] args) {
    A c1 = new C(); B c2 = new C(); C c3 = new C(); D d1 = new D();
    d1.m1(c1); d1.m1(c2); d1.m1(c3);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AAA
b. Prints: ABC
c. Prints: CCC
d. Compile-time error
e. Run-time error
f. None of the above

---

# Question 3

```
class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D {
```

```
  public static void main(String[] args) {
    A c1 = new C(); B c2 = new C(); C c3 = new C(); C c4 = new C();
    c4.m1(c1); c4.m1(c2); c4.m1(c3);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AAA
b. Prints: ABC
c. Prints: CCC
d. Compile-time error
e. Run-time error
f. None of the above

---

# Question 4

```
class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D {
  public static void main(String[] args) {
    A c1 = new C(); C c2 = new C(); c1.m1(c2);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A
b. Prints: B
c. Prints: C
d. Compile-time error
e. Run-time error
f. None of the above

---

# Question 5

```
class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D {
  public static void main(String[] args) {
    A a1 = new A(); A b1 = new B(); A c1 = new C(); C c4 = new C();
    a1.m1(c4); b1.m1(c4); c1.m1(c4);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AAA
b. Prints: ABC
c. Prints: CCC
d. Compile-time error
e. Run-time error
f. None of the above

---

# Question 6

```
class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D {
  public static void main(String[] args) {
    A a1 = new A(); B b1 = new B(); C c1 = new C(); A c2 = new C();
    c2.m1(a1); c2.m1(b1); c2.m1(c1);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AAA
b. Prints: ABC
c. Prints: CCC
d. Compile-time error
e. Run-time error
f. None of the above

# Question 7

```
class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D {
  public static void main(String[] args) {
    A a1 = new A(); B b1 = new A(); C c1 = new A(); C c2 = new C();
    c2.m1(a1); c2.m1(b1); c2.m1(c1);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AAA
b. Prints: ABC
c. Prints: CCC
d. Compile-time error
e. Run-time error
f. None of the above

---

# Question 8

```
class A {void m1(A a) {System.out.print("A");}}
class B extends A {void m1(B b) {System.out.print("B");}}
class C extends B {void m1(C c) {System.out.print("C");}}
class D {
  public static void main(String[] args) {
    A a1 = new A(); B b1 = new B(); C c1 = new C(); C c2 = new A();
    c2.m1(a1); c2.m1(b1); c2.m1(c1);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: AAA
b. Prints: ABC

c. Prints: CCC
d. Compile-time error
e. Run-time error
f. None of the above

| No. | Answer | Remark |
|---|---|---|
| 1 | b | Prints: ABC The method invocation expression d1.m1(a1) uses reference d1 of type D to invoke method m1. Since the reference d1 is of type D, the class D is searched for an applicable implementation of m1. The methods inherited from the superclasses, C, B and A, are included in the search. The argument, a1, is a variable declared with the type A; so method A.m1(A a) is invoked. |
| 2 | b | Prints: ABC Three methods overload the method name m1. Each has a single parameter of type A or B or C. For any method invocation expression of the form m1(referenceArgument), the method is selected based on the declared type of the variable referenceArgument--not the run-time type of the referenced object. The method invocation expression d1.m1(c1) uses reference d1 of type D to invoke method m1 on an instance of type D. The argument, c1, is a reference of type A and the run-time type of the referenced object is C. The argument type is determined by the declared type of the reference variable c1--not the run-time type of the object referenced by c1. The declared type of c1 is type A; so the method m1(A a) is selected. The declared type of c2 is type B; so the method invocation expression d1.m1(c2) invokes method m1(B b). The declared type of c3 is type C; so the method invocation expression d1.m1(c3) invokes method m1(C c). |
| 3 | b | Prints: ABC Three methods overload the method name m1. Each has a single parameter of type A or B or C. For any method invocation expression of the form m1(referenceArgument), the method is selected based on the declared type of the variable referenceArgument--not the run-time type of the referenced object. The method invocation expression c4.m1(c1) uses reference c4 of type C to invoke method m1 on an instance of type C. The argument, c1, is a reference of type A and the run-time type of the referenced object is C. The argument type is determined by the declared type of the reference variable c1--not the run-time type of the object referenced by c1. The declared type of c1 is type A; so the method A.m1(A a) is selected. The declared type of c2 is type B; so the method invocation expression c4.m1(c2) invokes method B.m1(B b). The declared type of c3 is type C; so the method invocation expression c4.m1(c3) invokes method C.m1(C c). |
| 4 | a | Prints: A The reference c1 is of the superclass type, A; so it can be used to invoke only the method m1 declared in class A. The methods that overload the method name m1 in the subclasses, B and C, can not be invoked using the reference c1. A method invocation conversion promotes the argument referenced by c2 from type C to type A, and the method declared in class A is executed. Class A declares only one method, m1. The single parameter is of type A. Class B inherits the method declared in class A and overloads the method name with a new method that has a single parameter of type B. Both methods sharing the overloaded name, m1, can be invoked using a reference of type B; however, a reference of type A can be used to invoke only the method declared in class A. Class C inherits the methods declared in classes A and B and overloads the method name with a new method that has a single parameter of type C. All three methods sharing the overloaded name, m1, can be invoked using a reference of type C; however, a reference of type B can be used to invoke only the method declared in class B and the method declared in the superclass A. The method invocation expression c1.m1(c2) uses reference c1 of type A to invoke method m1. Since the reference c1 is of type A, the search for an applicable implementation of m1 is limited to class A. The subclasses, B and C, will not be searched; so the overloading methods declared in the subclasses can not be invoked using a reference of the superclass type. |

5 a Prints: AAA The declared type of the reference variables, a1, b1 and c1, is the superclass type, A; so the three reference variables can be used to invoke only the method m1(A a) that is declared in the superclass, A. The methods that overload the method name m1 in the subclasses, B and C, can not be invoked using a reference variable of the superclass type, A. A method invocation conversion promotes the argument referenced by c4 from type C to type A, and the method declared in class A is executed.

6 a Prints: AAA The reference c2 is of the superclass type, A; so it can be used to invoke only the method, m1, declared in class A. The methods that overload the method name m1 in the subclasses, B and C, can not be invoked using the reference c2.

7 d Compile-time error The declarations of b1 and c1 cause compile-time errors, because a reference of a subclass type can not refer to an instance of the superclass type.

8 d Compile-time error The declaration of c2 causes a compile-time error, because a reference of a subclass type can not refer to an instance of the superclass class.

# 7.Threads

# Exam1:

# Question 1

Which of the following methods are members of the Object class?

a. Join
b. notify
c. notifyAll
d. Run
e. sleep
f. Start
g. yield
h. Wait

---

# Question 2

Which of the following methods are static members of the Thread class?

a. Join
b. Notify
c. NotifyAll
d. Run
e. sleep
f. start
g. Yield
h. Wait

---

## Question 3

Which of the following methods are deprecated members of the Thread class?

a. Join
b. Notify
c. NotifyAll
d. Resume
e. Run
f. Sleep
g. Start
h. Stop
i. Suspend
j. Yield
k. Wait

---

## Question 4

Which of the following methods name the InterruptedException in its throws clause?

a. Join
b. Notify

c. NotifyAll
d. Run
e. Sleep
f. Start
g. Yield
h. Wait

---

## Question 5

A timeout argument can be passed to which of the following methods?

a. Join
b. Notify
c. NotifyAll
d. Run
e. Sleep
f. Start
g. Yield
h. Wait

---

## Question 6

Which of the following instance methods should only be called by a thread that holds the lock of the instance on which the method is invoked?

a. join
b. Notify
c. notifyAll
d. run
e. start
f. Wait

---

## Question 7

Which of the following is a checked exception?

a. IllegalMonitorStateException
b. IllegalThreadStateException
c. IllegalArgumentException
d. InterruptedException
e. None of the above

---

## Question 8

Which kind of variable would you prefer to synchronize on?

a. A member variable of a primitive type
b. A member variable that is an object reference
c. A method local variable that is a reference to an instance that is created within the method
d. None of the above

---

## Question 9

synchronized (expression) block

The synchronized statement has the form shown above. Which of the following are true statements?

a. A compile-time error occurs if the expression produces a value of any reference type
b. A compile-time error occurs if the expression produces a value of any primitive type
c. A compile-time error does not occur if the expression is of type boolean
d. The sychronized block may be processed normally if the expression is null
e. If execution of the block completes normally, then the lock is released
f. If execution of the block completes abruptly, then the lock is released

g. A thread can hold more than one lock at a time
h. Synchronized statements can be nested
i. Synchronized statements with identical expressions can be nested

---

## Question 10

Which of the following is a true statement?

a. The process of executing a synchronized method requires the thread to acquire a lock
b. Any overriding method of a synchronized method is implicitly synchronized
c. If any method in a class is synchronized, then the class itself must also be declared using the synchronized modifier
d. If a thread invokes a static synchronized method on an instance of class A, then the thread must acquire the lock of that instance of class A
e. None of the above

---

## Question 11

Which of the following thread state transitions model the lifecycle of a thread?

a. The Dead state to the Ready state
b. The Ready state to the Not-Runnable state
c. The Ready state to the Running state
d. The Running state to the Not-Runnable state
e. The Running state to the Ready state
f. The Not-Runnable state to the Ready state
g. The Not-Runnable state to the Running state

---

## Question 12

Which of the following are true statements?

a. The Thread.yield method might cause the thread to move to the Not-Runnable state
b. The Thread.yield method might cause the thread to move to the Ready state
c. The same thread might continue to run after calling the Thread.yield method
d. The Thread.yield method is a static method
e. The behavior of the Thread.yield method is consistent from one platform to the next
f. The Thread.sleep method causes the thread to move to the Not-Runnable state
g. The Thread.sleep method causes the thread to move to the Ready state

---

## Question 13

Which of the following will not force a thread to move into the Not-Runnable state?

a. Thread.yield method
b. Thread.sleep method
c. Thread.join method
d. Object.wait method
e. By blocking on I/O
f. Unsuccessfully attempting to acquire the lock of an object
g. None of the above

---

## Question 14

Which of the following will cause a dead thread to restart?

a. Thread.yield method
b. Thread.join method
c. Thread.start method
d. Thread.resume method
e. None of the above

---

## Question 15

When a thread is created and started, what is its initial state?

a. New
b. Ready
c. Not-Runnable
d. Runnning
e. Dead
f. None of the above

---

## Question 16

Which of the following are true statements?

a. The Thread.run method is used to start a new thread running
b. The Thread.start method causes a new thread to get ready to run at the discretion of the thread scheduler
c. The Runnable interface declares the start method
d. The Runnable interface declares the run method
e. The Thread class implements the Runnable interface
f. If an Object.notify method call appears in a synchronized block, then it must be the last method call in the block
g. No restriction is placed on the number of threads that can enter a synchronized method
h. Some implementations of the Thread.yield method will not yield to a thread of lower priority

---

## Question 17

Which of the following are true statements?

a. Thread.MAX_PRIORITY == 10
b. Thread.MAX_PRIORITY == 5
c. Thread.NORM_PRIORITY == 5

d. Thread.NORM_PRIORITY == 3
e. Thread.NORM_PRIORITY == 0
f. Thread.MIN_PRIORITY == 1
g. Thread.MIN_PRIORITY == 0
h. Thread.MIN_PRIORITY == -5
i. Thread.MIN_PRIORITY == -10

---

# Question 18

Which of the following are true statements?

a. A program will terminate only when all daemon threads stop running
b. A program will terminate only when all user threads stop running
c. A daemon thread always runs at Thread.MIN_PRIORITY
d. A thread inherits its daemon status from the thread that created it
e. The daemon status of a thread can be changed at any time using the Thread.setDaemon method
f. The Thread.setDaemon method accepts one of two argument values defined by the constants Thread.DAEMON and Thread.USER

---

# Question 19

```
class A extends Thread {
  public A(Runnable r) {super(r);}
  public void run() {System.out.print("A");}
}
class B implements Runnable {
  public void run() {System.out.print("B");}
}
class C {
  public static void main(String[] args) {
    new A(new B()).start();
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A
b. Prints: B
c. Prints: AB
d. Prints: BA
e. Compile-time error
f. Run-time error
g. None of the above

---

# Question 20

```
class A implements Runnable {
  public void run() {System.out.print(Thread.currentThread().getName());}
}
class B implements Runnable {
  public void run() {
    new A().run();
    new Thread(new A(),"T2").run();
    new Thread(new A(),"T3").start();
}}
class C {
  public static void main (String[] args) {
    new Thread(new B(),"T1").start();
}}
```

What is the result of attempting to compile and run the program?

a. Prints: T1T1T1
b. Prints: T1T1T2
c. Prints: T1T2T2
d. Prints: T1T2T3
e. Prints: T1T1T3
f. Prints: T1T3T3
g. Compile-time error
h. Run-time error
i. None of the above

---

# Question 21

```
class AnException extends Exception {}
class A extends Thread {
  public void run() throws AnException {
    System.out.print("A"); throw new AnException();
}}
class B {
  public static void main (String[] args) {
    A a = new A(); a.start(); System.out.print("B");
}}
```

What is the result of attempting to compile and run the program?

a. Prints: A
b. Prints: B
c. Prints: AB
d. Prints: BA
e. Compile-time error
f. Run-time error
g. None of the above

---

# Question 22

```
class A extends Thread {
  public void run() {System.out.print("A");}
}
class B {
  public static void main (String[] args) {
    A a = new A();
    a.start();
    a.start();  // 1
  }
}
```

What is the result of attempting to compile and run the program?

a. The program compiles and runs without error

b. The second attempt to start thread t1 is successful
c. The second attempt to start thread t1 is ignored
d. Compile-time error at marker 1
e. An IllegalThreadStateException is thrown at run-time
f. None of the above

| No. | Answer | Remark |
|---|---|---|
| 1 | b c h | notify notifyAll wait |
| 2 | e g | sleep yield |
| 3 | d h i | resume stop suspend For the purposes of the exam, you don't need to memorize the deprecated methods of the Thread class. Even though a question such as this will not be on the exam, every Java programmer should know that the deprecated methods should not be used in new programs. |
| 4 | a e h | join sleep wait |
| 5 | a e h | join sleep wait |
| 6 | b c f | notify notifyAll wait |
| 7 | d | InterruptedException The methods Object.wait, Thread.join and Thread.sleep name InterruptedException in their throws clauses. |
| 8 | b | A member variable that is an object reference Primitives don't have locks; therefore, they can not be used to synchronize threads. A method local variable that is a reference to an instance that is created within the method should not be used to synchronize threads, because each thread has its own instance of the object and lock. Synchronization on an instance that is created locally makes about as much sense as placing on your doorstep a box full of keys to the door. Each person that comes to your door would have their own copy of the key; so the lock would provide no security. |
| 9 | b e f g h i | A compile-time error occurs if the expression produces a value of any primitive type If execution of the block completes normally, then the lock is released If execution of the block completes abruptly, then the lock is released A thread can hold more than one lock at a time Synchronized statements can be nested Synchronized statements with identical expressions can be nested |
| 10 | a | The process of executing a synchronized method requires the thread to acquire a lock The synchronized modifier can not be applied to a class. A method that overrides a synchronized method does not have to be synchronized. If a thread invokes a synchronized instance method on an instance of class A, then the thread must acquire the lock of that instance of class A. The same is not true for synchronized static methods. A synchronized static method is synchronized on the lock for the Class object that represents the class for which the method is a member. |

**11 c d e f The Ready state to the Running state The Running state to the Not-Runnable state The Running state to the Ready state The Not-Runnable state to the Ready state A dead thread can not be restarted.**

**12 b c d f The Thread.yield method might cause the thread to move to the Ready state The same thread might continue to run after calling the Thread.yield method The Thread.yield method is a static method The Thread.sleep method causes the thread to move to the Not-Runnable state**

**The Thread.yield method is intended to cause the currently executing thread to move from the Running state to the Ready state and offer the thread scheduler an opportunity to allow a different thread to execute based on the discretion of the thread scheduler. The thread scheduler may select the same thread to run immediately, or it may allow a different thread to run. The Thread.yield method is a native method; so the behavior is not guaranteed to be the same on every platform. However, at least some implementations of the yield method will not yield to a thread that has a lower priority.**

**13 a Thread.yield method The Thread.yield method may cause a thread to move into the Ready state, but that state transition is not guaranteed. The JLS states that the Thread.yield method provides a hint to the thread scheduler, but the scheduler is free to interpret--or ignore--the hint as it sees fit. Nothing in the JLS suggests that the thread might move to the Not-Runnable state.**

**14 e None of the above A dead thread can not be restarted.**

**15 b Ready**

**16 b d e h The Thread.start method causes a new thread to get ready to run at the discretion of the thread scheduler The Runnable interface declares the run method The Thread class implements the Runnable interface Some implementations of the Thread.yield method will not yield to a thread of lower priority The Object.notify method can only be called by the thread that holds the lock of the object on which the method is invoked. Suppose that thread T1 enters a block that is synchronized on an object, A. Within the block, thread T1 holds the lock of A. Even if thread T1 calls the notify method immediately after entering the synchronized block, no other thread can grab the lock of object A until T1 leaves the synchronized block. For that reason, the transfer of control from thread T1 to any waiting thread can not be accelerated by moving the notify method to an earlier point in the synchronized block. The behavior of Thread.yield is platform specific. However, at least some implementations of the yield method will not yield to a thread that has a lower priority. Invoking the Thread.yield method is like offering a suggestion to the JVM to allow another thread to run. The response to the suggestion is platform specific.**

**17 a c f Thread.MAX_PRIORITY == 10 Thread.NORM_PRIORITY == 5 Thread.MIN_PRIORITY == 1**

**18 b d A program will terminate only when all user threads stop running A thread inherits its daemon status from the thread that created it**

**19 a Prints: A If a Runnable target object is passed to the constructor of the Thread class, then the Thread.run method will invoke the run method of the Runnable target. In this case, the Thread.run method is overridden by A.run. The A.run method does nothing more than print the letter A. The invocation of the A.start method inside the main method results in the invocation of A.run, and the letter A is printed. The B.run method is never invoked.**

**20 e Prints: T1T1T3 The Thread.currentThread method returns a reference to the currently executing thread. When the run method is invoked directly it does not start a new thread; so T1 is printed twice.**

**21 e Compile-time error The Runnable.run method does not have a throws clause; so any implementation of run can not throw a checked exception.**

**22 e An IllegalThreadStateException is thrown at run-time For the purposes of the exam, invoking the start method on a thread that has already been started will generate an IllegalThreadStateException. The actual behavior of the method might be different. If the start method is invoked on a thread that is already running, then an IllegalThreadStateException will probably be thrown. However, if the thread is already dead then the second attempt to start the thread will probably be ignored, and no exception will be thrown. For the purposes of the exam, the exception is always thrown in response to the second invocation of the start method. This is a case where the exam tests your knowledge of the specification and ignores the actual behavior of the 1.4 version of the JVM.**

# Exam 2:

# 7.2

# Question 1

```
class A extends Thread {
  private int i;
  public void run() {i = 1;}
  public static void main(String[] args) {
    A a = new A(); a.start(); System.out.print(a.i);
}}
```

What are the possible results of attempting to compile and run the program?

a. Prints nothing
b. Prints: 0
c. Prints: 1
d. Prints: 01
e. Prints: 10
f. Compile-time error
g. Run-time error

---

# Question 2

```
class A extends Thread {
  private int i;
  public void run() {i = 1;}
  public static void main(String[] args) {
    A a = new A(); a.run(); System.out.print(a.i);
}}
```

What is the result of attempting to compile and run the program?

a. Prints nothing
b. Prints: 0
c. Prints: 1
d. Prints: 01
e. Prints: 10
f. Compile-time error
g. Run-time error
h. None of the above

---

# Question 3

```
class A extends Thread {
  public void run() {
    try {sleep(10000);} catch (InterruptedException ie){}
  }
  public static void main(String[] args) {
    A a1 = new A();
    long startTime = System.currentTimeMillis();
    a1.start();
    System.out.print(System.currentTimeMillis() - startTime);
}}
```

What are the possible results of attempting to compile and run the program?

a. Prints a number greater than or equal to 0
b. The number printed must always be greater than 10000

c. This program will run for at least ten seconds
d. Compile-time error
e. Run-time error

---

# Question 4

Which of the following is used to force each thread to reconcile its working copy of a variable with the master copy in main memory?

a. Final
b. Static
c. synchronized
d. transient
e. volatile
f. native

---

# Question 5

```
class A extends Thread {
  public void run() {
    synchronized (this) {
      try {wait(5000);} catch (InterruptedException ie){}
  }}
  public static void main(String[] args) {
    A a1 = new A();
    long startTime = System.currentTimeMillis();
    a1.start();
    System.out.print(System.currentTimeMillis() - startTime + ",");
    try {a1.join(6000);} catch (InterruptedException ie) {}
    System.out.print(System.currentTimeMillis() - startTime);
}}
```

What are the possible results of attempting to compile and run the program?

a. The first number printed is greater than or equal to 0
b. The first number printed must always be greater than 5000

c. The second number printed must always be greater than 5000
d. The second number printed must always be greater than 6000
e. The synchronized block inside the run method is not necessary
f. Compile-time error
g. Run-time error

---

# Question 6

```
class A extends Thread {
  String[] sa;
  public A(String[] sa) {this.sa = sa;}
  public void run() {
    synchronized (sa) {System.out.print(sa[0] + sa[1] + sa[2]);}
}}
class B {
  private static String[] sa = new String[]{"X","Y","Z"};
  public static void main (String[] args) {
    synchronized (sa) {
      Thread t1 = new A(sa); t1.start();
      sa[0] = "A"; sa[1] = "B"; sa[2] = "C";
}}}
```

What is the result of attempting to compile and run the program?

a. Prints: XYZ
b. Prints: AYZ
c. Prints: ABZ
d. Prints: ABC
e. Compile-time error
f. Run-time error
g. None of the above

---

# Question 7

```
class A extends Thread {
```

```
  String[] sa;
  public A(String[] sa) {this.sa = sa;}
  public void run() {
    synchronized (sa) {
      while (!sa[0].equals("Done")) {
        try {sa.wait();} catch (InterruptedException ie) {}
    }}
    System.out.print(sa[1] + sa[2] + sa[3]);
}}
class B {
  private static String[] sa = new String[]{"Not Done","X","Y","Z"};
  public static void main (String[] args) {
    Thread t1 = new A(sa); t1.start();
    synchronized (sa) {
      sa[0] = "Done";
      sa[1] = "A"; sa[2] = "B"; sa[3] = "C";
      sa.notify();
}}}
```

What is the result of attempting to compile and run the program?

a. Prints: XYZ
b. Prints: AYZ
c. Prints: ABZ
d. Prints: ABC
e. Compile-time error
f. Run-time error
g. None of the above

---

# Question 8

Which of the following are true statements?

a. The Thread.join method is static
b. The Thread.join method is always invoked on an instance of Thread
c. The Thread.join method causes the current thread to wait for the referenced thread to die
d. The Thread.join method declares an InterruptedException in the throws clause

e. The Thread.join method accepts a timeout value as an argument
f. The timeout value sets the minimum time that the current thread will wait for the death of the referenced thread
g. Thread.join will return immediately if the timeout value is zero
h. A timeout of zero will allow Thread.join to wait forever if necessary

---

# Question 9

Which of the following allows a thread t1 to become the holder of the lock of object obj1.

a. By blocking on I/O
b. By entering a synchronized instance method of the obj1
c. By invoking the wait method on the object
d. By entering the body of a block that is synchronized on obj1
e. By entering a synchronized static method of the obj1
f. By invoking the notify method on obj1

---

# Question 10

After invoking the wait method on an object, obj1, a thread, T1, will remain in the wait set of obj1 until which of the following occurs?

a. Another thread invokes the notify method on the object, obj1, and T1 is selected to move out of the wait set
b. Another thread invokes the notifyAll method on the object
c. Another thread invokes the resume method on thread T1
d. Another thread interrupts thread T1
e. The priority of thread T1 is increased
f. A specified timeout period has elapsed
g. Another thread invokes the join method on thread T1

---

# Question 11

```
class A implements Runnable{public void run() {}}
class B {
  public static void main(String[] args) {
    Thread t1 = new Thread();            // 1
    Thread t2 = new Thread(new A());     // 2
    Thread t3 = new Thread(new A(), "A"); // 3
    Thread t4 = new Thread("A");         // 4
}}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. None of the above

---

# Question 12

```
class A implements Runnable{public void run() {}}
class B {
  public static void main(String[] args) {
    Thread t1 = new Thread();            // 1
    Thread t2 = new Thread(new A());     // 2
    Thread t3 = new Thread("A", new A()); // 3
    Thread t4 = new Thread("A");         // 4
}}
```

A compile-time error is generated at which line?

a. 1
b. 2
c. 3
d. 4
e. None of the above

---

# Question 13

```
class A extends Thread {
  private boolean done;
  public void setDone(boolean done) {this.done = done;}
  public void run() {
    synchronized (this) {
      while (!done) {try {wait();} catch (InterruptedException ie){}}
  }}
  public static void main(String[] args) {
    A a1 = new A();
    long startTime = System.currentTimeMillis();
    a1.start();
    System.out.print(System.currentTimeMillis() - startTime);
}}
```

Which is a possible result of attempting to compile and run the program?

a. The number printed is greater than or equal to 0
b. The synchronized block inside the run method is not necessary
c. This program runs to completion after the elapsed time is printed
d. Compile-time error
e. Run-time error
f. None of the above

---

# Question 14

```
class A extends Thread {
  public void run() {
    synchronized (this) {
      try {wait();} catch (InterruptedException ie){}
  }}
  public static void main(String[] args) {
    A a1 = new A(); a1.setDaemon(true);
    long startTime = System.currentTimeMillis();
    a1.start();
    System.out.print(System.currentTimeMillis() - startTime + ",");
```

```
}}
```

Which is a possible result of attempting to compile and run the program?

a. The number printed is greater than or equal to 0
b. The synchronized block inside the run method is not necessary
c. Thread a1 waits forever and the program runs forever
d. Compile-time error
e. Run-time error
f. None of the above

---

# Question 15

```
class A extends Thread {
  private Object obj;
  public A(Object obj) {this.obj = obj;}
  public void run() {
    try {
      synchronized (obj) {obj.wait();}
    } catch (InterruptedException ie) {}
    System.out.print(Thread.currentThread().getName());
}}
class B {
  private void m1() {
    for (int i = 0; i < 10; i++) {
      A t1 = new A(this);
      t1.setName(String.valueOf(i)); t1.setDaemon(true); t1.start();
    }
    synchronized (this) {notifyAll();}
  }
  public static void main(String[] args) {new B().m1();}
}
```

What are the possible results of attempting to compile and run the program?

a. All of the numbers 0 through 9 must always be printed
b. Some or all of the numbers 0 through 9 could be printed
c. Nothing is printed

d. Run-time error

---

# Question 16

```
class C extends Thread {
  private static String[] sa = new String[]{"Not Done","X","Y","Z"};
  public void run() {
    synchronized (sa) {
      while (!sa[0].equals("Done")) {
        try {sa.wait();} catch (InterruptedException ie) {}
    }}
    System.out.print(sa[1] + sa[2] + sa[3]);
  }
  public static void main (String[] args) {
    start();
    synchronized (sa) {
      sa[0] = "Done";
      sa[1] = "A"; sa[2] = "B"; sa[3] = "C";
      sa.notify();
}}}
```

Which is a possible result of attempting to compile and run the program?

a. Prints: XYZ
b. Prints: AYZ
c. Prints: ABZ
d. Prints: ABC
e. Compile-time error
f. Run-time error
g. None of the above

---

# Question 17

```
class C extends Thread {
  private static String[] sa = new String[]{"Not Done","X","Y","Z"};
  public void run() {
    synchronized (this) {
```

```
    while (!sa[0].equals("Done")) {
      try {wait();} catch (InterruptedException ie) {}
   }}
   System.out.print(sa[1] + sa[2] + sa[3]);
  }
  void m1() {
    start();
    synchronized (this) {
      sa[0] = "Done";
      sa[1] = "A"; sa[2] = "B"; sa[3] = "C";
  }}
  public static void main (String[] args) {
    new C().m1(); notify();
}}
```

Which is a possible result of attempting to compile and run the program?

a. Prints: XYZ
b. Prints: AYZ
c. Prints: ABZ
d. Prints: ABC
e. Compile-time error
f. Run-time error
g. None of the above

---

# Question 18

```
class A extends Thread {
  public void run() {
    long startTime = System.currentTimeMillis();
    long endTime = startTime + 10000;
    while (System.currentTimeMillis() < endTime) {
      yield();
  }}
  public static void main(String[] args) {
    A a1 = new A();
    long startTime = System.currentTimeMillis();
    a1.start(); sleep(1000); a1.interrupt(); a1.join();
    System.out.print(System.currentTimeMillis() - startTime);
}}
```

Which is a possible result of attempting to compile and run the program?

a. Prints a number that is less than 1000
b. Prints a number between 1000 and 9999
c. Prints a number larger than 10000
d. Compile-time error
e. Run-time error
f. None of the above

---

# Question 19

```
class A extends Thread {
  public void run() {System.out.print("A");}
}
class B {
  public static void main (String[] args) {
    A a = new A(); a.start();
    try {
      a.join();
    } catch (InterruptedException ie) {ie.printStackTrace();}
    a.start();  // 1
}}
```

What is the result of attempting to compile and run the program?

a. The program compiles and runs without error
b. The second attempt to start thread t1 is successful
c. The second attempt to start thread t1 is ignored
d. Compile-time error at marker 1
e. An IllegalThreadStateException is thrown at run-time
f. None of the above

---

# Question 20

```
class A extends Thread {
```

```
  private static B b = new B();
  private String s1;
  public void run() {System.out.print(b.m1(s1));}
  A(String threadName, String s1) {
    super(threadName); this.s1 = s1;
  }
  public static void main (String[] args) {
    A a = new A("T1","A"), b = new A("T2","B"); a.start(); b.start();
}}
class B {
  private String s1;
  public synchronized String m1(String s) {
    s1 = s;
    try {Thread.sleep(1);} catch (InterruptedException ie) {}
    return "["+Thread.currentThread().getName()+","+s1+"]";
}}
```

What are the possible results of attempting to compile and run the program?

a. Prints nothing
b. Prints: [T1,A][T2,B]
c. Prints: [T1,B][T2,B]
d. Prints: [T2,B][T1,A]
e. Prints: [T2,A][T1,A]
f. Compile-time error
g. Run-time error

---

# Question 21

```
class A extends Thread {
  static long startTime;
  public void run() {
    for (int i = 0; i < 99999; i++) {Math.sin(i);}
    String name = Thread.currentThread().getName();
    long time = System.currentTimeMillis();
    System.out.println(name + " done at " + (time - startTime));
  }
  public static void main(String[] args) {
    A t1 = new A(); A t2 = new A();
    t1.setName("T1"); t2.setName("T2");
```

```
   t1.setPriority(Thread.MIN_PRIORITY);
   t2.setPriority(Thread.MAX_PRIORITY);
   startTime = System.currentTimeMillis();
   t1.start(); t2.start();
}}
```

Which of the following is a true statement?

a. The priority assigned to thread T2 is greater than the priority assigned to T1
b. Java guarantees that thread T2 will get more CPU time than T1
c. Java guarantess that thread T2 will run to completion before T1
d. None of the above

---

| No. | Answer | Remark |
|---|---|---|
| **1** | **b c** | **Prints: 0 Prints: 1 The new thread is started before the print statement, but there is no guarantee that the new thread will run before the print statement is processed. The guarantee could be provided by placing the method invocation expression a.join() before the print statement, but the invocation of the join method does not appear in the program. If the new thread runs before the print statement is processed, then 1 is printed. Otherwise, 0 is printed.** |
| **2** | **c** | **Prints: 1 The a.run() method was called instead of a.start(); so the entire program runs as a single thread, and a.run() is guaranteed to complete before the print statement is called.** |
| **3** | **a c** | **Prints a number greater than or equal to 0 This program will run for at least ten seconds Thread a1 will run for at least ten seconds, but the main method is likely to run to completion very quickly. The start method will return without waiting for thread a1 to complete. Since thread a1 immediately goes to sleep the thread that is processing the main method has an opportunity to complete the main method quickly. The number printed in the main method can be as small as zero.** |
| **4** | **e** | **volatile A field might be shared between two or more threads. Each thread is allowed to maintain a working copy of the field. If the threads do not reconcile the working copies then each might be working with a different value. The volatile modifier is used to force each thread to reconcile its working copy of the field with the master copy in main memory.** |
| **5** | **a c** | **The first number printed is greater than or equal to 0 The second number printed must always be greater than 5000 The notify method is never invoked on thread a1; so it will sleep for at least five seconds. The invocation of the join method forces the main thread to wait for the completion of thread a1. The argument of 6000 will allow the main thread to wait for six seconds if necessary, but we know that thread a1 will complete in only five seconds. The first number printed will be greater than or equal to zero, and the second number will be greater than or equal to 5000. The synchronized block is necessary, because it is necessary to hold the lock of an object when the wait method is invoked.** |

**6 d Prints: ABC The block inside the main method is synchronized on the String array object sa. Inside the block, a new thread t1 is started and will run at the discretion of the thread scheduler. The A.run method also contains a block that is synchronized on the String array object sa. Even if the thread scheduler moves thread t1 into the Running state, it will block while attempting to acquire the lock of the String array object sa. Thread t1 will continue to block until the synchronized block in the B.main method runs to completion. At that time, the contents of the String array object have all been updated.**

**7 d Prints: ABC Inside the main method, thread t1 is started and will move into the Running state at the discretion of the thread scheduler. The A.run method invokes the wait method on the String array object sa causing the thread to block until another thread invokes the sa.notify method. Before the B.main method invokes sa.notify, all of the elements of the String array object sa have already been updated.**

**8 b c d e h The Thread.join method is always invoked on an instance of Thread The Thread.join method causes the current thread to wait for the referenced thread to die The Thread.join method declares an InterruptedException in the throws clause The Thread.join method accepts a timeout value as an argument A timeout of zero will allow Thread.join to wait forever if necessary The Thread.join method is not static. Thread.join is always invoked on an instance of Thread. Thread.join causes the current thread to wait until the referenced thread has died. The maximum time limit to wait for the death of the referenced thread can be specified in milliseconds by an argument. Thread.join will throw an InterruptedException if the interrupt method is invoked on the current Thread.**

**9 b d By entering a synchronized instance method of the obj1 By entering the body of a block that is synchronized on obj1 Blocking on I/O or invoking the Thread.sleep or Object.wait method causes a thread to enter the Not-Runnable state. Invoking the notify method on an object wakes up a thread that is waiting on the object. The thread that invokes wait or notify on an object should already hold the lock of the object. Invoking the wait or notify method does not cause the thread to hold the lock. Static methods are synchronized on the lock of the Class object of the class. Instance methods are synchronized on the lock of the instance of the class.**

**10 a b d f Another thread invokes the notify method on the object, obj1, and T1 is selected to move out of the wait set Another thread invokes the notifyAll method on the object Another thread interrupts thread T1 A specified timeout period has elapsed**

**11 e None of the above All of the class instance creation expressions are legal. The String instance A is the name of the Thread. Yes, the exam requires you to memorize the Thread constructor signatures.**

**12 c 3 The position of the arguments have been reversed in the constructor on line 3. The Runnable argument should appear before the thread name argument. Yes, the exam requires you to memorize the Thread constructor signatures.**

**13 a The number printed is greater than or equal to 0 The main thread invokes the start method on thread a1. There is no way to predict when the new thread will start to run. At some point in time, the main thread will proceed to the print statement where the value of the startTime variable is subtracted from the current time. The value printed will be greater than or equal to one. At some point in time, thread a1 will begin to run and it will invoke the wait method. Since no other thread invokes the notify method on a1, it will wait forever, and the program will never run to completion.**

**14 a The number printed is greater than or equal to 0 The a1 thread is a daemon thread; so the program can run to completion even if thread a1 is still running, waiting or sleeping. The notify method is never invoked on thread a1. If thread a1 were not a daemon thread, then the program would wait forever. However, the program will run to completion without waiting for a1.**

**15 b c Some or all of the numbers 0 through 9 could be printed Nothing is printed All of the threads started in method B.m1 are daemon threads; so the program can run to completion even if some or all of the daemon threads have not run.**

**16 e Compile-time error Remember that the Thread.start method is an instance method and can not be invoked from a static context.**

**17 e Compile-time error Remember that the Object.notify method is an instance method and can not be invoked from a static context. Also, the thread that invokes the notify method on an object must hold the lock of the object.**

**18 d Compile-time error Both the sleep and join methods declare an InterruptedException that must be caught or declared in the throws clause of A.main.**

**19 e An IllegalThreadStateException is thrown at run-time For the purposes of the exam, invoking the start method on a thread that has already been started will generate an IllegalThreadStateException. The actual behavior of Java might be different. If the start method is invoked on a thread that is already running, then an IllegalThreadStateException will probably be thrown. However, if the thread is already dead then the second attempt to start the thread will probably be ignored, and no exception will be thrown. However, for the purposes of the exam, the exception is always thrown in response to the second invocation of the start method. This is a case where the exam tests your knowledge of the specification of the Thread.start method and ignores the actual behavior of the 1.4 version of the JVM. The Thread.join method is included here to verify that the thread is already dead before the start method is invoked the second time. If this code is executed using the 1.4 version of the JVM, the exception will not be thrown. However, for the purposes of the exam, the exception is always thrown. The real exam question will probably not include the invocation of the join method.**

**20 b d Prints: [T1,A][T2,B] Prints: [T2,B][T1,A] Since method m1 is synchronized, it is guaranteed that no more than one thread will execute the method at any one time. Even though the start method is invoked on thread T1 first, there is no guarantee that it will actually begin to run first.**

**21 a The priority assigned to thread T2 is greater than the priority assigned to T1 The Java Language Specification suggests that higher priority threads should be given preference over lower priority threads, but explicitly states that the preference is not a guarantee. It is very important to remember that no guarantee exists.**

**Section 9: The Collections Framework**

Collections

Question 1

Which implementation of the List interface produces the slowest access to an element in the middle of the list by means of an index?

a. Vector

b. ArrayList

c. LinkedList

d. None of the above

Question 2

```
import java.util.*;
class GFC100 {
  public static void main (String args[]) {
    Object a1 = new LinkedList(), b1 = new TreeSet();
    Object c1 = new TreeMap();
    System.out.print((a1 instanceof Collection)+",");
    System.out.print((b1 instanceof Collection)+",");
    System.out.print(c1 instanceof Collection);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false

b. Prints: false,false,true

c. Prints: false,true,false

d. Prints: false,true,true

e. Prints: true,false,false

f. Prints: true,false,true

g. Prints: true,true,false

h. Prints: true,true,true

i. None of the above

Question 3

- Each element must be unique.
- Contains no duplicate elements.
- Elements are not key/value pairs.
- Accessing an element can be almost as fast as performing a similar operation on an array.

Which of these classes provides the specified features?

a. LinkedList

b. TreeMap

c. TreeSet

d. HashMap

e. HashSet

f. LinkedHashMap

g. Hashtable

h. None of the above

Question 4

```
import java.util.*;
class GFC101 {
  public static void main (String args[]) {
    Object a1 = new HashMap(), b1 = new ArrayList();
    Object c1 = new HashSet();
    System.out.print((a1 instanceof Collection)+",");
    System.out.print((b1 instanceof Collection)+",");
    System.out.print(c1 instanceof Collection);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false

b. Prints: false,false,true

c. Prints: false,true,false

d. Prints: false,true,true

e. Prints: true,false,false

f. Prints: true,false,true

g. Prints: true,true,false

h. Prints: true,true,true

i. None of the above

Question 5

- Entries are organized as key/value pairs.
- Duplicate entries replace old entries.

Which interface of the java.util package offers the specified behavior?

a. List

b. Map

c. Set

d. None of the above

Question 6

```
import java.util.*;
class GFC102 {
  public static void main (String args[]) {
    Object a = new HashSet();
    System.out.print((a instanceof Set)+",");
```

```
    System.out.print(a instanceof SortedSet);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false

b. Prints: false,true

c. Prints: true,false

d. Prints: true,true

e. None of the above

Question 7

Which implementation of the List interface provides for the fastest insertion of a new element into the middle of the list?

a. Vector

b. ArrayList

c. LinkedList

d. None of the above

Question 8

```
import java.util.*;
class GFC103 {
  public static void main (String args[]) {
```

```
    Object a1 = new TreeSet();
   System.out.print((a1 instanceof Set)+",");
   System.out.print(a1 instanceof SortedSet);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false

b. Prints: false,true

c. Prints: true,false

d. Prints: true,true

e. None of the above

Question 9

- Stores key/value pairs.
- Duplicate entries replace old entries.
- Entries are sorted using a Comparator or the Comparable interface.

Which of these classes provides the specified features?

a. LinkedList

b. TreeMap

c. TreeSet

d. HashMap

e. HashSet

f. Hashtable

g. None of the above

Question 10

```
import java.util.*;
class GFC104 {
  public static void main (String args[]) {
    LinkedList a1 = new LinkedList();
    ArrayList b1 = new ArrayList();
    Vector c1 = new Vector();
    System.out.print((a1 instanceof List)+",");
    System.out.print((b1 instanceof List)+",");
    System.out.print(c1 instanceof List);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false

b. Prints: false,false,true

c. Prints: false,true,false

d. Prints: false,true,true

e. Prints: true,false,false

f. Prints: true,false,true

g. Prints: true,true,false

h. Prints: true,true,true

i. None of the above

Question 11

- Entries are not organized as key/value pairs.
- Duplicate entries are rejected.

Which interface of the java.util package offers the specified behavior?

a. List

b. Map

c. Set

d. None of the above

Question 12

```
import java.util.*;
class GFC105 {
  public static void main (String args[]) {
    Object a = new HashSet(), b = new HashMap();
    Object c = new Hashtable();
```

```
    System.out.print((a instanceof Collection)+",");
    System.out.print((b instanceof Collection)+",");
    System.out.print(c instanceof Collection);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false

b. Prints: false,false,true

c. Prints: false,true,false

d. Prints: false,true,true

e. Prints: true,false,false

f. Prints: true,false,true

g. Prints: true,true,false

h. Prints: true,true,true

i. None of the above

Question 13

- Elements are not key/value pairs.
- Contains no duplicate elements.
- The entries can be sorted using the Comparable interface.

Which of these classes provides the specified features?

a. LinkedList

b. TreeMap

c. TreeSet

d. HashMap

e. HashSet

f. Hashtable

g. None of the above

Question 14

```
import java.util.*;
class GFC106 {
  public static void main (String args[]) {
    Object a = new HashSet(), b = new HashMap();
    Object c = new Hashtable();
    System.out.print((a instanceof Map)+",");
    System.out.print((b instanceof Map)+",");
    System.out.print(c instanceof Map);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false

b. Prints: false,false,true

c. Prints: false,true,false

d. Prints: false,true,true

e. Prints: true,false,false

f. Prints: true,false,true

g. Prints: true,true,false

h. Prints: true,true,true

i. None of the above

Question 15

- Stores key/value pairs.
- Allows null elements, keys, and values.
- Duplicate entries replace old entries.
- Entries are not sorted.

Which of these classes provides the specified features?

a. LinkedList

b. TreeMap

c. TreeSet

d. HashMap

e. HashSet

f. Hashtable

g. None of the above

Question 16

```
import java.util.*;
class GFC107 {
  public static void main (String args[]) {
    Object a = new HashSet(), b = new HashMap();
    Object c = new Hashtable();
    System.out.print((a instanceof Cloneable)+",");
    System.out.print((b instanceof Cloneable)+",");
    System.out.print(c instanceof Cloneable);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false

b. Prints: false,false,true

c. Prints: false,true,false

d. Prints: false,true,true

e. Prints: true,false,false

f. Prints: true,false,true

g. Prints: true,true,false

h. Prints: true,true,true

i. None of the above

Question 17

- Entries are not organized as key/value pairs.
- Generally accepts duplicate elements.
- Entries may be accessed by means of an index.

Which interface of the java.util package offers the specified behavior?

a. List

b. Map

c. Set

d. None of the above

Question 18

```
import java.util.*;
import java.io.Serializable;
class GFC108 {
  public static void main (String args[]) {
    HashMap a = new HashMap();
    boolean b1, b2, b3;
    b1 = (a instanceof Cloneable) & (a instanceof Serializable);
```

```
    b2 = a instanceof Map;
    b3 = a instanceof Collection;
    System.out.print(b1 + "," + b2 + "," + b3);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false

b. Prints: false,false,true

c. Prints: false,true,false

d. Prints: false,true,true

e. Prints: true,false,false

f. Prints: true,false,true

g. Prints: true,true,false

h. Prints: true,true,true

i. None of the above

Question 19

Which of the following classes allow unsynchronized read operations by multiple threads?

a. Vector

b. Hashtable

c. TreeMap

d. TreeSet

e. HashMap

f. HashSet

Question 20

- Entries are organized as key/value pairs.
- Duplicate entries replace old entries.
- Entries are sorted using a Comparator or the Comparable interface.

Which interface of the java.util package offers the specified behavior?

a. List

b. Map

c. Set

d. SortedSet

e. SortedMap

f. None of the above

Question 21

```
import java.util.*;
class GFC110 {
  public static void main (String[] args) {
    Object m = new LinkedHashMap();
```

```
    System.out.print((m instanceof Collection)+",");
    System.out.print((m instanceof Map)+",");
    System.out.print(m instanceof List);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false

b. Prints: false,false,true

c. Prints: false,true,false

d. Prints: false,true,true

e. Prints: true,false,false

f. Prints: true,false,true

g. Prints: true,true,false

h. Prints: true,true,true

i. None of the above

Question 22

```
import java.util.*;
class GFC109 {
  public static void main (String[] args) {
    Object v = new Vector();
```

```
    System.out.print((v instanceof Collections)+",");
    System.out.print((v instanceof Arrays)+",");
    System.out.print(v instanceof List);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false

b. Prints: false,false,true

c. Prints: false,true,false

d. Prints: false,true,true

e. Prints: true,false,false

f. Prints: true,false,true

g. Prints: true,true,false

h. Prints: true,true,true

i. None of the above

| No. | Answer | | Remark |
|---|---|---|---|
| 1 | c | LinkedList | ArrayList and Vector both use an array to store the elements of the list; so access to any element using an index is very fast. A LinkedList is implemented using a doubly linked list; so access to an element requires the list to be traversed using the links. |
| 2 | g | Prints: true,true,false | The List and Set interfaces extend the Collection interface; so both List and Set could be cast to type Collection without generating a ClassCastException. Therefore, the first two of the three relational expressions return true. The Map interface does not extend Collection. The reference variable c1 refers to an instance of TreeMap; so the relational expression c1 instanceof Collection returns the value false. |
| 3 | e | HashSet | The elements of a Map are key/value pairs; so a Map is not a good choice. A List generally accepts duplicate |

| No. | Answer | | Remark |
|---|---|---|---|
| | | | elements. A Set stores a collection of unique elements. Any attempt to store a duplicate element in a Set is rejected. Adding and removing an element in a TreeSet involves walking the tree to determine the location of the element. A HashSet stores the elements in a hashtable; so elements in a HashSet can be accessed almost as quickly as elements in an array as long as the hash function disperses the elements properly. Although the LinkedHashSet is not among the answer options it could arguably satisfy the requirements. However, the put and remove methods of the LinkedHashSet are a little slower than the same methods of the HashSet due to the need to maintain the linked list through the elements of the LinkedHashSet. |
| 4 | d | Prints: false,true,true | The Map interface does not extend Collection. The reference variable a1 refers to an instance of HashMap; so the relational expression a1 instanceof Collection returns the value false. The List and Set interfaces extend the Collection interface; so both List and Set could be cast to type Collection without generating a ClassCastException. Therefore, the second and third relational expressions return true. |
| 5 | b | Map | The List and Set interfaces do not support key/value pairs. A list generally allows duplicate entries. A Set rejects duplicate entries. |
| 6 | c | Prints: true,false | HashSet implements the Set interface, but not the SortedSet interface. |
| 7 | c | LinkedList | ArrayList and Vector both use an array to store the elements of the list. When an element is inserted into the middle of the list the elements that follow the insertion point must be shifted to make room for the new element. The LinkedList is implemented using a doubly linked list; an insertion requires only the updating of the links at the point of insertion. Therefore, the LinkedList allows for fast insertions and deletions. |
| 8 | d | Prints: true,true | TreeSet implements the Set interface and the SortedSet interface. |
| 9 | b | TreeMap | The requirement to store key/value pairs is directly satisfied by a concrete implementation of the Map interface. The List and Set interfaces recognize objects, but do not recognize keys and values. TreeMap and TreeSet store elements in a sorted order based on the key, but the TreeSet does not support key/value pairs. |
| 10 | h | Prints: true,true,true | The 1.2 version of Java introduced the updated Vector class that implements the List interface. |
| 11 | c | Set | The Map interface organizes entries as key/value pairs. A list generally allows duplicate entries. A Set rejects duplicate entries. |
| 12 | e | Prints: true,false,false | HashSet is a subclass of AbstractSet and AbstractCollection; therefore, it implements the Collection interface. HashMap and Hashtable do not implement the Collection interface. Instead, HashMap extends AbstractMap and implements the Map interface. Hashtable extends Dictionary and implements the Map interface. |
| 13 | c | TreeSet | The elements are not key/value pairs; so a Map is not a good choice. A List generally accepts duplicate elements. A Set stores a collection of unique objects; so any attempt to store a duplicate object is rejected. TreeSet stores elements in an order that is determined either by a Comparator or by the Comparable interface. |
| 14 | d | Prints: false,true,true | HashSet implements the Set interface, but not the Map interface. HashMap extends AbstractMap and implements the Map interface. Hashtable extends Dictionary and implements the Map interface. |

| No. | Answer | | Remark |
|---|---|---|---|
| 15 | d | HashMap | The requirement to store key/value pairs is directly satisfied by a concrete implementation of the Map interface. The List and Set interfaces recognize objects, but do not recognize keys and values. The requirement to allow null elements is not satisfied by a Hashtable. TreeMap and TreeSet store elements in a sorted order based on the key. |
| 16 | h | Prints: true,true,true | All three implement Cloneable. |
| 17 | a | List | The Map interface organizes entries as key/value pairs. A list generally allows duplicate entries. A Set rejects duplicate entries. A List allows entries to be accessed using an index. |
| 18 | g | Prints: true,true,false | HashMap does not implement the Collection interface. |
| 19 | c<br>d<br>e f | TreeMap TreeSet<br>HashMap HashSet | The Vector and Hashtable methods are synchronized and do not allow for simultaneous access by multiple threads. The concrete subclasses of the AbstractList, AbstractMap, and AbstractSet classes allow for unsynchronized read operations by multiple threads. Additionally, the sychronized wrapper methods of the Collections class allow for the instantiation of a Collection, List, Map, Set, SortedMap, or SortedSet with synchronized methods. If simultaneous read and write operations are necessary then a synchronized instance should be used. |
| 20 | e | SortedMap | The List and Set interfaces do not support key/value pairs. A list generally allows duplicate entries. A Set rejects duplicate entries. A Map organizes the entries as key/value pairs. The SortedMap is similar to a Map except that the ordering of the elements is determined by a Comparator or the Comparable interface. |
| 21 | c | Prints: false,true,false | LinkedHashMap does not implement the Collection interface or the List interface. |
| 22 | b | Prints: false,false,true | The Collections class is not the same as the Collection interface. The Collections class contains a variety of methods used to work with collections. For example, Collections.shuffle is used to randomly shuffle the elements of a Collection. Similarly, the Arrays class provides utility methods for working with arrays. |

Question 1

- Entries are not organized as key/value pairs.
- Duplicate entries are rejected.
- Entries are sorted using a Comparator or the Comparable interface.

Which interface of the java.util package offers the specified behavior?

a. List
b. Map
c. Set
d. SortedSet
e. SortedMap
f. None of the above

---

Question 2

```
import java.util.*;
class GFC111 {
  public static void main (String[] args) {
    Object m = new LinkedHashMap();
    System.out.print((m instanceof Collection)+",");
    System.out.print((m instanceof Map)+",");
    System.out.print(m instanceof HashMap);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false
b. Prints: false,false,true
c. Prints: false,true,false
d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true
g. Prints: true,true,false
h. Prints: true,true,true
i. None of the above

---

Question 3

```
import java.util.*;
```

```
class GFC112 {
  public static void main (String[] args) {
    Object m = new LinkedHashSet();
    System.out.print((m instanceof Collection)+",");
    System.out.print((m instanceof Set)+",");
    System.out.print(m instanceof List);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false
b. Prints: false,false,true
c. Prints: false,true,false
d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true
g. Prints: true,true,false
h. Prints: true,true,true
i. None of the above

---

Question 4

```
import java.util.*;
class GFC113 {
  public static void main (String[] args) {
    Object m = new LinkedHashSet();
    System.out.print((m instanceof Collection)+",");
    System.out.print((m instanceof Set)+",");
    System.out.print(m instanceof HashSet);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false
b. Prints: false,false,true

c. Prints: false,true,false
d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true
g. Prints: true,true,false
h. Prints: true,true,true
i. None of the above

---

Question 5

```
import java.util.*;
class GFC116 {
  public static void main (String[] args) {
    Object x = new Vector().elements();
    System.out.print((x instanceof Enumeration)+",");
    System.out.print((x instanceof Iterator)+",");
    System.out.print(x instanceof ListIterator);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false
b. Prints: false,false,true
c. Prints: false,true,false
d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true
g. Prints: true,true,false
h. Prints: true,true,true
i. None of the above

---

Question 6

```
import java.util.*;
class GFC117 {
  public static void main (String[] args) {
    Object i1 = new HashMap(), i2 = new TreeMap();
    System.out.print((i1 instanceof SortedMap)+",");
    System.out.print((i2 instanceof SortedMap)+",");
    System.out.print(i1 instanceof Collection);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false
b. Prints: false,false,true
c. Prints: false,true,false
d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true
g. Prints: true,true,false
h. Prints: true,true,true
i. None of the above

---

Question 7

```
import java.util.*;
class GFC118 {
  public static void main (String[] args) {
    Object i = new ArrayList().listIterator();
    System.out.print((i instanceof List)+",");
    System.out.print((i instanceof Iterator)+",");
    System.out.print(i instanceof ListIterator);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false

b. Prints: false,false,true
c. Prints: false,true,false
d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true
g. Prints: true,true,false
h. Prints: true,true,true
i. None of the above

---

Question 8

Which of the following classes allow elements to be accessed in the order that they were added?

a. HashMap
b. HashSet
c. Hashtable
d. TreeMap
e. TreeSet
f. LinkedHashMap
g. LinkedHashSet

---

Question 9

Which of the following are true statements?

a. The Enumeration interface was introduced with the collections framework with Java 1.2.
b. The Enumeration interface declares only two methods: hasMoreElements and nextElement.
c. The Iterator interface extends the Enumeration interface.
d. The Iterator interface declares a total of three methods.

---

Question 10

Which of the following is a true statement?

a. The Iterator interface declares only two methods: hasMoreElements and nextElement.
b. The ListIterator interface extends both the List and Iterator interfaces.
c. The ListIterator interface was introduced with Java 1.2 to replace the older Iterator interface that was released with Java 1.0.
d. The ListIterator interface declares only three methods: hasNext, next and remove.
e. None of the above.

---

Question 11

- Stores key/value pairs.
- Allows null elements, keys, and values.
- Duplicate entries replace old entries.
- Entries are not sorted using a Comparator or the Comparable interface.
- The iteration order is unspecified.

Which of these classes provides the specified features?

a. LinkedList
b. LinkedHashMap
c. LinkedHashSet
d. TreeMap
e. TreeSet
f. HashMap
g. HashSet
h. Hashtable
i. None of the above

---

Question 12

- Stores key/value pairs.

- Does not allow null elements, keys, and values.

Which of these classes provides the specified features?

a. LinkedList
b. LinkedHashMap
c. LinkedHashSet
d. TreeMap
e. TreeSet
f. HashMap
g. HashSet
h. Hashtable
i. None of the above

---

Question 13

- Stores key/value pairs.
- Duplicate entries replace old entries.
- Provides constant-time performance for the add, contains and remove operations.

Which of these classes provides the specified features?

a. LinkedHashMap
b. LinkedHashSet
c. LinkedList
d. TreeMap
e. TreeSet
f. HashMap
g. HashSet
h. Hashtable

---

Question 14

```
import java.util.*;
class GFC114 {
  public static void main (String[] args) {
    Object a = new ArrayList();
    System.out.print((a instanceof Collections)+",");
    System.out.print((a instanceof Arrays)+",");
    System.out.print(a instanceof List);
}}
```

What is the result of attempting to compile and run the program?

a. Prints: false,false,false
b. Prints: false,false,true
c. Prints: false,true,false
d. Prints: false,true,true
e. Prints: true,false,false
f. Prints: true,false,true
g. Prints: true,true,false
h. Prints: true,true,true
i. None of the above

---

Question 15

- Each element must be unique.
- Contains no duplicate elements.
- Elements are not key/value pairs.
- Entries are not sorted using a Comparator or the Comparable interface.
- The iteration order is determined by the insertion order.

Which of these classes provides the specified features?

a. HashMap
b. HashSet
c. Hashtable
d. LinkedHashMap
e. LinkedHashSet
f. LinkedList
g. TreeMap
h. TreeSet
i. None of the above

---

Question 16

Suppose that you would like to create a new instance of a class that implements the Set interface, and you would like the new instance to be initialized with the elements of an existing Set. If you would like the iteration order of the new Set to be the same as that of the existing Set, then which concrete implementation of the Set interface should be used for the new instance?

a. Hashtable
b. HashMap
c. HashSet
d. LinkedHashSet
e. TreeMap
f. TreeSet
g. None of the above.

---

Question 17

Suppose that you would like to create a new instance of a class that implements the Map interface, and you would like the new instance to be initialized with the elements of an existing Map. If you would like the iteration order of the new Map to be the same as that of the existing Map, then which concrete implementation of the Map interface should be used for the new instance?

a. Hashtable
b. HashSet

c. HashMap
d. TreeMap
e. TreeSet
f. LinkedHashMap
g. None of the above.

---

Question 18

- Stores key/value pairs.
- Allows null elements, keys, and values.
- Duplicate entries replace old entries.
- The least recently used element can be removed automatically when a new element is added.

Which of these classes provides the specified features?

a. LinkedHashMap
b. LinkedHashSet
c. LinkedList
d. TreeMap
e. TreeSet
f. HashMap
g. HashSet
h. Hashtable
i. None of the above

---

Question 19

Which of the following classes would provide the most efficient implementation of a First In First Out queue?

a. ArrayList
b. LinkedHashMap

c. LinkedHashSet
d. LinkedList
e. HashMap
f. HashSet
g. Hashtable
h. TreeMap
i. TreeSet
j. Vector
k. None of the above

---

Question 20

In addition to implementing the List interface, which of the following also provides methods to get, add and remove elements from the head and tail of the list without specifying an index?

a. Collection
b. ArrayList
c. LinkedList
d. List
e. Vector
f. None of the above

---

Question 21

Which of the following is a true statement?

a. All implementations of the List interface provide fast random access.
b. A LinkedList provides faster random access than an ArrayList.
c. The LinkedList implements the RandomAccess interface.
d. Each collection that implements the List interface must also implement the RandomAccess interface.
e. The RandomAccess interface declares methods that use of an index argument to access elements of the List.
f. The RandomAccess interface declares the next and hasNext methods.

g. None of the above.

---

Question 22

Which of the following is not a true statement?

a. The Iterator interface declares only three methods: hasNext, next and remove.
b. The ListIterator interface extends both the List and Iterator interfaces.
c. The ListIterator interface provides forward and backward iteration capabilities.
d. The ListIterator interface provides the ability to modify the List during iteration.
e. The ListIterator interface provides the ability to determine its position in the List.
f. None of the above.

No. Answer Remark

1 d SortedSet The Map interface organizes entries as key/value pairs. A list generally allows duplicate entries. A Set rejects duplicate entries. The SortedSet is similar to a Set except that the ordering of the elements is determined by a Comparator or the Comparable interface.

2 d Prints: false,true,true LinkedHashMap does not implement the Collection interface. LinkedHashMap extends HashMap and implements Map, Cloneable and Serializable.

3 g Prints: true,true,false LinkedHashSet does not implement the List interface. LinkedHashSet extends HashSet and implements Collection, Set, Cloneable and Serializable.

4 h Prints: true,true,true LinkedHashSet is a subclass of HashSet; therefore, it is also a subclass of AbstractSet, and it implements the Collection interface.

5 e Prints: true,false,false The Vector.elements method returns an Enumeration over the elements of the Vector. The Vector class implements the List interface and extends AbstractList; so it is also possible to obtain an Iterator over a Vector by invoking the iterator or listIterator method.

6 c Prints: false,true,false Both HashMap and TreeMap are subclasses of type AbstractMap; so both implement the Map interface. Neither implements the Collection interface. The TreeMap implements the SortedMap interface, but HashMap does not.

7 d Prints: false,true,true ListIterator extends Iterator.

8 f g LinkedHashMap LinkedHashSet The HashMap, HashSet and Hashtable classes are all implemented with an internal hashtable that organizes the elements in buckets according to the hashcode and not according to the order of insertion. The LinkedHashMap and LinkedHashSet classes also use an internal hashtable, and both also maintain a linked list through all of the elements. The order of the list is determined by the order of insertion. Optionally, the LinkedHashMap can maintain the order of the list based on the time of the most recent access of each element. The TreeMap and TreeSet classes are both implemented using a tree structure that is ordered based on a Comparator or the Comparable interface.

9 b d The Enumeration interface declares only two methods: hasMoreElements and nextElement. The Iterator interface declares a total of three methods. The Enumeration interface was introduced with Java 1.0 to provide an easy means of moving through the elements of a Vector or the keys or values of a Hashtable. The Iterator interface was introduced with the collections framework with Java 1.2. The Iterator interface declares three methods: hasNext, next and remove. The first two methods, hasNext and next, are similar to the two methods declared in the Enumeration interface, hasMoreElements and nextElement. The third method of the Iterator interface, remove, provides new functionality relative to the Enumeration interface.

10 e None of the above. The Iterator interface declares three methods: hasNext, next and remove. The ListIterator interface was introduced in Java 1.2 along with the Iterator interface. The ListIterator interface extends the Iterator interface and declares additional methods to provide forward and backward iteration capabilities, List modification capabilities and the ability to determine the position of the iterator in the List. The ListIterator interface does not extend the List interface.

11 f HashMap The requirement to store key/value pairs is directly satisfied by a concrete implementation of the Map interface. The List and Set interfaces recognize objects, but do not recognize keys and values. The requirement to allow null elements is not satisfied by a Hashtable. TreeMap and TreeSet store elements in a sorted order based on the key. The iteration order of LinkedHashMap and LinkedHashSet is not unspecified. By default, the iteration order of LinkedHashMap and LinkedHashSet is based on the order in which elements were inserted. Optionally, the iteration order of the LinkedHashMap can be set to the order in which the elements were last accessed.

12 h Hashtable The requirement to store key/value pairs is directly satisfied by a Hashtable or any concrete implementation of the Map interface. The List and Set interfaces recognize objects, but do not recognize keys and values. The requirement to NOT allow null elements is satisfied by Hashtable, but not by HashMap or any of the other Collection implementations that were introduced with Java 1.2 and later.

13 a f h LinkedHashMap HashMap Hashtable The requirement to store key/value pairs is directly satisfied by a concrete implementation of the Map interface. The List and Set interfaces recognize objects, but do not recognize keys and values. The Hashtable, HashMap and LinkedHashMap classes store elements in a hashtable. Elements are accessed using a hashcode that identifies the bucket that contains the element. Access time is therefore not dependent on the number of buckets. As long as the hashcode methods of the elements are properly implemented, the time required to access an element in a hashtable remains constant as the number of buckets in the hashtable grows. In contrast, the TreeMap and TreeSet classes store elements in a sorted order in a tree structure. Access to any element requires walking the tree; so access time depends on the size of the tree.

14 b Prints: false,false,true The Collections class is not the same as the Collection interface. The Collections class contains a variety of methods used to work with collections. For example, Collections.shuffle is used to randomly shuffle the elements of a Collection. Similarly, the Arrays class provides utility methods for working with arrays.

15 e LinkedHashSet The elements of a Map are key/value pairs; so a Map is not a good choice. A List generally accepts duplicate elements. A Set stores a collection of unique elements. Any attempt to store a duplicate element in a Set is rejected. TreeSet stores elements in a sorted order based on the key. HashSet does not sort the elements based on the key. The iteration order of LinkedHashMap and LinkedHashSet is clearly defined. By default, the iteration order of LinkedHashMap and LinkedHashSet is based on the order in which elements were inserted. While a LinkedHashSet rejects duplicate entries, the LinkedHashMap allows duplicate entries to replace old entries.

16 d LinkedHashSet The iteration order of a Set is the order in which an iterator moves through the elements of the Set. The iteration order of a LinkedHashSet is determined by the order in which elements are inserted. When a reference to an existing Set is passed as an argument to the constructor of LinkedHashSet, the Collection.addAll method will add the elements of the existing Set to the new instance. Since the iteration order of the LinkedHashSet is determined by the order of insertion, the iteration order of the new LinkedHashSet must be the same as the iteration order of the old Set.

17 f LinkedHashMap The iteration order of a Map is the order in which an iterator moves through the elements of the Map. The iteration order of a LinkedHashMap is determined by the order in which elements are inserted. When a reference to an existing Map is passed as an argument to the constructor of LinkedHashMap, the Collection.addAll method will add the elements of the existing Map to the new instance. 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 iteration order of the old Map.

18 a LinkedHashMap The requirement to store key/value pairs is directly satisfied by a concrete implementation of the Map interface. The List and Set interfaces recognize objects, but do not recognize keys and values. The requirement to allow null elements is not satisfied by a Hashtable. The LinkedHashMap offers the option to remove the least recently used (LRU) element when a new element is added. The LinkedHashSet does not offer the LRU option.

19 d LinkedList A stack or queue must be implemented using a data structure that stores the elements based on the order of insertion. Any data structure that is implemented using a hashtable is not a good choice. The ArrayList and Vector are both implemented using an internal array. Although an array allows elements to be easily organized based on the order of insertion, an array does not allow the list of elements to be easily shifted in memory as elements are appended to the tail of the list and removed from the head of the list. The LinkedList is implemented using a doubly linked list that allows elements to be easily appended to the tail of the list, and removed from the head of the list.

20 c LinkedList The LinkedList class provides methods such as addFirst, addLast, getFirst, getLast, removeFirst and removeLast that facilitate the implementation of stacks and queues.

21 g None of the above. The RandomAccess interface is a marker interface; so it does not declare any methods. Its purpose is to provide information about the RandomAccess capabilities of a List implementation. Generic list algorithms can check to see if an instance of a List implements the RandomAccess marker interface. If not, then the algorithm can avoid operations that require fast random access. Both Vector and ArrayList implement the RandomAccess interface. LinkedList does not implement RandomAccess.

22 b The ListIterator interface extends both the List and Iterator interfaces. The ListIterator interface extends the Iterator interface and declares additional methods to provide forward and backward iteration capabilities, List modification capabilities and the ability to determine the position of the iterator in the List.

Hashcodes

Question 1

Suppose that an instance of class C has legal implementations of the hashCode and equals methods. Within any one execution of the Java application, the hash code contract requires that each invocation of the hashCode method on the same instance of class C must consistently return the same result as long as the fields used for the equals comparison remain unchanged.

a. false

b. true

Question 2

If two instances of a class type are equal according to the equals method, then the same integer value must be returned by the hashCode method of the two objects.

a. false

b. true

Question 3

If two instances of a class type are not equal according to the equals method, then the same integer value must not be returned by the hashCode method of the two objects.

a. false

b. true

Question 4

```
class A {
  static void m1 (B a, B b, B c, B d, B e, B f, B g, B h) {
    if (a.equals(b)) {System.out.print("A");}
    if (!c.equals(d)) {System.out.print("B");}
    if (e.hashCode() == f.hashCode()) {System.out.print("C");}
    if (g.hashCode() != h.hashCode()) {System.out.print("D");}
}}
```

Suppose that method m1 is invoked with eight instances of the same class and the output is ABCD. If the B.equals and B.hashCode methods are implemented according to the hash code contract, then which of the following statements must always be true?

a. (a.hashCode() == b.hashCode())

b. (c.hashCode() != d.hashCode())

c. (e.equals(f))

d. (!g.equals(h))

Question 5

```
class B {
  private int i1;
  public int hashCode() {return 1;}
}
class C {
  private int i1;
  public int hashCode() {return -1;}
}
class D {
  private int i1;
  public int hashCode() {return i1;}
}
```

Suppose that the equals method of classes B, C and D all make use of the value of the int variable, i1. Which class has a hashCode method that is not consistent with the hash code contract?

a. B

b. C

c. D

d. None of the above

Question 6

Which of the following classes override both the equals and hashCode methods?

a. java.lang.Byte

b. java.lang.Integer

c. java.util.Vector

d. java.lang.String

e. java.lang.StringBuffer

Question 7

```
class A {
  int i1, i2;
  public void setI1(int i) {i1 = i;}
  public int getI1() {return i1;}
  public void setI2(int i) {i2 = i;}
  public int getI2() {return i2;}
  public A(int ii1, int ii2) {i1 = ii1; i2 = ii2;}
  public boolean equals(Object obj) {
    if (obj instanceof A) {
      return (i1 == ((A)obj).getI1());
    }
    return false;
  }
```

```
  public int hashCode() {
    // Insert statement here.
}}
```

Which of the following statements could be inserted at the specified location without violating the hash code contract?

a. return 31;

b. return getI1();

c. return getI2();

d. return 31 * getI1() + getI2();

Question 8

```
class A {
  int i1, i2;
  public void setI1(int i) {i1 = i;}
  public int getI1() {return i1;}
  public void setI2(int i) {i2 = i;}
  public int getI2() {return i2;}
  public A(int ii1, int ii2) {i1 = ii1; i2 = ii2;}
  public boolean equals(Object obj) {
    if (obj instanceof A) {
      return (i1 == ((A)obj).getI1()) & (i2 == ((A)obj).getI2());
```

```
    }
    return false;
  }
  public int hashCode() {
    // Insert statement here.
}}
```

If inserted at the specified location, which of the following statements would produce the most efficient hashCode method?

a. return 31;

b. return getI1();

c. return getI2();

d. return getI1() + getI2();

e. return 31 * getI1() + getI2();

f. None of the above

1 b true

2 b true If two objects are equal according to the equals method, then the hashcodes produced by the hashCode method must also be equal. If two objects are not equal according to the equals method, then the hashcodes may or may not be equal. It is preferable that unequal objects have different hashcodes, but that is not always possible. Since the hash code value is a 32 bit primitive int, it is not possible to produce a unique hash code for each value of a primitive long.

3 a false If two objects are equal according to the equals method, then the hashcodes must also be equal. If two objects are not equal according to the equals method, then the hashcodes may or may not be equal. It is preferable that unequal objects have different hashcodes, but that is

not always possible. Since the hash code value is a 32 bit primitive int, it is not possible to produce a unique hash code for each value of a primitive long.

4 a d (a.hashCode() == b.hashCode()) (!g.equals(h)) If two objects are equal according to the equals method, then the hashcodes must also be equal. If two objects are not equal according to the equals method, then the hashcodes may or may not be equal. If two objects have the same hash code, then the objects may or may not be equal. If two objects have different hashcodes, then the objects must not be equal.

5 d None of the above Suppose that the hashCode method is invoked on the same object more than once during the same execution of a Java application. If no information used in the equals comparison is modified, then each invocation of the hashCode method must produce the same hash code value. The hashCode methods of classes B and C will always return the same value during an execution of the Java application and are therefore consistent with the hash code contract. Even so, the hashCode methods of classes B and C are not efficient, because they will cause hashtables to place every instance of classes B and C in the same bucket. The hashCode method of class D is appropriate and will allow a hash table to operate efficiently.

6 a b c d java.lang.Byte java.lang.Integer java.util.Vector java.lang.String The wrapper classes and the collection classes override the equals and hashCode methods. The String class overrides the equals and hashCode methods, but StringBuffer does not.

7 a b return 31; return getI1(); A hashCode method that returns the constant value 31 is consistent with the hash code contract. Even so, a hashCode method that returns the same value regardless of the internal state of the object is not very good, because it will cause hashtables to place every instance of the class in the same bucket. If the equals method determines that two instances are equal, then the two instances must produce the same hash code. For that reason, the hash code must not be calculated using fields that are not used to determine equality. In this case, the equals method determines equality based on the value of i1. The value of i2 is not used to determine equality; therefore, i2 can not be used to calculate the hash code.

8 e return 31 * getI1() + getI2(); All of the statements would produce a hashCode method that is consistent with the hash code contract. The expression 31 * getI1() + getI2() produces the most efficient hashCode method, because it is most likely to produce unique hashcodes for various combinations of i1 and i2. The expression getI1() + getI2() is less efficient, because it produces the same hash code when the values of i1 and i2 are swapped.