1. Given the declarations

```
int p = 5, q = 3;
```

which of the following expressions evaluate to 7.5?

I. `(double)p * (double)q / 2;`
II. `(double)p * (double)(q / 2);`
III. `(double)(p * q / 2);`

(A) I only
(B) II only
(C) I and II
(D) I, II, and III
(E) None of the above

2. Consider the following method:

```
public void mystery(int a, int b)
{
  System.out.print(a + " ");
  if (a <= b)
    mystery(a + 5, b - 1);
}
```

What is the output when `mystery(0, 16)` is called?

(A) `0`
(B) `0 5`
(C) `0 5 10`
(D) `0 5 10 15`
(E) `0 5 10 15 20`

3. Assuming that `c` and `d` are Boolean variables, the expression

```
!c || d
```

is equivalent to which of the following?

(A) `!(c && d)`
(B) `!(c && !d)`
(C) `c && !d`
(D) `!(c || !d)`
(E) `!(!c && d)`

4. Suppose the method `fun2` is defined as:

```
public int fun2(int x, int y)
{
  y -= x;
  return y;
}
```

What are the values of the variables `a` and `b` after the following code is executed?

```
int a = 3, b = 7;
b = fun2(a, b);
a = fun2(b, a);
```

(A) `a` is `-1` and `b` is `4`
(B) `a` is `-4` and `b` is `7`
(C) `a` is `-4` and `b` is `4`
(D) `a` is `3` and `b` is `7`
(E) `a` is `3` and `b` is `4`

5. Assuming that `a` and `b` are Boolean variables, when is the following expression true?

```
!(!a || b) || (!a && b)
```

(A) If and only if `a` and `b` have different values
(B) If and only if `a` and `b` have the same value
(C) If and only if both `a` and `b` are true
(D) If and only if both `a` and `b` are false
(E) Never

6. Suppose a, b, and c are positive integers under 1000 and x satisfies the formula

$$\frac{a}{b} = \frac{c}{x}$$

The integer value d is obtained by truncating x to an integer. Which of the following code segments correctly calculates d?

I.
```
d = c * b / a;
```

II.
```
int temp = c * b;
d = temp / a;
```

III.
```
int temp = b / a;
d = c * temp;
```

(A) I only
(B) II only
(C) I and II
(D) II and III
(E) I, II, and III

7. Given two classes, `Animal` and `Mammal`, which of the situations below would make the following statement valid?

```
Animal a = new Mammal("Elephant");
```

(A) `Mammal` extends `Animal`, and `Mammal` has a constructor with one parameter of the `String` type.
(B) `Mammal` extends `Animal`, and `Animal` has a constructor with one parameter of the `String` type.
(C) `Animal` has a method `Mammal` that takes one parameter of the `String` type.
(D) `Animal` has a public data field `String Mammal`.
(E) None of the above

8. What is the value of `v[4]` after the following code is executed?

```
int d = 1;
int[] v = {1, 1, 1, 1, 1};

for (int i = 0; i < v.length; i++)
{
  d *= 2;
  v[i] += d;
}
```

(A) 16
(B) 32
(C) 33
(D) 64
(E) 65

9. Which of the following is NOT a good reason to use comments in programs?

(A) To document the names of the programmers and the date of the last change
(B) To document requirements for correct operation of a method
(C) To document which methods of a class are private
(D) To describe parameters of a method
(E) To explain a convoluted piece of code

10. What is the result from the following code segment?

```
ArrayList<String> xyz = new ArrayList<String>();
xyz.add("X");
xyz.add("Y");
xyz.add("Z");

int count = 0;
for (String s1 : xyz)
{
  for (String s2 : xyz)
  {
    if (s1.equals(s2))
    {
      count++;
    }
  }
}

System.out.print(count);
```

(A) Syntax error
(B) 0 is displayed
(C) 1 is displayed
(D) 3 is displayed
(E) `NullPointerException`

11. Which of the following statements about Java's platform independence are true?

I. The number of bytes used by an `int` variable is the same on any computer.

II. Java source code is compiled into bytecodes, which may then be run on any computer that has a Java Virtual Machine installed.

III. Overflow in arithmetic operations occurs at the same values regardless of the platform on which the Java program is running.

(A) I only
(B) II only
(C) I and II
(D) II and III
(E) I, II, and III

12. Suppose a class `Particle` has the following variables defined:

```
public class Particle
{
  public static final int START_POS = 100;
  private double velocity;

  /* other code not shown */
}
```

Which of the following is true?

(A) `velocity` can be passed as an argument to one of `Particle`'s methods, but `START_POS` cannot.
(B) Java syntax rules wouldn't allow us to use the name `startPos` instead of `START_POS`.
(C) A statement `double pos = START_POS + velocity;` in one of `Particle`'s methods would result in a syntax error.
(D) Java syntax rules wouldn't allow us to make `velocity` public.
(E) A statement `START_POS += velocity;` in one of `Particle`'s methods would result in a syntax error.

13. What is the output of the following code segment?

```
String s = "ban";
ArrayList<String> words = new ArrayList<String>();
words.add(s);
words.add(s.substring(1));
words.add(s.substring(1,2));
String w = "";
for (int k = 0; k < words.size(); k++)
{
  w += words.get(k);
}
System.out.print(w.indexOf("an"));
```

(A) `1`
(B) `2`
(C) `3`
(D) `ana`
(E) `banana`

14. Consider the following code segment, intended to find the position of an integer `targetValue` in `int[] a`:

```
int i = 0, position;
while (a[i] != targetValue)
{
  i++;
}
position = i;
```

When will this code work as intended?

(A) Only when `0 <= targetValue < a.length`
(B) Only when `targetValue == a[0]`
(C) Only when `targetValue == a[i]` for some `i`, `0 <= i < a.length`
(D) Only when `targetValue != a[i]` for any `i`, `0 <= i < a.length`
(E) Always

15. Given two initialized `String` variables, `str1` and `str2`, which of the following conditions correctly tests whether the value of `str1` is greater than or equal to the value of `str2` (in lexicographical order)?

(A) `str1.compareTo(str2) == true`
(B) `str1.compareTo(str2) >= 0`
(C) `str1 >= str2`
(D) `str1.equals(str2) || str1.compareTo(str2) == 1`
(E) `str1.length() > str2.length() || str1 >= str2`

Questions 16-17 refer to the method `smile` below:

```
public static void smile(int n)
{
  if (n == 0)
    return;
  for (int k = 1; k <= n; k++)
    System.out.print("smile!");
  smile(n-1);
}
```

16. What is the output when `smile(4)` is called?

(A) `smile!`
(B) `smile!smile!`
(C) `smile!smile!smile!`
(D) `smile!smile!smile!smile!`
(E) `smile!smile!smile!smile!smile!smile!smile!smile!smile!smile!`

17. When `smile(4)` is called, how many times will `smile` actually be called, including the initial call?

(A) 2
(B) 3
(C) 4
(D) 5
(E) 10

18. Consider the following method from `ClassX`:

```
private int modXY(int x, int y)
{
  r = x / y;
  return x % y;
}
```

If `ClassX` compiles with no errors, which of the following statements must be true?

I. `modXY` has a side effect since `r` is not a local variable in `modXY`.
II. `r` must be an instance variable in the superclass of `ClassX`.
III. `r` must have the type `double`.

(A) I only
(B) II only
(C) III only
(D) I and II
(E) None

19. What is the output from the following code segment?

```
double pi = 3.14159;
int r = 100;
int area = (int)(pi * Math.pow(r, 2));
System.out.println(area);
```

(A) `30000`
(B) `31415`
(C) `31416`
(D) `314159`
(E) Depends on the particular computer system

20. Consider the following three code segments:

I.
```
int  i = 1;
while (i <= 10)
{
  System.out.print(i + " ");
  i += 2;
}
```

II.
```
for (int i = 0; i < 5; i++)
{
  System.out.print(2*i + 1
                          + " ");
}
```

III.
```
for (int i = 0; i < 10; i++)
{
  i++;
  System.out.print(i + " ");
}
```

Which of the three segments produce the same output?

(A) I and II only
(B) II and III only
(C) I and III only
(D) I, II, and III
(E) All three outputs are different.

21. Classes `Salsa` and `Swing` implement an interface `Dance`. If both of the calls

```
perform(new Salsa());
perform(new Swing());
```

are valid, which of the following could serve as definitions of the `perform` method(s)?

I. Two methods:

```
public void perform(Salsa dance) { /* code not shown */ }
public void perform(Swing dance) { /* code not shown */ }
```

II.
```
public void perform(Dance dance) { /* code not shown */ }
```

III.
```
public void perform(Object dance) { /* code not shown */ }
```

(A) I only
(B) II only
(C) I and II only
(D) II and III only
(E) I, II, and III

22. Consider the following class definitions:

```
public class Country
{
  public String toString() { return "Country"; }
}

public class Brazil extends Country
{
  public String toString() { return "Brazil"; }
}
```

What is the output of the following code segment?

```
Country country1 = new Country();
Country country2 = new Brazil();
System.out.print(country1 + " " + country2 + " ");
country1 = country2;
System.out.print(country1);
```

(A) `Country Country Country`
(B) `Country Country Brazil`
(C) `Country Brazil Country`
(D) `Country Brazil Brazil`
(E) `Brazil Brazil Brazil`

23. Consider the following method with two missing statements:

```
// returns the sum of all positive odd values
// among the first n elements of arr
// precondition:  1 <= n <= arr.length
public static int addPositiveOddValues(int[] arr, int n)
{
  int sum = 0;
  < statement1 >
  {
    < statement2 >
      sum += arr[i];
  }
  return sum;
}
```

Which of the following are appropriate replacements for < *statement1* > and < *statement2* > so that the method works as specified?

	< *statement1* >	< *statement2* >
(A)	`for (int i = 1; i < n; i += 2)`	`if (arr[i] > 0)`
(B)	`for (int i = 0; i < n; i++)`	`if (arr[i] > 0 && arr[i] % 2 != 0)`
(C)	`for (int i = 1; i <= n; i += 2)`	`if (arr[i] > 0)`
(D)	`for (int i = 0; i <= n; i++)`	`if (arr[i] % 2 != 0)`
(E)	None of the above	

Questions 24-29 refer to the code from the GridWorld case study.

24. How does a `Bug` act if there is a `Flower` directly in front of it in the grid?

(A) The `Bug` is removed from the grid.
(B) The `Bug` remains in its current state — no action is taken.
(C) The `Flower` is removed, and the `Bug` moves forward, putting into its old location a new `Flower`.
(D) The `Bug` turns 180 degrees.
(E) The `Bug` turns 45 degrees to the right.

25. Consider the following method:

```
// Returns true if location in front of bug contains a Rock;
// otherwise returns false.
public boolean rockInFront(Bug bug)
{
  Grid<Actor> gr = bug.getGrid();
  if (gr == null)
    return false;
  Location loc = bug.getLocation();
  Location next = < expression1 >;
  if (!gr.isValid(next))
    return false;
  return < expression2 >;
}
```

Which of the following could replace < *expression1* > and < *expression2* > for the method to work as specified?

(A) < *expression1* >: `loc.getAdjacentLocation(Location.AHEAD)`
< *expression2* >: `gr.get(next).equals(Rock)`

(B) < *expression1* >: `loc.getLocationToward(Location.AHEAD)`
< *expression2* >: `gr.get(next).equals(Rock)`

(C) < *expression1* >: `loc.getLocationToward(bug.getDirection())`
< *expression2* >: `gr.get(next).equals(Rock)`

(D) < *expression1* >: `loc.getAdjacentLocation(Location.AHEAD)`
< *expression2* >: `gr.get(next) instanceof Rock`

(E) < *expression1* >: `loc.getAdjacentLocation(bug.getDirection())`
< *expression2* >: `gr.get(next) instanceof Rock`

26. Suppose `grid` and `location` are instance fields in the `Actor` class. In `Actor`'s `putSelfInGrid` method —

```
public void putSelfInGrid(Grid<Actor> gr, Location loc)
{
  if (grid != null)
    throw new IllegalStateException(
            "This actor is already contained in a grid.");

  < missing code >

  grid = gr;
  location = loc;
}
```

which of the following could replace < *missing code* >?

I.
```
Actor actor = gr.get(loc);
if (actor != null)
  actor.removeSelfFromGrid();
gr.put(loc, this);
```

II.
```
gr.remove(loc);
gr.put(loc, this);
```

III.
```
gr.put(loc, this);
```

(A) I only
(B) II only
(C) I and II
(D) II and III
(E) I, II, and III

27. Suppose we replace the `act` method in `Actor` with an empty method (only braces and no code) and remove the `act` method from `Rock`. What effect will this have on the `BugRunner` program?

(A) The program will work as before with no changes.
(B) The program will work as before, except `Actor` objects, if added to the grid, won't flip over.
(C) The `Rock` class won't compile.
(D) It will become possible for a `Bug` to move to a location occupied by a `Rock`.
(E) `Critter`'s `processActors` method will no longer work as specified.

28. Let us change the design of the `Critter` class, moving the call to `getActors` from `act` to `processActors` —

```
public void act()
{
  if (getGrid() == null)
    return;
  processActors();
  ...
}

public void processActors()
{
  ArrayList<Actor> actors = getActors();
  for (Actor a : actors)
  ...
}
```

How does the new design compare to the original design?

(A) The new design violates encapsulation.

(B) The new design is less flexible than the original design, because some of the subclasses of `Critter` that overrode only `getActors` and `processActors` now will have to override `act`, too.

(C) The new design is less flexible than the original design, because in the original design a `Critter`'s subclass can override only the `getActors` method, while in the new design that is not possible.

(D) The new design is less flexible than the original design, because in the original design methods of `Critter`'s subclasses can call `super.getActors`, while in the new design that won't work.

(E) The new design is as flexible as the original design and may be more economical, because it may eliminate the need to override `getActors` in some subclasses of `Critter`.

29. Suppose a `RockChameleonCritter` "acts" exactly like a `ChameleonCritter`, except it changes color to the color of a randomly chosen adjacent `Rock`. If there are no rocks among its neighbors, `RockChameleonCritter`'s color remains unchanged. Which of the following approaches to implementing the `RockChameleonCritter` class can work?

I. Derive `RockChameleonCritter` from `Critter` and override the `processActors` and `makeMove` methods

II. Derive `RockChameleonCritter` from `ChameleonCritter` and override only the `processActors` method

III. Derive `RockChameleonCritter` from `ChameleonCritter` and override only the `getActors` method

(A) I only
(B) II only
(C) I and II
(D) II and III
(E) I, II, and III

Questions 30-32 involve reasoning about classes and objects used in an implementation of a library catalog system. An object of the class `BookInfo` represents information about a particular book, and an object of the class `LibraryBook` represents copies of a book on the library's shelves:

```
public class BookInfo
{
  private String title;
  private String author;
  private int numPages;

  // ... constructors not shown

  public String toString()
  {
    return title + " by " + author;
  }

  public String getTitle() { return title; }
  public int getNumPages() { return numPages; }
}

public class LibraryBook
{
  private BookInfo info;
  private int numCopies;   // Number of copies on shelf

  // ... constructors not shown

  public int getNumCopies() { return numCopies; }
  public void setNumCopies(int num)
                             { numCopies = num; }
  public BookInfo getInfo() { return info; }

  // if there are copies on shelf, decrements
  // the number of copies left and returns true;
  // otherwise returns false
  public boolean checkOut() { /* code not shown */ }
}
```

30. If `catalog` is declared in a client class as

```
LibraryBook[] catalog;
```

which of the following statements will correctly display *title* by *author* of the third book in `catalog`?

I.
```
System.out.println(catalog[2]);
```

II.
```
System.out.println(catalog[2].getInfo());
```

III.
```
System.out.println(catalog[2].getInfo().toString());
```

(A) I only
(B) II only
(C) I and II
(D) II and III
(E) I, II and III

31. Which of the following code segments will correctly complete the `checkOut()` method of the `LibraryBook` class?

I.
```
if (getNumCopies() == 0)
  return false;
else
{
  setNumCopies(getNumCopies() - 1);
  return true;
}
```

II.
```
int n = getNumCopies();
if (n == 0)
  return false;
else
{
  setNumCopies(n - 1);
  return true;
}
```

III.
```
if (numCopies == 0)
  return false;
else
{
  numCopies--;
  return true;
}
```

(A) I only
(B) II only
(C) I and II
(D) I and III
(E) I, II, and III

32. Consider the following method from another class, a client of `LibraryBook`:

```
// returns the total number of pages in
// all books in catalog that are on the shelves
public int totalPages(LibraryBook[] catalog)
{
  int count = 0;

  for (LibraryBook bk : catalog)
  {
    < statement >
  }
  return count;
}
```

Which of the following replacements for < *statement* > completes the method as specified?

(A) `count += bk.numCopies * bk.info.numPages;`
(B) `count += bk.getNumCopies() * bk.getNumPages();`
(C) `count += bk.(numCopies * info.getNumPages());`
(D) `count += bk.getNumCopies() * bk.getInfo().getNumPages();`
(E) None of the above

33. The following method is intended to remove from `ArrayList<Integer> list` all elements whose value is less than zero:

```
public void removeNegatives(ArrayList<Integer> list)
{
  int i = 0, n = list.size();

  while (i < n)
  {
    if (list.get(i) < 0)
    {
      list.remove(i);
      n--;
    }
    i++;
  }
}
```

For which lists of `Integer` values does this method work as intended?

(A) Only an empty list
(B) All lists that do not contain negative values in consecutive positions
(C) All lists where all the negative values occur before all the positive values
(D) All lists where all the positive values occur before all the negative values
(E) All lists

34. Consider the following interface and class:

```
public interface Student
{
  double getGPA();
  int getSemesterUnits();
}

public class FullTimeStudent
    implements Student, Comparable<Student>
{
  < required methods go here >
}
```

What is the minimum set of methods that a developer must implement in order to successfully compile the `FullTimeStudent` class?

(A) No methods would need to be implemented
(B) `getGPA(), getSemesterUnits()`
(C) `getGPA(), getSemesterUnits(), compareTo(Student s)`
(D) `getGPA(), getSemesterUnits(), equals(Student s), toString()`
(E) `getGPA(), getSemesterUnits(), compareTo(Student s), equals(Student s), toString()`

35. Which of the following best describes the return value for the method `propertyX` below?

```
// precondition: v.length >= 2
public boolean propertyX(int[] v)
{
  boolean flag = false;

  for (int i = 0; i < v.length - 1; i++)
  {
    flag = flag || (v[i] == v[i+1]);
  }

  return flag;
}
```

(A) Returns `true` if the elements of `v` are sorted in ascending order, `false` otherwise
(B) Returns `true` if the elements of `v` are sorted in descending order, `false` otherwise
(C) Returns `true` if `v` has two adjacent elements with the same value, `false` otherwise
(D) Returns `true` if `v` has two elements with the same value, `false` otherwise
(E) Returns `true` if all elements in `v` have different values, `false` otherwise

36. Consider the following method:

```
// returns the location of the target value
// in the array a, or -1 if not found
// precondition:  a[0] ... a[a.length - 1] are
//                sorted in ascending order
public static int search(int[] a, int target)
{
  int first = 0;
  int middle;
  int last = a.length - 1;

  while (first <= last)
  {
    middle = (first + last) / 2;
    if (target == a[middle])
      return middle;
    else if (target < a[middle])
      last = middle;
    else
      first = middle;
  }
  return -1;
}
```

This method fails to work as expected under certain conditions. If the array has five elements with values `3 4 35 42 51`, which of the following values of `target` would make this method fail?

(A) `3`
(B) `4`
(C) `35`
(D) `42`
(E) `51`

37. Brad has derived his class from the library class `JPanel`. `JPanel`'s `paintComponent` method displays a blank picture in a panel. Brad has redefined `JPanel`'s `paintComponent` to display his own picture. Brad's class compiles with no errors, but when he runs the program, only a blank background is displayed. Which of the following hypotheses CANNOT be true in this situation?

(A) Brad misspelled "paintComponent" in his method's name.
(B) Brad specified an incorrect return type for his `paintComponent` method.
(C) Brad chose the wrong type for a parameter in his `paintComponent` method.
(D) Brad specified two parameters for his `paintComponent` method, while `JPanel`'s `paintComponent` takes only one parameter.
(E) Brad has a logic error in his `paintComponent` code which prevents it from generating the picture.

Questions 38-40 refer to the following `SortX` class:

```
public class SortX
{
  public static void sort(String[] items)
  {
    int n = items.length;
    while (n > 1)
    {
      sortHelper(items, n - 1);
      n--;
    }
  }

  private static void sortHelper(String[] items, int last)
  {
    int m = last;
    for (int k = 0; k < last; k++)
    {
      if (items[k].compareTo(items[m]) > 0)
        m = k;
    }
    String temp = items[m];
    items[m] = items[last];
    items[last] = temp;
  }
}
```

38. The sorting algorithm implemented in the `sort` method can be best described as:

(A) Selection Sort
(B) Insertion Sort
(C) Quicksort
(D) Mergesort
(E) Incorrect implementation of a sorting algorithm

39. Suppose `names` is an array of `String` objects:

```
String[] names = {"Dan", "Alice", "Claire", "Evan", "Boris"};
```

If `SortX.sort(names)` is running, what is the order of the values in `names` after two complete iterations through the `while` loop in the `sort` method?

(A) `"Boris", "Alice", "Claire", "Dan", "Evan"`
(B) `"Alice", "Claire", "Boris", "Dan", "Evan"`
(C) `"Alice", "Boris", "Claire", "Evan", "Dan"`
(D) `"Alice", "Claire", "Dan", "Evan", "Boris"`
(E) None of the above

40. If `items` contains five values and `SortX.sort(items)` is called, how many times, total, will `items[k].compareTo(items[m])` be called in the `sortHelper` method?

(A) 5
(B) 10
(C) 15
(D) 25
(E) Depends on the values in `items`