Practice Test 4: Answers and Explanations

PRACTICE TEST 4 ANSWER KEY

1.	C	21.	C
2.	A	22.	C
3.	C	23.	D
4.	C	24.	D
5.	E	25.	A
6.	B	26.	E
7.	A	27.	B
8.	B	28.	E
9.	C	29.	B
10.	E	30.	A
11.	A	31.	B
12.	B	32.	B
13.	D	33.	E
14.	C	34.	C
15.	B	35.	E
16.	E	36.	E
17.	A	37.	E
18.	D	38.	A
19.	C	39.	E
20.	C	40.	C

ANSWER EXPLANATIONS

Section I: Multiple-Choice Questions

1. **C** This question tests how well you understand assigning values to variables and following the steps of the code. When `trial()` is called, `a` is assigned to the integer value of 10 and `b` is assigned to the integer value of 5. Next `doubleValues()` is called with `a` and `b` as inputs `c` and `d`. The method `doubleValues()` multiplies the input values `c` and `d` by 2 and prints them out. Thus, the value of `c` is 10 * 2 = 20 and the value of `d` is 5 * 2 = 10. Because `System.out.print()` does not print any line breaks or spaces, the resulting print is "2010." While `c` and `d` have been reassigned new values, these values exist only within the `doubleValues()` call, so the values of `a` and `b` are unchanged. After the `doubleValues()` method is completed, the `trial()` method then prints the values of `b` and then `a`, printing `510`. Once again, no spaces or line breaks are printed, so, combined together, what is printed from calling `trial()` is `2010510`. Therefore, the correct answer is (C).

2. **A** The method `mystery(int a, int b)` takes as input integers `a` and `b` with the precondition that `a > b > 0`. Plug in `x` = 5 and `y` = 2. These values are passed as parameters to make `a` = 5 and `b` = 2. Set `d = 0`. Now go to the for loop, initializing `c = a` = 5. Since it is still the case that `c > b`, execute the for loop. Execute `d = d + c` by setting `d` = 0 + 5 = 5. Decrease `c` by 1 to get `c` = 4. Since `c > b`, execute the for loop again. Execute `d = d + c` by setting `d` = 5 + 4 = 9. Decrease `c` by 1 to get `c` = 3. Since it is still the case that `c > b`, execute the for loop again. Execute `d = d + c` by setting `d` = 9 + 3 = 12. Decrease `c` by 1 to get `c` = 2. Since it is no longer the case that `c > b`, stop executing the for loop. Return `d` = 12. Now go through the choices and eliminate any that don't describe a return of 12. Choice (A) is the sum of all integers greater than `y` but less than or equal to `x`. The sum of all integers greater than 2 but less than or equal to 5 is 3 + 4 + 5 = 12, so keep (A). Choice (B) is the sum of all integers greater than or equal to `y` but less than or equal to `x`. The sum of all integers greater than or equal to 2 but less than or equal to 5 is 2 + 3 + 4 + 5 = 14, so eliminate (B). Choice (C) is the sum of all integers greater than `y` but less than `x`. The sum of all integers greater than 2 but less than 5 is 3 + 4 = 7, so eliminate (C). Choice (D) is the sum of all integers greater than or equal to `y` but less than `x`. The sum of all integers greater than or equal to 2 but less than 5 is 2 + 3 + 4 = 9, so eliminate (D). Choice (E) is the sum of all integers less than `y` but greater than or equal to `x`. However, there are infinitely many integers less than 2 and infinitely many integers greater than or equal to 5, so this return would require an infinite loop. Eliminate (E). Only one answer remains. The correct answer is (A).

3. **C** The question asks for what is printed by the call `mystery(3)`. In the call, 3 is taken as a parameter and assigned to `n`. The integer `k` is then initialized in the for loop and set equal to 0. Since `k < n`, execute the for loop. Call `mystery(0)`, while keeping your place in the call of `mystery(3)`. Again, `k` is initialized in the for loop and set equal to 0. In this call, since `n` = 0, it is not the case that `k < n`, so do not execute the for loop. This completes the call of `mystery(0)`, so return to `mystery(3)`. The next command is `System.out.print(k)`. Since `k` = 0, the first character printed

is `0`. Look to see whether any choices can be eliminated. However, all choices begin with `0`, so no choices can be eliminated. Continue with the method call. Since this is the last command of the for loop, increment `k` to get `k` = 1. Since it is still the case that `k` < `n`, execute the for loop again. Call `mystery(1)`, again keeping your place in the call of `mystery(3)`. Again, `k` is initialized in the for loop and set equal to 0. In this call, since `n` = 1, it is the case that `k` < `n`, so execute the for loop. Call `mystery(k)`, which is `mystery(0)`. As seen above, `mystery(0)` prints nothing, so execute the next line in `mystery(1)`, which is `System.out.print(k)`. Since `k` = 0, print `0`. Eliminate (A), since the second character printed is not `0`. Increment `k` to get `k` = 1. Since it is no longer the case that `k < n`, do not execute the for loop again and end this call of `mystery(1)`. Return to the original call of `mystery(3)`, where `k` = 1 and `System.out.print(k)` is the next statement. Print `1`. All remaining choices have `1` as the third character, so do not eliminate any choices. This is the end of the for loop, so increment `k` to get `k` = 2. Since `n` = 3, it is still the case that `k` < `n`, so execute the for loop. Call `mystery(2)`, while holding your place in `mystery(3)`. Again, `k` is initialized in the for loop and set equal to 0. In this call, since `n` = 2, it is the case that `k` < `n`, so execute the for loop. Call `mystery(0)`, which prints nothing. Execute `System.out.print(k)` to print `0`. Eliminate (B), since the next character is not `0`. Increment `k` to get `k` = 1. Since `k` < `n`, execute the for loop. Call `mystery(1)`, which, as described above, prints `0`. All remaining choices have `0` as the next character, so don't eliminate any choices. The next command in `mystery(2)` is `System.out.print(k)`, so print `1`. Again, keep all the remaining choices. In `mystery(2)`, increment `k` to get `k` = 2. Since `n` = 2, it is no longer the case that `k` < `n`, so stop executing the for loop. End `mystery(2)` and return to `mystery(3)`, where `k` = 2 and the next command is `System.out.print(k)`, so print `2`. Again, keep all remaining choices. Increment `k` to get `k` = 3. Since it is no longer the case that `k` < `n`, stop executing the for loop. The method call is complete for `mystery(3)`, so there will be no more printed characters. Eliminate (D) and (E), which have further characters printed.

In other words, the calling sequence looks like this

```
Function Call                                          Prints
mystery(3)
  mystery(0)
  System.out.print(0)                                  0
  mystery(1)
     mystery(0)
     System.out.print(0)                               0
  System.out.print(1)                                  1
  mystery(2)
     mystery(0)
     System.out.print(0)                               0
     mystery(1)
          mystery(0)
          System.out.print(0)                          0
     System.out.print(1)                               1
  System.out.print(2)                                  2
```

The correct answer is (C).

4. **C** `SelectionSort` walks through the array to find the smallest element in the part of the array not yet sorted. It then swaps that smallest element with the first unsorted element.

Start with the original array of values.

```
4    10    1    2    6    7    3    5
```

The smallest element is 1. Swap that with the first unsorted element, 4. The array now looks like this.

```
1    10    4    2    6    7    3    5
```

This is not a choice, so continue. The smallest element in the unsorted part of the array (from 10 to 5) is 2. Swap that with the first unsorted element, 10.

```
1    2    4    10    6    7    3    5
```

This is still not a choice, so continue this process. The smallest element in the unsorted part of the array (from 4 to 5) is 3. Swap that with the first unsorted element, 4.

```
1    2    3    10    6    7    4    5
```

This is (C), so stop here. If you're interested, here are the complete moves for selection sort.

```
1    2    3    4    6    7    10    5

1    2    3    4    5    7    10    6

1    2    3    4    5    6    10    7

1    2    3    4    5    6    7    10
```

The correct answer is (C).

5. **E** Execute the commands. The integer `k` and the integer array `A` are initialized. The array `A` is given length 7. Thus, the array, `A`, has seven elements with indexes from 0 to 6. Execute the first for loop. Set `k` = 0. Since `k <  A.length`, execute the for loop, which sets `A[0]` equal to `A.length – k` = 7 – 0 = 7. Increment `k` to get `k` = 1. Since 1 < 7, execute the for loop, which sets `A[1]` equal to `A.length – k` = 7 – 1 = 6. Increment `k` to get `k` = 2. Since 2 < 7, execute the for loop, which sets `A[2]` equal to `A.length – k` = 7 – 2 = 5. Increment `k` to get `k` = 3. Since 3 < 7, execute the for loop, which sets `A[3]` equal to `A.length – k` = 7 – 3 = 4. Increment `k` to get `k` = 4. Since 4 < 7, execute the for loop, which sets `A[4]` equal to `A.length – k` = 7 – 4 = 3. Increment `k` to get `k` = 5. Since 5 < 7, execute the for loop, which sets `A[5]` equal to `A.length – k` = 7 – 5 = 2. Increment `k` to get `k` = 6. Since 6 < 7, execute the for loop, which sets `A[6]` equal to `A.length – k` = 7 – 6 = 1. Increment `k` to get `k` = 7. Since it is not the case that 7 < 7, stop executing the for loop. Therefore, the array has the values

```
7    6    5    4    3    2    1
```

The second for loop puts the value of the `k`th element of the array into the `(k + 1)`st element. Be careful!

A quick glance might lead you to believe that each value is shifted in the array one spot to the right, giving an answer of (D). However, a step-by-step analysis demonstrates that this will not be the case. The for loop sets `k` = 0. Since 0 < `A.length` – 1, execute the for loop. The command in the for loop is `A[k+1] = A[k]`. Since `k` = 0, set `A[1]` = `A[0]` = 7. Increment `k` to get `k` = 1. Since it is still the case that `k` < `A.length` – 1, execute the for loop. Since `k` = 1, set `A[2]` = `A[1]` = 7. Continue in this manner. Since `A.length` – 1 = 7 – 1 = 6, only execute the for loop through `k` = 5. Each execution of the for loop is shown below.

`k`	Assignment	`A[]`
0	`A[1] = A[0]`	`7 7 5 4 3 2 1`
1	`A[2] = A[1]`	`7 7 7 4 3 2 1`
2	`A[3] = A[2]`	`7 7 7 7 3 2 1`
3	`A[4] = A[3]`	`7 7 7 7 7 2 1`
4	`A[5] = A[4]`	`7 7 7 7 7 7 1`
5	`A[6] = A[5]`	`7 7 7 7 7 7 7`

The final values contained by `A` are `7 7 7 7 7 7 7`. The correct answer is (E).

6. **B** Choices (C), (D), and (E) are syntactically invalid according to the given class definitions. In (C), `p` is used as a `PostOffice` object rather than an array of `PostOffice` objects. Choice (D) treats `getMail` and `getBox` as static methods without invoking them from an object. Choice (E) creates a new `Mail` object attempting to use a `Mail` constructor. Even if such a constructor were available, there would be no way for the constructor to know about `p`, the array of PostOffices.

   Choices (A) and (B) differ only in the indexes of the array and methods. There are two clues in the question that indicate that (B) is the correct answer. First, `p` is declared as an array of 10 `Post-Offices`. This means that `p[10]` would raise an `ArrayListOutOfBoundsException`. Remember that `p[9]` actually refers to the 10th post office, since array indexes begin with 0. Second, the comments in the class definitions for the `getBox` and `getMail` methods indicate that the parameter they take is zero-based. Therefore, they should be passed an integer one less than the number of the box or piece of mail needed. For either reason, eliminate (A). The correct answer is (B).

7. **A** In the method `printEmptyBoxes`, the loop variable `k` refers to the index of the post office and the loop variable `x` refers to the index of the box within the post office. Choice (A) is correct. It checks to see if the box is assigned and if it does not have mail using the appropriate methods of the Box class. It then prints out the box number of the box. Choice (B) is similar to (A) but incorrectly interchanges `x` and `k`. Eliminate (B). Choice (C) omits the call to the method `getBox`. Therefore, it attempts to call the `getBoxNumber()` method of the `PostOffice` object `p[k]`. Since `PostOffice` objects do not have a `getBoxNumber()` method, this would result in a compile error. Eliminate (C). Choice (D) is similar to (C) but interchanges `x` and `k`. Eliminate (D). Choice (E) prints the

value of `k`, which is the index of the post office rather than the index of the box. Eliminate (E). The correct answer is (A).

8. **B** There are four possibilities for the values of `a` and `b`. Either both `a` and `b` are true, both `a` and `b` are false, `a` is true and `b` is false, or `a` is false and `b` is true.

Analyze the possibilities in a table.

`a` (initial value)	`b` (initial value)	`a = a && b` (final value)	`b = a || b` (final value)
`true`	`true`	`true`	`true`
`true`	`false`	`false`	`false`
`false`	`true`	`false`	`true`
`false`	`false`	`false`	`false`

Remember when calculating `b = a || b` that a has already been modified. Therefore, use the final, rather than the initial, value of `a` when calculating the final value of `b`.

Go through each statement. Statement I says that the final value of a is equal to the initial value of `a`. This is not the case in the second row of the table, so Statement I is not always true. Eliminate any choice that includes it: (A) and (D). Statement II says that the final value of `b` is equal to the initial value of `b`. This is true in each row and Statement II is always true, so eliminate the choice that does not include it: (C). Statement III says that the final value of `a` is equal to the initial value of `b`. This is not the case in the third row of the table, so Statement III is not always true. Eliminate any choice that includes it: (E). The correct answer is (B).

9. **C** The best way to solve this problem is to look at each if statement individually. The integer `x` is initialized and then set equal to 53. The next statement is an if statement with the condition `(x > 10)`. Since 53 > 10, execute the statement, `System.out.print("A")`, and print `A`. The next statement is an if-else statement, this one having the condition `(x > 30)`. Since 53 > 30, execute the if statement, `System.out.print("B")`, and print `B`. Even though 53 > 40, because the statement is an else statement, it cannot be executed if the original if statement was executed. Therefore, do not execute the else statement, and do not print `C`. The next statement is another if statement, this one having the condition `(x > 50)`. Since 53 > 50, execute the statement, `System.out.print("D")`, and print `D`. The next statement is another if statement, this one having the condition `(x > 70)`. Since it is not the case that 53 > 70, do not execute the statement. There is no further code to execute, so the result of the printing is `ABD`. The correct answer is (C).

10. **E** This problem tests your ability to work with nested for loops. Although it appears complicated at first glance, it can easily be solved by systematically walking through the code.

Be sure to write the values of the variables `j` and `k` down on paper; don't try to keep track of `j` and `k` in your head. Use the empty space in the question booklet for this purpose.

Code	j	k	Output
Set j to the value -2.	–2		
Test the condition: j <= 2? Yes.	–2		
Set k to the value that j has.	–2	–2	
Test the condition: k < j + 3? Yes.	–2	–2	
Output k.	–2	–2	-2
Update k: k++	–2	–1	-2
Test the condition: k < j + 3? Yes.	–2	–1	-2
Output k.	–2	–1	-2 -1
Update k: k++	–2	0	-2 -1
Test the condition: k < j + 3? Yes.	–2	0	-2 -1
Output k.	–2	0	-2 -1 0
Update k: k++	–2	1	-2 -1 0
Test the condition: k < j + 3? No.	–2	1	-2 -1 0
Update j: j = j + 2	0	1	-2 -1 0
Test the condition: j <= 2? Yes	0	1	-2 -1 0
Set k to the value that j has.	0	0	-2 -1 0
Test the condition: k < j + 3? Yes.	0	0	-2 -1 0
Output k.	0	0	-2 -1 0 0

At this point, stop because (E) is the only choice that starts -2 -1 0 0. However, repeating this process will result in the correct output of -2 -1 0 0 1 2 2 3 4. The correct answer is (E).

11. **A** To solve this problem, first note that count % 3 is equal to the remainder when count is divided by 3. Execute the method call mystery(5, "X"), taking count = 5 and s = "X" as parameters. Because it is not the case that 5 <= 0, do not execute the first if statement.

Because 5 % 3 = 2, calling mystery(5, "X") will execute the else statement, printing X. Then, it will call mystery(4, "X"), and return. Because 4 % 3 = 1, calling mystery(4, "X") will execute the else if statement, printing X–X, call mystery(3, "X"), and return. At this point, (C) and (E) can be eliminated. Because 3 % 3 = 0, calling mystery(3, "X") will execute the if statement, printing X–X, call mystery(2, "X"), and return. Note that you can stop at this point because (B) and (D) can be eliminated. If you're interested, here is the remainder of the execution. Because 2 % 3 = 2, calling mystery(2, "X") will execute the else statement, printing X, call mystery(1, "X"), and return. Because 1 % 3 = 1, calling mystery(1, "X") will execute the else if statement, printing X-X, call mystery(0, "X"), and return. Finally, calling mystery(0, "X") will simply return because count is less than or equal to zero. Putting it all together, mystery(5, "X") prints

```
XX-XX--XXX-X
```

The correct answer is (A).

12. **B** The only information that you need to solve this problem is the first sentence in the description of the constructor. Class `DiningRoomSet` has a constructor, which is passed a `Table` object and an `ArrayList` of `Chair` objects. Because you are writing a constructor, you can immediately eliminate (A) and (C), which are void methods. Constructors never return anything, so there is never a need to specify a void return. Choice (E) is incorrect because the constructor is passed a `Table` object and a `Chair` object, not a `Table` object and an `ArrayList` of `Chair` objects. Choice (D) is incorrect because the second parameter has two types associated with it. It should have only one type: `ArrayList`. Choice (B) is correct.

13. **D** The best way to solve this problem is to eliminate choices. Choices (A) and (B) are incorrect because the class description states that the `getPrice` method of the `DiningRoomSet` class does not take any parameters. Choice (C) can be eliminated because the private data field `myChairs` is not a `Chair`; it is an `ArrayList`. Therefore, it does not have a `getPrice` method. This leaves (D) and (E). You need to know that `ArrayLists` are accessed using the get method while arrays are accessed using the `[]` notation. `MyChairs` is an `ArrayList`, so the correct answer is (D).

14. **C** To solve this type of problem, use information about the output and the loops to eliminate incorrect choices. Then, if necessary, work through the code for any remaining choices to determine the correct answer. There are six lines of output. By examining the outer loop of each of the choices, you can eliminate (B) because it will traverse the loop seven times, and during each traversal at least one number will be printed. You can also eliminate (A) because the outer loop will never be executed. The initial value of j is 6, but the condition `j < 0` causes the loop to terminate immediately. The remaining choices have the same outer loop, so turn your attention to the inner loop. Eliminate (D) because the first time through the loop, a 7 will be printed—clearly not the correct output. Finally, eliminate (E) because the condition of the inner loop, `k >= 0,` will cause a 0 to be printed at the end of each line. This leaves (C), which is indeed the correct answer.

15. **B** This problem tests your knowledge of the methods of the ArrayList class. The following table shows the contents of list after each line of code.

Code	Contents of list	Explanation
`list = new ArrayList();`	`[]`	A newly created ArrayList is empty
`list.add(new Integer(7));`	`[7]`	Adds 7 to the end of list
`list.add(new Integer(6));`	`[7, 6]`	Adds 6 to the end of list
`list.add(1, new Integer(5));`	`[7, 5, 6]`	Inserts 5 into list as position 1, shifting elements to the right as necessary
`list.add(1, new Integer(4));`	`[7, 4, 5, 6]`	Inserts 4 into list as position 1, shifting elements to the right as necessary
`list.add(new Integer(3));`	`[7, 4, 5, 6, 3]`	Adds 3 to the end of list

`list.set(2, new Integer(2));`	`[7, 4, 2, 6, 3]`	Replaces the number at position 2 in the list with 2
`list.add(1, new Integer(1));`	`[7, 1, 4, 2, 6, 3]`	Inserts 1 into list at position 1, shifting elements to the right as necessary

The next command is to print list. Therefore, the correct answer is (B).

16. **E** Although the `Dog` class extends `Animal`, Java does not require `Dog` to have any specific methods. New methods will be unique to `Dog` objects (and those of its subclasses, when applicable) and methods inherited from `Animal` can be invoked as normal.

The correct answer is (E).

17. **A** This question is testing your understanding of static and dynamic types.

```
Fish Bob = new Shark();
```

This line creates the `Bob` variable with the static type of `Fish` and the dynamic type of `Shark`.

```
System.out.println(Bob.endoskeleton);
```

This line prints the `endoskeleton` field of the variable `Bob`. When looking up a field, the static type is used so "`bone`" is printed.

```
Bob.action();
```

This line executes the method `action()`. When looking up a method, the dynamic type is used so "`chomp chomp`" is printed. The correct answer is (A).

18. **D** The insertion sort algorithm creates a sorted array by sorting elements one at a time.

The for loop of the code shows that the elements are being sorted from left to right. To determine the position of the new element to be sorted, the value of the new element must be compared with the values of the sorted elements from right to left. This requires `index--` in the while loop, not `index++`. Choices (A) and (C) are wrong.

In order to place the new element to be sorted in its correct position, any sorted elements larger than the new element must be shifted to the right. This requires `sort[index] = sort[index -1]`. Choices (B) and (E) are wrong. The correct answer is (D).

19. **C** In order for the array to be sorted in descending order, you will need to make a change. Plug each answer choice into the array to see which is correct. In (A) and (B), as the index will never be less than 0, the contents of the while loop will never be executed. Choice (C) is the correct answer as it changes the condition of finding the position of the new element from being less than the compared element to greater. In (D), altering the for loop this way would lead to the elements being sorted from right to left but still in ascending order. In (E), the index is initialized at 1 and decremented while index > 0, so it will execute the for loop only once. The correct answer is (C).

20. **C** The rate at which insertion sort runs depends on the number of comparisons that are made. The number of comparisons is minimized with an array with elements that are already sorted in ascending order and maximized with an array with elements that are sorted in descending order. The array in (C) is sorted in reverse order and will require the most comparisons to sort. The correct answer is (C).

21. **C** All three responses look very similar, so look carefully at the difference. Statement I is missing the keyword `new`, which is needed to create a new object. Eliminate any choice that includes Statement I: (A) and (D). Statements II and III differ in that Statement II uses `f.numerator` and `f.denominator` while Statement III uses `f.getNumerator()` and `f.getDenominator()`. Since the integers `numerator` and `denominator` are private while the methods `getNumerator()` and `getDenominator()` are public, the methods must be called by another object. Therefore, Statement II is not valid. Eliminate any choice that includes it: (B) and (E). The correct answer is (C).

22. **C** Go through the choices one at a time. Choice (A) is incorrect because the multiplication operator, `*`, is not defined to work on `Fraction` objects. Eliminate (A). Choice (B) is incorrect because the multiply method takes only one parameter and it is not correctly invoked by a `Fraction` object. Eliminate (B). Choice (C) correctly calls the `multiply()` method as through a fraction object, taking the other fraction as a parameter. Keep (C). Choice (D) attempts to create a new `Fraction` object but incorrectly constructs it by passing two `Fraction` objects rather than two integers. Eliminate (D). Finally, while (E) calculates the value of the result of the multiplication, the "`/`" operator assigns integer types, which cannot be applied to `answer`, which is an object of type `Fraction`. Eliminate (E). The correct answer is (C).

23. **D** Go through each statement one at a time. Constructor I is legal. This default constructor of the `ReducedFraction` class will automatically call the default constructor of the `Fraction` class and the private data of both classes will be set appropriately. Eliminate any choice that does not include Constructor I: (B), (C), and (E). Because Constructor II is not included in the remaining choices, do not worry about analyzing it. Constructor III is also legal. The call `super(n, d)` invokes the second constructor of the `Fraction` class which sets the private data of the `Fraction` class appropriately. The private data of the `ReducedFraction` class is then set explicitly in the `ReducedFraction` constructor. Note that if the call `super(n, d)` were not present, Constructor III would still be legal. However, it would create a logical error in the code as the default constructor of the Fraction class would be invoked and the private data of the Fraction class would not be set appropriately. Eliminate (A), which does not include Constructor III. Only one choice remains, so there is no need to continue. However, to see why Constructor II is illegal, remember that derived classes may not access the private data of their super classes. In other words, the constructor for a `ReducedFraction` may not directly access numerator and denominator in the Fraction class. The correct answer is (D).

24. **D** Go through each statement one at a time. Statement I is incorrect. "==" checks object references as opposed to their contents. This is an important distinction as two different objects may hold the same data. Eliminate (A), (C), and (E), which contain Statement I. Both of the remaining choices contain Statement II, so don't worry about analyzing it. Statement III is correct. The `compareTo()` method of the `String` class returns 0 if the two `String` objects hold the same strings. Eliminate (B), which does not include Statement III. Only one choice remains, so there is no need to continue. However, you should see why Statement II is also correct. The `equals` method of the `String` class returns true if the two `String` objects hold the same strings. The correct answer is (D).

25. **A** The values of `s` and `t` are not changed in `mystery()`. Even though `s` and `t` are both parameters of the method, only the instance variables `a` and `b` are changed. Thus, the original Strings `s` and `t` are unaffected by any action in `mystery()`. The correct answer is (A).

26. **E** Though these two classes are related through an inheritance relationship, there is no rule in Java that requires this structure to share methods and/or data. Therefore, there are no requirements on either class. The correct answer is (E).

27. **B** An example will help clarify this question.

Consider the following for loop:

```
for (int k = 0; k < 3; k++)
{
      Systemout.println(k);
}
```

This prints out the integers 0 to 2, one number per line. Matching `int k = 0` to <1>; `k < 3` to <2>; `k++` to <3> and `System.out.println(k);` to <4>, go through each choice one at a time and determine whether each has the same functionality.

Choice (A) initializes `k` and assigns it the value 0. Then it tests to determine whether `k` < 3. This is true, so execute the while loop. The next command is `k++`, so increase `k` by 1 to get `k` = 0 + 1 = 1. Now execute `System.out.println(k)` to print `1`. However, the first number printed in the original was `0`, so this is incorrect. Eliminate (A).

Choice (B) initializes `k` and assigns it the value 0. Then it tests to determine whether `k` < 3. This is true, so execute the while loop. Now execute `System.out.println(k)` to print `0`. The next command is `k++`, so increase `k` by 1 to get `k` = 0 + 1 = 1. Go back to the top of the while loop. Since `k` = 1, it is still the case that `k` < 3, so execute the while loop. Now execute `System.out.println(k)` to print `1`. The next command is `k++`, so increase `k` by 1 to get `k` = 1 + 1 = 2. Go back to the top of the while loop. Since `k` = 2, it is still the case that `k` < 3, so execute the while loop. Now execute `System.out.println(k)` to print `2`. The next command is `k++`, so increase `k` by 1 to get `k` = 2 + 1 = 3. Go back to the top of the while loop. Since `k` = 3, it is no longer the case that `k` < 3, so stop executing the while loop. The result of the program is printing the integers 0 to 2, one number per line, so keep (B).

Choice (C) initializes `k` and assigns it the value 0. Then it tests the condition `!(k < 3)`. Since it is the case that `k` < 3, the statement `(k < 3)` has the boolean value true making the boolean value of `!(k < 3)` false. Thus the while loop is not executed. Since the while loop is not executed, nothing is printed. Eliminate (C).

Choice (D) has the same initialization statement and while loop condition as choice (C), so nothing is printed by this choice either. Eliminate (D).

Choice (E) initializes `k` and assigns it the value 0. The next command is `k++`, so increase `k` by 1 to get `k` = 0 + 1 = 1. Then it tests to determine whether `k` < 3. This is true, so execute the while loop. Execute `System.out.println(k)` to print `1`. However, the first number printed in the original was 0, so this is incorrect. Eliminate (E).

The correct answer is (B).

28. **E** This question tests your knowledge of operator precedence. In Java, multiplication, division, and modulus are performed before addition and subtraction. If more than one operator in an expression has the same precedence, the operations are performed left-to-right. Parenthesizing the expression one step at a time

```
      a / b + c − d % e * f
→     (a / b) + c − d % e * f
→     (a / b) + c − (d % e) * f
→     (a / b) + c − ((d % e) * f)
→     ((a / b) + c) − ((d % e) * f)
```

The correct answer is (E).

29. **B** Come up with a sample array of 10 strings with length 5. Let `String[] x` be `{“gator”, “teeth”, “ducky”, “quack”, “doggy”, “woofs”, “kitty”, “meows”, “bears”, “growl”}`. The code must print the first letter of all 10 strings, followed by the second letter of all 10 strings, and so on. Therefore, it must print

```
gtdqdwkmbgaeuuooieer…
```

Go through each segment one at a time.

Segment I initializes `i` and `j` with no values. The outer for loop sets `i = 0`, and the inner for loop sets `j = 0`. The instruction of the inner for loop is `System.out.print(x[i].substring(j, j + 1)`. The `substring(a, b)` method of the `String` class returns a string made up of all the characters starting with the index of the `a` through the index of `b − 1`. Therefore, `x[i].substring(j, j + 1)` returns the characters of `x[i]` from index `j` through index `(j + 1) − 1`. Since `(j + 1) − 1 = j`, it returns all the characters from indexes `j` through `j`—a single character string made up of the character at index `j`. Therefore, if i = 0 and `j` = 0, `System.out.print(x[i].substring(j, j + 1)` returns the character at index 0 of the string at index 0. This is the "g" from "gator". This is what should be printed, so continue. The inner for loop increments `j` to get `j` = 1. Since `j` < 5, the loop is executed again, printing the character at index 1 of the string at index 0. This is the character "a" from "gator". However, the character "t" from "teeth" should be the next character printed, so Segment I does not execute as intended. There is no need to continue with Segment I, but note that it will eventually print all of the characters of the first string followed by all of the characters of the second string, and so on. Eliminate (A), (C), and (E), which include Segment I.

Both remaining choices include Segment II, so don't worry about checking this one. Instead, analyze Segment III.

Segment III initializes `i` and `j` with no values. The outer for loop sets `i` = 0, and the inner for loop sets `j` = 0. The instruction of the inner for loop is `System.out.print(x[i].substring(j, j + 1)`. As discussed above, when `i = 0` and `j = 0`, this command returns the character at index 0 of the string at index 0, which is the "g" from "gator". This is what should be printed, so continue. The inner for loop increments `j` to get `j` = 1. Since `j` < 5, the loop is executed again, printing the character at index 1 of the string at index 0. This is the character "a" from "gator". Again, the character "t" from "teeth" should be the next character printed, so Segment III does not execute as intended. There is no need to continue with Segment III, but note that it will attempt to print the first 10 characters of the first 5 strings. Since each String contains only 5 characters, an `IndexOutOfBoundsException` will be thrown when `j` = 5 and the program attempts to print the character at index 5. Eliminate (D), which includes Segment III.

Only one answer remains, so there is no need to continue. However, to see why Segment II is correct, note that it is similar to Segment I, but it reverses the roles of `i` and `j`. The command `System.out.print(x[j].substring(i, i + 1))` prints the character at index `j` of the string at index `i`. The inner for loop increments the index of the `String` array before the outer loop increments the index of the character in each `String`. Therefore, Segment II correctly prints out the first character of all 10 strings followed by the second character of all 10 strings, and so on. The correct answer is (B).

30. **A** Begin with (D) and (E), which discuss whether an error would occur. Because `c` is initialized with the command in the third line, `double c;`, eliminate (D). It is perfectly legal to assign a value of type int to a variable of type double in an expression, so eliminate (E). However, certain rules apply when evaluating the expression. In the expression `c = a / b; a` and `b` are both integers. Therefore, the `/` operator represents integer division. The result of the division is truncated by discarding the fractional component. In this example, 7 / 4 has the value 1—the result of truncating 1.75. When assigning an integer to a variable of type double, the integer value is converted to its equivalent double value. Therefore, the correct answer is (A).

31. **B** For the code to print the word `Yes` two conditions must be true. The remainder must be zero when `x` is divided by 2. In other words, `x` must be divisible by 2, that is, even. The value after truncating the result when `x` is divided by 3 must be 1. In other words, 1 ≤ `x` / 3 < 2. The second condition is

more narrow than the first. The only integers that fulfill the second condition are 3, 4, and 5. Of those, only 4 is even and therefore also fulfills the first condition. Therefore, the correct answer is (B).

32. **B** Go through each statement one at a time. Segment I is incorrect because all parameters in Java are passed by value. After a method returns to its caller, the value of the caller's parameters are not modified. The strings will not be swapped. Eliminate (A), (C), and (E), which include Segment I. Since both remaining choices include Segment III, don't worry about it. Segment II is incorrect because `myName` is a private data field of the `SomeClass` class and may not be accessed by the `swap` method. Note that the question specifically states that the `swap` method is not a method of the `SomeClass` class. Eliminate (D), which includes Segment II. Only one choice remains, so there is no need to continue. To see that Segment III correctly swaps the names of the objects, note that it calls the public methods `setName()` and `getName()` rather than the private `String myName`. Because the references of the instance object variables are the same as the references of the class object variables, modifying the data of the instance variables also changes the data of the class variable. The correct answer is (B).

33. **E** The way to approach this type of design problem is to look for HAS-A and IS-A relationships among the distinct pieces of data. A book HAS-A title and author. The title and author should be data fields of the `Book` class, either as `Strings` or as their own unrelated classes. This information is not enough to answer the question though. Looking for the IS-A relationships, a mystery IS-A work of fiction which IS-A book. Therefore, it makes good design sense for these three items to be separate classes. Specifically, `Mystery` should be a subclass of `FictionWork`, which should be a subclass of `Book`. Similarly, `RomanceNovel` and `ScienceFiction` should be subclasses of `FictionWork` and `Biography`, `Cookbook`, and `SelfHelpBook` should be subclasses of `NonFictionWork`, which should be a subclass of `Book`. Only (E) meets all of these design criteria.

34. **C** In order to solve this recursive problem, work backward from the base case to the known value of `mystery(4)`.

Let y represent the <missing value>. Note that `mystery(1) = y`. Now calculate `mystery(2)`, `mystery(3)`, and `mystery(4)` in terms of y.

```
mystery(2) = 2 * mystery(1) + 2
           = 2 * y + 2
mystery(3) = 2 * mystery(2) + 3
          = 2 * (2 * y + 2) + 3
          = 4 * y + 4 + 3
          = 4 * y + 7
mystery(4) = 2 * mystery(3) + 4
          = 2 * (4 * y + 7) + 4
          = 8 * y + 14 + 4
          = 8 * y + 18
```

Because `mystery(4)` also equals 34, set 8 * y + 18 = 34 and solve for y.

8 * y + 18 = 34
8 * y = 16
y = 2

The correct answer is (C).

35. **E** The for loop terminates when the condition is no longer true. This can happen either because k is no longer less than `X.length` or because `X[k]` does not equal `Y[k]`. Choice (E) states this formally. Another way to approach the problem is to use DeMorgan's Law to negate the condition in the for loop. Recall that DeMorgan's Law states that

`! (p && q)` is equivalent to `!p || !q`

Negating the condition in the for loop gives

```
     ! (k < X.length && X[k] == Y[k])
=>  !(k < X.length) || !(X[k] == Y[k])
=>  k >= X.length || X[k] != Y[k]
```

This method also gives (E) as the correct answer.

36. **E** Choices (A), (B), (C), and (D) are examples of an `ArithmeticException`, a `nullPointer-Exception`, an `ArrayIndexOutOfBoundsException`, and an `IndexOutOfBoundsException`, respectively. Be careful! While (E) may appear to be an example of an `IllegalArgumentExcep-tion`, it is actually an example of an error that is caught at compile time rather than at runtime. An `IllegalArgumentException` occurs when a method is called with an argument that is either illegal or inappropriate—for instance, passing –1 to a method that expects to be passed only positive integers. Therefore, the correct answer is (E).

37. **E** First note that `a` and `b` are of type `Double`, not of type `double`. This distinction is important. The type `double` is a primitive type; the type `Double` is a subclass of the `Object` class that implements the `Comparable` interface. A `Double` is an object wrapper for a `double` value. Go through each choice one at a time. Choice (A) is incorrect. It compares `a` and `b` directly rather than the `double` values inside the objects. Even if `a` and `b` held the same value, they might be different objects. Eliminate (A). Choice (B) is incorrect. The `Double` class does not have a `notEquals` method. Eliminate (B). Choice (C) is incorrect, because `a.doubleValue()` returns a double. Since `double` is a primitive type, it does not have any methods. Had this choice been `!(a.equals(b))`, it would have been correct.

The expression `a.compareTo(b)` returns a value less than zero if `a.doubleValue()` is less than `b.doubleValue()`, a value equal to zero if `a.doubleValue()` is equal to `b.doubleValue()`, and a value greater than zero if `a.doubleValue()` is greater than `b.doubleValue()`. Choice (D) is incorrect because the `compareTo` method returns an int rather than a boolean. Choice (E) is correct. Since `compareTo()` returns 0 if and only if the two objects hold the same values, it will not return 0 if the values are different. The correct answer is (E).

38. **A** While "encapsulating functionality in a class" sounds like (and is) a good thing, "declaring all data fields to be public" is the exact opposite of good programming practice. Data fields to a class should be declared to be private in order to hide the underlying representation of an object. This, in turn, helps increase system reliability. Choices (B), (C), (D), and (E) all describe effective ways to ensure reliability in a program. The correct answer is (A).

39. **E** Go through each method one at a time. Method 1 examines at most 10 words using a binary search technique on 1,024 pages to find the correct page. Remember, the maximum number of searches using binary search is $\log_2(n)$ where n is the number of entries. It then searches sequentially on the page to find the correct word. If the target word is the last word on the page, this method will examine all 50 words on the page. Therefore, Method 1 will examine at most 60 words. Eliminate (A), (B), and (C), which don't have 60 for Method 1. The two remaining choices both have the same number for Method 2, so worry only about Method 3. Method 3 sequentially searches through all of the words in the dictionary. Because there are 51,200 words in the dictionary and the target word may be the last word in the dictionary, this method may have to examine all 51,200 words in order to find the target word. Eliminate (D), which does not have 51,200 for Method 3. Only one choice remains, so there is no need to continue. However, note that Method 2 first uses a sequential search technique to find the correct page. If the target word is on the last page, this method will examine the first word on all 1,024 pages. Then, as with Method 1, it may examine as many as all 50 words on the page to find the target word. Therefore, Method 2 will examine at most 1,074 words. The correct answer is (E).

40. **C** Pick a positive value of `j`. The question says that `j` is a positive integer. A recursive method is easiest if it takes fewer recursions to get to the base case, so try `j` = 1. If `j` = 1, then `m` = 2`j` = 2(1) = 2. Determine the return of `mystery(2)`. Since it is not the case that `m` = 0, the if condition is false, so execute the else statement, which is to return 4 + `mystery(m - 2)`. Since `m` – 2 = 2 – 2 = 0, `mystery(2)` returns 4 + `mystery(0)`. Determine the return of `mystery(0)`. In `mystery(0)`, `m` = 0, so the if condition is true, which causes the method to return 0. Therefore, `mystery(0)` returns 0, and `mystery(2)` returns 4 + `mystery(0)` = 4 + 0 = 4. Go through each choice and eliminate any that are not 4. Choice (A) is 0, so eliminate (A), (B) is `m`, which is 2, so eliminate (B). Choice (C) is 2`m`, which is 2(2) = 4, so keep (C). Choice (D) is `j`, which is 1, so eliminate (D). Choice (E) is 2`j`, which is 2(1) = 2, so eliminate (E). Only one choice remains. The correct answer is (C).

Section II: Free-Response Questions

1. (a) Sample Answer #1

```
private Elective getElectiveByName (String name)
{
  for (int eIndex = 0; eIndex < this.getElectiveListSize(); eIndex++)
  {
     Elective e = electiveList.get(eIndex);
     String eName = e.getName();
     if (name.equals(eName))
          return e;
  }
  return null;
}
```

Sample Answer #2

```
private Elective getElectiveByName (String name)
{
  int index;
  for (int eIndex; getElectiveList)
  {
     Elective e = electiveList.get(eIndex);
     String eName = e.getName();
     if (name.equals(eName))
          index = eIndex;
  }
  return electiveList.get(index);
}
```

Sample Answer #3

```
private Elective getElectiveByName (String name)
{
  int eIndex = 0;
  while (eIndex < this.getElectiveListSize())
  {
     Elective e = electiveList.get(eIndex);
     String eName = e.getName();
     if (name.equals(eName))
          Return e;
     eIndex++
  }
  return null;
}
```

The goal of the method is to return the `Elective` with the name matching the parameter. Go through `electiveList` using either a while loop, for loop, or enhanced-for loop. At each index of `electiveList`, the method must determine whether the name of the Elective matches the parameter String name. Remember that since name is a private variable, it cannot be called by another class, so the public method `getName()` must be used. Also, when comparing Strings, the `==` operator cannot be used, because this operator tests whether the objects are the same rather than whether two objects have the same value. To test whether two objects have the same value, the `equals()` method of the `String` class must be used. Once a match is found, there is no need to continue execution of the method, so the `Elective` with the matching name can be returned. Because a precondition indicates there will be a match, this return statement will be reached. However, the compiler will not recognize this. In order to avoid a compile-time error, there must be a return statement in a non-conditional statement, so add a `null` return (or any other return). An alternative method is to record the index of the match. After searching all indexes, return the `Elective` at the recorded index of `electiveList`.

(b) Sample Answer #1

```
public void assignElectivesToStudents()
{
     for (int sIndex = 0; sIndex < this.getStudentListSize(); sIndex++)
     {
          Student s = studentList.get(sIndex);
          int choice = 0;
          while (choice < 3 && !s.hasElective())
          {
                String name = s.getChoice(choice);
                Elective e = getElectiveByName(name);
                if (e.getClassSize() < e.getMaxClassSize())
                {
                      e.addStudent(s);
                      s.assignElective(e);
                }
                choice += 1;
          }
     }
}
```

Sample Answer #2

```
public void assignElectivesToStudents()
{
     for (int sIndex: this.getStudentListSize())
     {

          for (int choice = 0; choice < 3; choice++)
          {
                String name = studentList.get(sIndex).getChoice(choice);
                Elective e = getElectiveByName(name);
                if (e.getClassSize() < e.getMaxClassSize() &&
                                   !studentList.get(sIndex).hasElective())
                {
                      e.addStudent(studentList.get(sIndex));
                      s.assignElective(e);
                }
          }
     }
}
```

The method must assign each student to his or her first available choice of elective. Go through all the students, in order, using a for loop, an enhanced for loop, or a while loop. For each student, go through the student's three choices, in order, using another loop (which also could be any of the three listed above). For each choice, use the `getElectiveByName()` method from part (a) to find the matching `Elective` object, and assign the student to the class if there is room in the class (i.e., the class size is less than the max class size) and the student remains unassigned to an elective (i.e., the student's `hasElective()` method returns false). A student must be assigned to an elective using the `assignElective()` and the student must be added to the Elective's student list using the elective's `addStudents()` method.

(c) Sample Answer:

```
public ArrayList studentsWithoutElectives()
{
  ArrayList<Student> result = new ArrayList<Student>();
  for (int sIndex = 0; sIndex < this.getStudentListSize(); sIndex++)
  {
     Student s = studentList.get(sIndex);
     if (!s.hasElective())
          result.add(s);
  }
  return result;
}
```

The goal of the method is to go through each student and return an ArrayList of students who have been unassigned. First create an empty `ArrayList` of `Student` objects to be returned. Go through the school's student list using a for loop, an enhanced for loop, or a while loop. At each index, if the student at that index is unassigned (i.e., if the `hasElective()` method returns `false`), add the student to the `ArrayList`. After the end, return the `ArrayList`.

2. (a) Sample Answer #1

```
public boolean inOrder()
{
  for (int k = 0; k < cards.length; k++)
  {
     if (cards[k] != k)
          return false;
  }
  return true;
}
```

Sample Answer #2

```
public boolean inOrder()
{
  boolean temp = true;
  int k = 0;
  while (k < cards.length && temp)
  {
     if (cards[k] != k)
          temp =  false;
     k++;
  }
  return temp;
}
```

Sample Answer #3

```
public boolean inOrder()
{
  boolean temp = true;
  for (int k : cards)
  {
     if (!(cards[k] == k))
          temp =  false;
  }
  return temp;
}
```

The goal of the method is to determine whether the deck is in order and return the appropriate boolean. Because the cards in a deck of length *n* are numbered 0 through `n - 1`, an ordered deck will have all the values matching their indexes. Go through each card in the deck one at a time using a for loop, an enhanced-for loop, or a while loop. A deck is ordered only if *all* the card values match their indexes. If even one card value does not match the index, the method must return false. Therefore, the return false statement can be made immediately upon finding the card that doesn't match the index. If false is never returned in the for loop, then true must be returned. As an alternative, create a boolean outside the loop set to true. Set it to false if any card is found for which the value does not match the index. Then return the boolean after the loop.

(b) Sample Answer #1

```
public void shuffle()
{
  int[] newCards = new int[cards.length];
  for(int k = 0; k < cards.length/2; k++)
  {
     newCards[k*2] = cards[k];
     newCards[k*2+1] = cards[cards.length/2+k];
  }
  cards = newCards;
}
```

Sample Answer #2

```
public void shuffle()
{
  int[] newCards = new int[cards.length];
  for(int k : newCards)
  {
     if (k%2 == 0)
          newCards[k] = cards[k/2];
     else
          newCards[k] = cards[cards.length/2+k/2];
  }
  cards = newCards;
}
```

Sample Answer #3

```
public void shuffle()
{
  int[] front = new int[cards.length/2];
  for (int k = 0; k < front.length; k++)
     front[k] = cards[k];
  int[] back = new int[cards.length/2];
  for (int k = 0; back.length; k++)
     back[k] = cards[back.length + k];
  int[] newCards = new int[cards.length]
  int j = 0;
  while(j < newCards.length)
  {
     if (k % 2 == 1)
          newCards[k] = back[cards.length/2];
     else
          newCards[k] = front[k/2];
  }
  cards = newCards;
}
```

The goal of this method is to rearrange the values in cards according to the method described in the question. To do this, the deck must be split into a front half and a back half; then the cards must alternate, beginning with the front half. One possible approach is to directly separate the deck, as is done in Sample Answer #3. This is not necessary, though, as long as you place the first card of the deck in index 0 of the shuffled deck, the first card of the second half in index 1, the second card of the first half in index 2, the second card of the second half in index 3, and so on. A new deck of the same length as the original deck must be created. Copy the card from the original deck into the new deck, using one of the methods described above. The return is void, so the deck represented by the int array cards must be assigned the values of the new deck.

(c) Sample Answer #1

```
public int reorderingCount()
{
  int count = 0;
  while (!inOrder() || count == 0)
  {
     shuffle();
     count += 1;
  }
  return count;
}
```

Sample Answer #2

```
public int reorderingCount()
{
  Shuffle();
  int count;
  for (count = 1; !inOrder(); count++)
     shuffle();
  return count;
}
```

The goal of the method is to shuffle the deck repeatedly until it is back in its original order, and return the number of times it was shuffled. Create a counter by initializing an `int` and setting it equal to zero. Make sure to shuffle at least once. Use a for or while loop to do repeated shuffles until it is back in order, incrementing the counter at each shuffle, including the first. The loop must stop when the deck is back in order, so it must continue while `inOrder()` is `false`. After it is back in order, the loop must terminate, and the counter variable must be returned by the method.

3. (a)

```
public class Recipe
{
  private String name;
  private ArrayList<Ingredient> ingredientList;
  private String preparationProcess;
  private int numberServed;
  public ArrayList<Ingredient> getIngredientList()
     { /* implementation not needed */ }
  public Recipe(String recipeName, int numServed)
     { /* implementation not needed */ }
  public void addIngredient(Ingredient newIngredient)
     { /* implementation not needed */ }
  public void setPreparationProcess(String newPreparationProcess)
     { /* implementation not needed */ }
  public String getName()
     { /* implementation not needed */ }
  public int getNumberServed()
     { /* implementation not needed */ }
  public void scale(int newNumberServed)
     { /* implementation not needed */ }
}
```

Be sure to list all the needed variables with the appropriate data types as described in the question. Both the name of the recipe and the description of the preparation process are text, so use String types. The list of ingredients will need to be of a variable size, so use an `ArrayList`. Use an int for the `numberServed`. Also be sure to list out the method signatures as described in the question. To create a recipe with a given name and number of people served, use a constructor method with String and int parameters. To add an ingredient, use a method that takes an ingredient as a parameter. Since this method will alter the already existing ingredients list within the `Recipe` object rather than create a new list, the return should be void. Similarly, the method to set the description will alter an already existing data within the `Recipe` object, so take a String parameter and have a void return. To return the name of the recipe, number of people served, or the list of ingredients, take no parameters and return the appropriate data type. The method to scale the amount based on a new number of people served must take the new number of people served as a parameter and alter already existing `Ingredient` objects and the `NumberServed` integer so it should have a void return.

(b) Sample Answer #1

```
public void scale(int newNumberServed)
{
  double oldAmount;
  double newAmount;
  for (int k = 0; k < ingredientList.size; k++)
  {
     Ingredient ingred = (Ingredient) ingredientList.get(k);
     oldAmount = ingred.getAmount();
     newAmount = newNumberServed * (oldAmount / numberServed);
     ingred.setAmount(newAmount);
  }
  numberServed = newNumberServed;
}
```

Sample Answer #2

```
public void scale(int newNumberServed)
{

  for (int k : ingredientList)
  {
     double oldAmount = (Ingredient) ingredientList.get(k).getAmount();
     double newAmount = newAmount = (oldAmount * newNumberServed) /
numberServed;
     ingredientList.get(k).setAmount(newAmount);
  }
  numberServed = newNumberServed;
}
```

There are two goals for this recipe. One is to change the number of people served by the recipe. This can be achieved by one command: setting the class variable for the number of people served equal to the parameter. However, this should be done at the end of the method, so that the original number of people served can be used. The other goal is to scale the amount of each ingredient to the new number of people. Go through each ingredient in ingredientList using a for loop, an enhanced for loop, or a while loop. For each ingredient, record the original amount. Get the new amount by using an appropriate scaling. To determine how to do this, use a proportion: $\frac{oldAmount}{newAmount} = \frac{numberServed}{newNumberServed}$. To find an expression for newAmount, cross-multiply to get newAmount * numberServed = oldAmount * newNumberServed. Divide both sides by numberServed to get `newAmount = (oldAmount * newNumberServed) / numberServed`. (An alternative but equivalent formula is shown in Sample Answer # 1.) Use the `setAmount()` method of the ingredient, passing newAmount as a parameter to update the amount of the ingredient. Finally, end the loop and set the number served to the new amount as discussed above.

(c) Sample Answer #1

```
public void standardize(int numPeople)
{
  double oldAmount;
  double newAmount;
  for (int j = 0; j < recipeList.size(); j++)
  {
     for (int k = 0; k < recipeList.size(); k++)
     {
          Ingredient ingred = (Ingredient) recipeList.
          get(j).get
          IngredientList().get(k);
          oldAmount = ingred.getAmount();
          newAmount = numPeople / oldAmount;
          ingred.setAmount(newAmount);
     }
     recipeList.get(j).setNumberServed(numPeople);
  }
}
```

Sample Answer #2

```
public void standardize(int numPeople)
{
  int j = 0
  while (j < recipeList.size())
  {
     for (int k: recipeList.size())
     {
          (Ingredient) recipeList.get(j).getIngredientList().get(k).
ingred.scale(newAmount);
     }
     j++;
  }
}
```

The goal of the method is to apply the `setAmount()` method to all recipes using a predetermined number of people. Take the number of people as a parameter. Because the method is intended to alter already existing objects rather than create new objects, the return should be void. Go through each recipe using a for loop, an enhanced-for loop, or a while loop. For each recipe, use an inner loop of any of the three types to go through each ingredient and scale it to the new amount. This can be done in a similar method to what was done in part (b) or can be done simply by using the scale method.

4.

```
public class School
{
  private ArrayList<Classroom> classrooms;
  public School(ArrayList<Classroom> schoolRooms)
  {
     classrooms = SchoolRooms;
  }
  public String findStudent(String teacher, int IDnumber)
  {
     for (int k = 0; k < classrooms.size(); k++)
     {
          if (classrooms.get(k).getTeacherName().equals(teacher))
          {
                 int low = 0;
                 int high = classrooms.get(k).getStudents().size() - 1;
                 while (low <= high)
                 {
                        int middle = (low + high) / 2;
                        if (IDnumber < classrooms.get(k).getStudents().
get(middle).getStudentID())
                        {
                               high = middle - 1;
```

```
                       }
                       else if (IDnumber >
classrooms.get(k).getStudents().get(middle).getStudentID())
                       {
                             low = middle + 1;
                       }
                       else
                       {
                             return classrooms.get(k).getStudents().
get(middle).getStudentName();
                       }
                 }
           }
      }
      return “Student Not Found”;
  }
}


public class Classroom
{
  private String teacherName;
  private ArrayList<Student> Students;
  public Classroom(String teacher, ArrayList<Student> theStudents)
  {
     teacherName = teacher;
     Students = theStudents;
  }
  public String getTeacherName()
  {
     return teacherName;
  }
  public ArrayList<Student> getStudents()
  {
     return Students;
  }
}


public class Student
{
  private String studentName;
  private int studentID;
  public Student(String name, int ID)
  {
     studentName = name;
     studentID = ID;
  }
  public int getStudentID()
  {
     return studentID;
  }
  public String getStudentName()
  {
     return studentName;
  }
}
```

The question specifies that there must be three classes: `School`, `Classroom`, and `Student`. Create a class definition for each, including a constructor taking the required data fields as parameters. School must take `ArrayList<Classroom> schoolRooms`, `Classroom` must take `String teacherName` and `ArrayList<Student> theStudents`, and `Student` must take `String name` and `int ID`. The only method specified by the question is `findStudent` in the `School` class. However, the description of this method indicates that it will require information from `Classroom` and `Student` objects. Since, on the AP Exam, all variables should be specified as private, this will require the use of public methods in the `Classroom` and `Student` classes. The variables in `Classroom` are `teacherName` and `students`, so create a `getTeacherName()` method returning a `String` with the same value as `teacherName` and a `getStudents` to return the `ArrayList Students`. The variables in `Student` are `studentID` and `studentName`, so create a `getStudentID()` method returning an integer equal to `studentID` and a `getStudentName()` method returning a String with the same value as `studentName`.

To create the `findStudent` method, the question specifies that it must take a String for teacher name and an integer for student ID as parameters. The question also specifies that a sequential search must be used to find the classroom based on teacher name. A sequential search can be most easily accomplished using a for loop or an enhanced for loop (though a while loop is also possible). Search through the entire list of classrooms, one at a time, to determine whether the teacher parameter matches the `teacherName` in the `Classroom`. To do this, be sure to get the `getTeacherName()` method of the `Classroom` object, since `teacherName` is a private variable. Also, be sure to use the `equals()` method of the String class, since the == operator returns true only if the String objects are the same name rather than if they have the same value. If a match is found, search the classroom for the student using `ID`. The question specifies that a binary search must be used. Use a standard binary search. If a match is found, return the name of the student using the `getStudentName()` method of the `Student` object. If no match is found, nothing will be returned by the searches. Outside the two searches, return `"Student Not Found"` as specified by the question.