Practice Exam 1 Answers and Explanations

Part I Answers and Explanations (Multiple Choice)

Bullets mark each step in the process of arriving at the correct solution.

1. The answer is E.

- Lines 1 and 2 give us: myValue = 17 and multiplier = 3
- Line 3: answer = 17 % 3 + 17 / 3
  - Using integer division: 17 / 3 = 5 remainder 2, giving us:
  - 17 / 3 = 5
  - 17 % 3 = 2
  - The expression simplifies to answer = 2 + 5 = 7 (remember order of operations)
- Line 4: answer = 7 * 3 = 21

2. The answer is B.

- Let's use De Morgan's theorem to simplify the expression.

```
!(a < b || b <= c) && !(a < c || b >= a)
```

- Distribute the first !. Remember to change || to &&. This gives:

```
!(a < b) && !(b <= c)
```

- Distribute the second ! This gives:

```
!(a < c) && !(b >= a)
```

- It's easy to simplify all those !s. Remember that !< → >= and !<= → > (the same with > and >=).

```
(a >= b) && (b > c) && (a >= c) && (b < a)
```

- Since these are all &&s, all four conditions must be true. The first condition says a >= b and the fourth condition says b < a. No problem there. Options B and C both have a > b. Option A can be eliminated.
- The second condition says b > c. That's true in option B. Option C can be eliminated.
- The third condition says b < a. Still true in option B, so that's the answer.
- You could also solve this problem with guess and check by plugging in the given values, but all those ands and ors and nots tend to get pretty confusing. Simplifying at least a little will help, even if you are going to ultimately plug and chug.

3. The answer is D.

- The `while` loop condition is (myArray[index] < 7). If you did not read carefully, you may have assumed that the condition was (index < 7), which, along with the index++ is the way you would write it if you wanted to access every element in the array.
- Because the condition is (myArray[index] < 7), we will start at element 0 and continue until we come to an element that is greater than or equal to 7. At that point we will exit the loop and no more elements will be processed.
- Every element before the 7 in the array will have 3 added to its value.

4. The answer is A.

- Option II uses a `for-each` loop. That is an excellent option for this problem, since we need to process each element in exactly the same way, but option II does not access the element correctly. Read the `for-each` loop like this: "For each OnlinePurchaseItem (which I will call purchase) in items. . . ." The variable purchase already holds the needed element. Using get(i) to get the element is unnecessary and there is no variable i.
- Option III uses a `for` loop and processes the elements in reverse order. That is acceptable. Since we need to process every item, it doesn't matter what order we do it in. However, option III starts at i = items.size(), which will cause an `IndexOutOfBoundsException`. Remember that the last element in an `ArrayList` is list.size() − 1 (just like a String or an array).
- Option I correctly accesses and processes every element in the `ArrayList`.

5. The answer is D.

- This is a recursive method. Let's trace the calls. The parts in *italics* were filled in on the way back up. That is, the calls in the plain type were written top to bottom until the base case returned a value. Then the answers were filled in *bottom to top.*
  - mystery(5) = 2 + mystery(4) = *2 + 7 =* ***9*** which gives us our final answer.
  - mystery(4) = 2 + mystery(3) = *2 + 5 = 7*
  - mystery(3) = 2 + mystery(2) = *2 + 3 = 5*
  - mystery(2) = 2 + mystery(1) = *2 + 1 = 3*
  - mystery(1) Base Case! return 1

6. The answer is E.

- The outer loop will execute three times, starting at index = 0 and continuing as long as index < 4, increasing by one each time through. So index will equal 0, 1, 2, 3 in successive iterations through the loop.
- The only statement inside the outer loop is the inner loop. Let's look at what the inner loop does in general terms.
- The inner loop executes from i = 0 to i < index, so it will execute index times.
- Each time through it puts an "A" in front of and after myString.
- Putting it all together:
  - The first iteration of the outer loop, index = 0, the inner loop executes 0 times, myString does not change.
  - The second iteration of the outer loop, index = 1, the inner loop executes one time, adding an "A" in front of and after myString. myString = "AHA"
  - The third iteration of the outer loop, index = 2, the inner loop executes two times, adding two "A"s in front of and after myString. myString = "AAAHAAA"
  - The fourth (and last) iteration of the outer loop, index = 3, the inner loop executes three times, adding three more "A"s in front of and after myString. myString = "AAAAAAHAAAAAA"
- Putting it another way, when index = 1, we add one "A"; when index = 2, we add two "A"s; when index = 3, we add three "A"s. That's six "A"s all together added to the beginning and end of "H" → "AAAAAAHAAAAAA"

7. The answer is B.

- The general form for generating a random number between *high* and *low* is:

```
(int)(Math.random() * (high - low + 1)) + low
```

- high − low + 1 = 50, low = 10, so high = 59
- The correct answer is integers between 10 and 59 inclusive

8. The answer is C.

- This question requires that you understand the two uses of the + symbol.
- First, we execute what is in the parentheses. Now we have:

```
13 + 6 + "APCSA" + 4 + 4
```

- Now do the addition left to right. That's easy until we hit the string:

```
19 + "APCSA" + 4 + 4
```

- When you add a number and a string in any order, Java turns the number into a string and then concatenates the strings:

```
"19APCSA" + 4 + 4
```

- Now every operation we do will have a string and a number, so they all become string concatenations, not number additions.

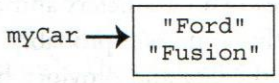
```
"19APCSA44"
```

9. The answer is B.

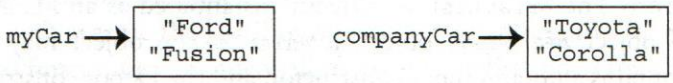
- This is a recursive method. Let's trace the calls. The parts in *italics* were filled in on the way back up. That is, the calls in the plain type were written top to bottom until the base case returned a value. Then the answers were filled in *bottom to top.*
- Be very careful to get the order of num1 and num2 correct in the modulus operation and the parameters correct in the recursive call.
  - guess(3, 17) = guess(17, 3) + 3 = *4* + *3* = *7* which gives us our final answer.
  - guess(17, 3) = guess(3, 2) + 2 = *2* + *2* = *4*
  - guess(3, 2) = guess(2, 1) + 1 = *1* + *1* = *2*
  - guess(2,1) Base case! return 3/2 = 1 (integer division)

10. The answer is A.

- myCar is instantiated as a reference variable pointing to an `Automobile` object with instance variables make = "Ford" and model = "Fusion".

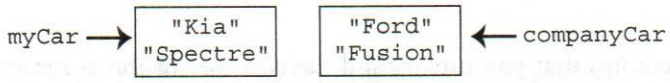


- companyCar is instantiated as a reference variable pointing to an `Automobile` object with instance variables make = "Toyota" and model = "Corolla".



- The statement companyCar = myCar reassigns the `companyCar` reference variable to point to the same object as the `myCar` variable. They are now aliases of each other. The original `companyCar` object is now inaccessible (unless yet another variable we don't know about points to it).

- The next statement instantiates a brand-new `Automobile` object with make = "Kia" and model = "Spectre". It sets the `myCar` reference variable to point to this new object. This does not change the `companyCar` variable.
- The `companyCar` variable still points to the object with instance variables make = "Ford" and model = "Fusion".



- So we print companyCar's make, which is "Ford" followed by myCar's model, which is "Spectre".

11. The answer is E.

- Methods or constructors with the same name (and return type, in the case of methods) may have different parameter lists. The parameters may differ in type or number. This is called *overloading*.

12. The answer is A.

- We can eliminate options B and D immediately by noticing that active is a `boolean` and will therefore print as either "true" or "false" and not as "active" even though that is more informative.
- We would expect option E to be correct by reading the code, but the constructor was called incorrectly. The overloaded constructor that is being called is the two-parameter constructor, and that constructor is going to assign its first parameter to state and its second parameter to city. Since we passed city, state, in that order, the city name and the state name have been assigned incorrectly and will therefore be printed incorrectly.
- Note that this is a confusing way to write an overloaded constructor! The order of the variables changes for no reason.

13. The answer is D.

- Option I is incorrect. Most classes have a no-argument constructor; either the default constructor provided automatically if no other constructor is provided, or one written by the programmer. However, the `Depot` class does not. The super call causes a compile-time error.
- Option II is correct. It implements a no-argument constructor in the `WhistleStop` class, which passes the needed parameters to the Depot constructor. It's hard to imagine why you would want to default to constructing a station in the tiny town of Waubaushene, but if you want to do that, this code will do it for you.
- Option III is also correct. It takes two parameters and correctly passes them to the Depot constructor via the call to super. However, this code will probably not give you the result you are expecting. Just like in question 12, the order of the city and province has been switched.

14. The answer is A.

- Option I is correct. The `ArrayList` is correctly instantiated as an `ArrayList` of `Depot` objects. Since a `WhistleStop` object *is-a* `Depot` object, a `WhistleStop` object may be added to the `ArrayList`. `WhistleStop` contains a no-argument constructor, and the Depot constructor is called correctly as well.
- Option II is incorrect. A `WhistleStop` *is-a* `Depot`, but not the other way around. We cannot put `Depot` objects into an `ArrayList` that is declared to hold `WhistleStop` objects. The `ArrayList` declaration will generate a compile-time error.
- Option III is incorrect. The `ArrayList` is correctly instantiated as an `ArrayList` of `WhistleStop` objects. We can add `WhistleStop` objects to this `ArrayList`, but we cannot add `Depot` objects. The second add will generate a compile-time error.

15. The answer is B.

- When we start the `for-each` loop, the variable crazyString = "crazy" and the `ArrayList` crazyList has two elements ["weird," "enigma"].
- You can read the `for-each` like this: "For each String (which I am going to call s) in crazyList. . . ."
- So for each String s in crazyList we execute the assignment statement:

```
crazyString = s.substring(1, 3) + crazyString.substring(0, 4);
```

which takes two characters starting at index 1 from s and concatenates the first four characters of crazyString.

- The first time through the loop, s = "weird" and crazyString = "crazy" so the assignment becomes:

```
crazyString = "ei" + "craz" = "eicraz"
```

Remember that substring starts at the first parameter and ends *before* the second parameter, and remember that we start counting the first letter at 0.

- The second time through the loop, s = "enigma" and crazyString = "eicraz" so the assignment becomes:

```
crazyString = "ni" + "eicr" = "nieicr"
```

16. The answer is E.

- There are four possible paths through the code. We need one data element that will pass through each path.
  - The first path is executed if n < -5.
  - The second path is executed if -5 ≤ n < 0 (because if n < -5, the first path is executed and we will not reach the second path).
  - The third path is executed if n > 10.
  - The fourth path is executed in all other cases.
- We need to be sure that we have at least one piece of data for all four of those cases.
- Option A does not test the fourth path, because 12 and 15 are both > 10.
- Option B does not test the first path, because it doesn't have a number that is < -5.
- Option C does not test the third path, because it doesn't have a number that is > 10.
- Option D does not test the second path, because 0 is not less than 0.
- Option E tests all four paths.

17. The answer is D.

- This statement should be written with if-else ladder. When the clause associated with a true condition is executed, the rest of the statements should be skipped. Unfortunately, that is not how it is written. For example, if the parameter temp = 80, the first if is evaluated and found to be true and activity = "go swimming"; however, since there is no else, the second if is also evaluated and found to be true, so activity = "go hiking". Then the third if is evaluated and found to be true, so activity = "go horseback riding". The else clause is skipped and "go horseback riding" is returned (incorrectly).
- All temperatures that are true for multiple `if` statements will cause activity to be set incorrectly. Only the values of temp that will execute the last if or the else clause will return the correct answer.
- In order for this code to function as intended, all ifs except the first one should be preceded with *else*.

18. The answer is D.

- This time, there is an if-else ladder, but they are in the wrong order. It is important to write the most restrictive case first. Let's test this code by using 80 degrees as an example. 80 > 75, and that is the condition we intend to execute, but unfortunately, it is also > 45, which is the first condition in the code, so that is the if clause that is executed and activity is set to "go horseback riding". Since the code is written as a cascading if-else, no more conditions are evaluated and flow of control jumps to the return statement.
- Two sets of values give the right answer:
  - Values that execute the first if clause correctly—that is, values that are greater than 45, but less than or equal to 60.
  - Values that fail all the if conditions and execute the else clause—that is, values that are less than or equal to 45.
- Those two conditions can be combined into temperatures <= 60.
- In order for this code to function as intended, the order of the conditions needs to be reversed.

19. The answer is A.

- Let's look at the three code segments one by one.
- Option I:
  - When we enter the loop, i = 1.
  - arr[1 / 2] = arr[0] = 1 (don't forget integer division)
  - i = i + 2 = 3, 3 < 6 so we enter the loop again.
  - arr [3 / 2] = arr[1] = 3
  - i = i + 2 = 5, 5 < 6 so we enter the loop again.
  - arr[5 / 2] = arr[2] = 5
  - i = i + 2 = 7, 7 is not < 6 so we exit the loop.
  - The array arr = [1, 3, 5]
- Option II:
  - When we enter the loop, i = 0.
  - arr[0] = 2 * 0 + 1 = 1
  - increment i to 1, 1 < 3 so we enter the loop again.
  - arr[1] = 2 * 1 + 1 = 3
  - increment i to 2, 2 < 3 so we enter the loop again.
  - arr[2] = 2 * 2 + 1 = 5
  - increment i to 3, 3 is not < 3 so we exit the loop.
  - The array arr = [1, 3, 5]
- Option III:
  - When we enter the loop, i = 6.
  - arr[(6 − 6) / 2] = arr[0] = 6 − 6 = 0
  - Since we know the first element should be 1, not 0, we can stop here and eliminate this option.
- Options I and II produce the same results.

20. The answer is C.

- Let's picture our `ArrayList` as a table. After the add statements, `ArrayList` nations looks like this:

Argentina	Canada	Australia	Cambodia	Russia	France

- It looks like perhaps the loop is intended to remove all elements whose length is greater than or equal to 7, but that is not what this loop does. Let's walk through it.
- i = 0. nations.size() = 6, 0 < 6 so we enter the loop.
  - nations.get(0).length() gives us the length of "Argentina" = 9.
  - 9 >= 7, so we remove the element at location 0. Now our `ArrayList` looks like this:

Canada	Australia	Cambodia	Russia	France

- increment i to 1. nations.size() = 5, 1 < 5, so we enter the loop.
  - nations.get(1).length() gives us the length of "Australia" = 9. (We skipped over "Canada" because it moved into position 0 when "Argentina" was removed.)
  - 9 >= 7, so we remove the element at location 0. Now our `ArrayList` looks like this:

Australia	Cambodia	Russia	France

- increment i to 2. nations.size() = 4, 2 < 4 so we enter the loop.
  - nations.get(2).length() gives us the length of "Russia" = 6.
  - 6 is not >= 7 so we skip the if clause.
- increment i to 3. nations.size() = 4, 3 < 4 so we enter the loop.
  - nations.get(3).length() gives us the length of "France" = 6.
  - 6 is not >= 7 so we skip the if clause.
- increment i to 4. nations.size() = 4, 4 is not < 4 so we exit the loop with the `ArrayList`:

Australia	Cambodia	Russia	France

- Note: You have to be really careful when you remove items from an `ArrayList` in the context of a loop. Keep in mind that when you remove an item, the items with higher indices don't stay put. They all shift over one, changing their indices. If you just keep counting up, you will skip items. Also, unlike an array, the size of the `ArrayList` will change as you delete items.

21. The answer is C.

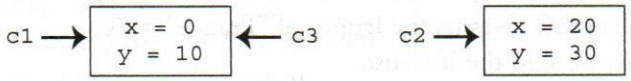
- doStuff is going to loop through the array, checking to see if each item is the same as the item before it. If it is, the values of index, maxCounter, and counter all change. If it is not, only counter changes.
- Begin by setting index = 0; maxCounter = 1; counter = 1; and noting that numberArray.length() = 5.
- First iteration: k = 1, enter the loop with index = 0; maxCounter = 1; counter = 1;
  - Item 1 = item 0, execute the if clause
    - counter <= maxCounter, execute the if clause
      - maxCounter = 1; index = 1
    - counter = 2
- Second iteration: k = 2, enter the loop with index = 1; maxCounter = 1; counter = 2;
  - Item 2 != item 1, execute the else clause
    - counter = 1
- Third iteration: k = 3, enter the loop with index = 1; maxCounter = 1; counter = 1;
  - Item 3 = item 2, execute the if clause
    - counter <= maxCounter, execute the if clause
      - maxCounter = 1; index = 3
    - counter = 2
- Fourth iteration: k = 4, enter the loop with index = 3; maxCounter = 1; counter = 2;
  - Item 4 != item 3, execute the else clause
    - counter = 1
- Fifth iteration: k = 5, which fails the condition and the loop exits.
- index = 3, numberArray[3] = 4 is returned and printed.

22. The answer is D.

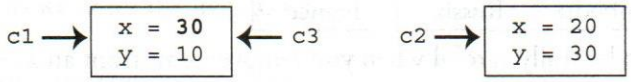
- This question is explained along with question 23.

23. The answer is D.

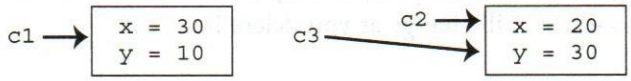
- We will solve questions 22 and 23 together by diagramming the objects and reference variables.
- The first three assignment statements give us:



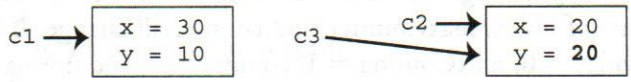
- `c3.setX(c2.getY());` results in:



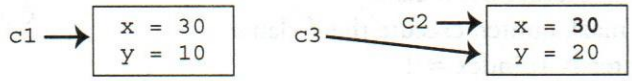
- `c3 = c2;` results in:



- `c3.setY(c2.getX());` results in:



- `c2.setX(c1.getX());` results in:



- Question 22: Both c2.getY() and c3.getY() return 20, but only c2.getY() is an option.
- Question 23: c1.getX(), c2.getX(), and c3.getX() all return 30.

24. The answer is C.

- This method takes a String parameter and loops through the first half of the letters in the string.
- The first substring statement assigns the letter at index i to sub1. You may have to run through an example by hand to discover what the second substring does.
- If our string is "quest":
  - the first time through the loop, i = 0, so sub1 = "q" and sub2 = st.substring(5 − 0 − 1, 5 − 0) = "t",
  - the second time, i = 1, so sub1 = "u" and sub2 = st.substring(5 − 1 − 1, 5 − 1) = "s".
- This loop is comparing the first letter to the last, the second to the second to last, and so on, which will tell us if the string is a palindrome.

25. The answer is C.

- To answer this problem, you need to understand that `String` objects are immutable (the only other immutable objects in our Java subset are Integer and Double.
- When the method begins, entry1 = "AP" and entry2 = "CS".
- Then the assignment statement reassigns entry2 so that both variables now refer to the `String` object containing "AP".
- When we execute the next statement, entry1 = entry1 + entry2, we would expect that to change for both variables, since they both reference the same object, but `String` objects don't work that way. Whenever a string is modified, a new `String` object is created to hold the new value. So now, entry1 references "APAP" but entry2 still references "AP".
- We put those together, and "APAPAP" is returned.

- When an object is passed to a method, its reference is copied into the formal parameter. The actual and formal parameters become aliases of each other; that is, they both point to the same object. Therefore, when the object is changed inside the method, those changes will be seen outside of the method. Changes to array values inside the method will be seen in array values outside of the method.
- num1 remains equal to 2, even though the method's num1 variable is changed to 4. num2 is reset to 4, because the method returns that value, and values[4] is set to 3.

31. The answer is C.

- Let's figure this out step by step. The length of "on your side" is 12 (don't forget to count spaces), so `s1.indexOf(s2.substring(s2.length() - 2))` simplifies to `s1.indexOf(s2.substring(10))`
- That gives us the last two letters of s2, so our statement becomes `s1.indexOf("de")`
- Can we find "de" in s1? Yes. It starts at index 3 and that is what is returned.

32. The answer is A.

- acArray is an array of objects. Initially, all the entries in the array are null. That is, they do not reference anything. If a program tries to use one of these entries as if it were an object, the program will terminate with a `NullPointerException`.
- To avoid the `NullPointerException`, we must not attempt to use a null entry as if it contained a reference to an object.
- Option B is incorrect. It uses `ArrayList` syntax, not array syntax, so it will not even compile.
- Options C and D are incorrect. They attempt to call getData() on a null entry and so will terminate with a `NullPointerException`.
- Option A is correct. It checks to see whether the array entry is null before allowing the System.out.println statement to treat it as an object.

33. The answer is A.

- The Binary Search algorithm should not be applied to this array! Binary Search only works on a sorted array. However, the algorithm will run; it just won't return reasonable results. Let's take a look at what it will do.
- First look at the middle element. That's 100. 50 < 100, so eliminate the second half of the array.

9	100	11	45	76	~~100~~	~~50~~	~~1~~	~~0~~	~~55~~	~~99~~

- Look at the middle element of the remaining part; 50 > 11, so eliminate the lower half.

~~9~~	~~100~~	~~11~~	45	76	~~100~~	~~50~~	~~1~~	~~0~~	~~55~~	~~99~~

- 50 does not appear in the remaining elements, return -1 (incorrectly, as it turns out).

34. The answer is C.

- After the array is instantiated, it looks like this:

0	1	2
3	4	5
6	7	8

- The `for` loop will execute three times: x = 0, x = 1, and x = 2.
- You should recognize that the code in the loop is a simple swap algorithm that swaps the value at nums[x][0] with the value at nums[x][2].
- The result is that in each row, element 0 and element 2 will be swapped.
- Our final answer is:

2	1	0
5	4	3
8	7	6

35. The answer is A.

- Let's trace the loop and see what happens. On entry we have:

  list (which will not change):

2	4	3	3	2	5	1

  newList:

2

- i = 1. In pseudocode, the `if` statement says "if (element 1 of list > the last element in newList) add it to newList". Since 4 > 2, add 4 to newList, which becomes:

2	4

- i = 2. "if (element 2 of list > the last element in newList) add it to newList" 3 is not > 4 so we do not add it.
- i = 3. "if (element 3 of list > the last element in newList) add it to newList" 3 is not > 4, we do not add it. (By this point, you may recognize what the code is doing and you may be able to complete newList without tracing the remaining iterations of the loop.)
- i = 4 "if (element 4 of list > the last element in newList) add it to newList" 2 is not > 4, we do not add it.
- i = 5. "if (element 5 of list > the last element in newList) add it to newList" 5 > 4, add 5 to newList which becomes:

2	4	5

- i = 6. "if (element 6 of list > the last element in newList) add it to newList" 1 is not > 5, so we do not add it.
- We have completed the loop; we exit and print newList.

36. The answer is E.

- After the array is instantiated, it is filled with 0s because that is the default value for an int, so anything we don't overwrite will be a 0.
- Let's look at the first loop. r goes from 0 to 4, so all rows are affected. c starts at r+1 and goes to 4, so not all columns are affected. When r = 0, c will = 1,2,3,4; when r = 1, c will = 2,3,4; when r = 2, c will = 3,4; when r = 3, c will = 4. That will assign 9 to the upper-right corner of the grid like this.

0	9	9	9	9
0	0	9	9	9
0	0	0	9	9
0	0	0	0	9
0	0	0	0	0

- In the second loop, d goes from 0 to 4, so all rows are affected, and c always equals r, so the diagonals are filled in with their row/column number.

0	9	9	9	9
0	1	9	9	9
0	0	2	9	9
0	0	0	3	9
0	0	0	0	4

- Now we just have to look for the three cells specified in the problem.

37. The answer is D.

- The first thing to do is to identify the sorting algorithm implemented by mysterySort.
- We can tell it's not Merge Sort, because it is not recursive.
- There are several things we can look for to identify whether a sort is Insertion Sort or Selection Sort. Some easy things to look for:
  - In the inner loop, Insertion Sort shifts items over one slot.
  - After the loops are complete, Selection Sort swaps two elements.
- We can see items being shifted over and we can't see a swap, so this is Insertion Sort.
- Let's look at the state of the array in table form. Before the sort begins, we have this. The sort starts by saying that 9 (all by itself) is already sorted.

9	1	3	0	2

- The first iteration through the loop puts the first two elements in sorted order.

1	9	3	0	2

- The second iteration through the loop puts the first three elements in sorted order.

1	3	9	0	2

- Notice that in Insertion Sort, the elements at the end of the array are never touched until it is their turn to be "inserted." Selection Sort will change elements all through the array.

38. The answer is C.

- This is a recursive method. Let's trace the calls. The parts in *italics* were filled in on the way back up. That is, the calls in the plain type were written top to bottom until the base case returned a value. Then the answers were filled in *bottom to top.*
  - puzzle (3, 4) = 3 * puzzle (3, 3) = *3 * 27 =* ***81*** which gives us our final answer
  - puzzle (3, 3) = 3 * puzzle (3, 2) = *3 * 9 = 27*
  - puzzle (3, 2) = 3 * puzzle (3, 1) = *3 * 3 = 9*
  - puzzle (3, 1) Base Case! return 3
- It is interesting to notice that this recursive method finds the first parameter raised to the power of the second parameter.

39. The answer is D.

- This nested `for` loop is traversing the array in column major order. I can tell this is the case because the outer loop, the one that is changing more slowly, is controlling the column variable. For every time the column variable changes, the row variable goes through the entire row.
- In addition, although the outer loop is traversing the columns from least to greatest, the inner loop is working backward through the rows.
- Since we increase val by 2 each time, values are being filled in like this:

8	18	28	38	48	58
6	16	26	36	46	56
4	14	24	34	44	54
2	12	22	32	42	52
0	10	20	30	40	50

- You probably didn't need to fill in the whole table to figure out that table[3][4] = 42.

40. The answer is E.

- Since this is a Selection Sort algorithm, we know that the inner loop is looking for the smallest element.
- small is storing the index of the smallest element we have found so far. We need to find out if the current element being examined is smaller than the element at index small, or, in other words, if arr[j] is less than arr[small].
- Option C would be correct if we were sorting an array of ints, but < doesn't work with strings. We have to use the method `compareTo`. `compareTo` returns a negative value if the calling element is before the parameter value. We need:

```
if (arr[j].compareTo(arr[small] < 0))
```

Part II Answers and Explanations (Free Response)

Please keep in mind that there are multiple ways to write the solution to a Free-response question, but the general and refined statements of the problem should be pretty much the same for everyone. Look at the algorithms and coded solutions, and determine if yours accomplishes the same task.

General Penalties (assessed only once per problem)**:**
-1 using a local variable without first declaring it
-1 returning a value from a void method or constructor
-1 confusing array/ArrayList access
-1 overwriting information passed as a parameter
-1 including unnecessary code that causes a side effect such as a compile error or console output

1. Password

(a) **General Problem:** Write the `isValid` method for the `Password` class.
Refined Problem: Determine if a password is valid by comparing the checking if the password has the correct length and the correct combination of letters and symbols.

Algorithm:
- Determine the length of the password.
- If the length is not between the minimum length and the maximum length, then return false. Otherwise, continue checking.
- Initialize counters for the number of uppercase letters, lowercase letters, and symbols to zero.
- For each character in the password increment the appropriate counter.
- If the counter for each isn't zero (meaning there was at least one occurrence) and the sum of all three counters equal the length of the password (meaning there were no symbols outside those that are acceptable), then it is a valid password.
- Return the result.

Java Code:

```
public boolean isValid(String password)
{
if (password.length() < minLength || password.length() > maxLength)
     return false;
int countUpper =0, countLower = 0, countSymbol = 0;
for (int i = 0; i < password.length(); i++)
{
     String letter = password.substring(i, i+1);
     if (upper.indexOf(letter) > -1)
          countUpper++;
     else if (lower.indexOf(letter) > -1)
          countLower++;
     else if (symbols.indexOf(letter) > -1)
          countSymbol++;
  }
return (countUpper > 0) && (countLower > 0) && (countSymbol > 0)
      && (countUpper + countLower + countSymbol == password.length());
  }
```

Common Errors:

- Not taking into account that there might be characters other than letters or acceptable symbols.
- Not checking for the length of the password.
- Not initializing or declaring the necessary variables.

(b) **General Problem:** Write the `generatePassword` method for the `Password` class.
Refined Problem: Use the random number generator to choose upper case, lower case, and symbols from the given strings. Determine if the generated password is valid by using the method written in part (a).

Algorithm:

- Concatenate the upper case, lower case and symbol strings.
- Determine if a valid password has been generated.
  - Use the random number generator to choose a valid length of the password.
  - Use the random number generator to randomly choose a letter from the concatenated upper/lower/symbol string.
  - Continue randomly choosing letters until the string has reached the desired length.
  - Test to see if the password is valid.
- If the password is not valid, start the process again until a valid password has been generated.
- Return the generated password string.

Java Code:

```
public String generatePassword()
{
     String password = "";
     String allPossible = upper + lower + symbols;
     int position;
     while (!isValid(password))
     {
       password = "";
       int length = (int) (Math.random() * (maxLength - minLength + 1)) + 1;
       for (int i = 1; i <= length; i++)
       {
            position = (int) (Math.random() * allPossible.length());
            password += allPossible.substring(position, position+1);
       }
     }
     return password;
}
```

Common Errors:

- Not checking to see if the generated password is valid.
- Not randomly choosing a length for the new password.
- Not initializing or declaring necessary variables.
- Not returning the generated password.

Scoring Guidelines: Password

Part (a) isValid 5 points

+1 Determine if the password has a valid length
+1 Examine each character of the password one at a time
+1 Determine if the password has upper case, lower case, and valid symbols
+1 Ensure there are no other characters besides upper case, lower case, and valid symbols
+1 Return the appropriate value

Part (b) generatePassword 4 points

+1 Calls the `isValid` method correctly
+1 Randomly chooses a length of the password
+1 Randomly choose upper case, lower case, and valid symbols
+1 Returns the generated password

Sample Driver:

There are many ways to write these methods. Maybe yours is a bit different from our sample solutions and you are not sure if it works. Here is a sample driver program. Running it will let you see if your code works, and will help you debug it if it does not.

Copy PasswordDriver into your IDE along with the complete `Password` class (including your solutions).

```
public class PasswordDriver
{
   public static void main(String[] args)
   {
     Password pass1 = new Password(3,15);
     String pass = "HelloWorld!";
     boolean valid = pass1.isValid(pass);
     System.out.println(pass + " is valid: " + valid);
     pass = pass1.generatePassword();
     System.out.println(pass);

     Password pass2 = new Password(3,15);
     pass = "#APComputerScienceRocks";
     valid = pass2.isValid(pass);
     System.out.println(pass + " is valid: " + valid);
     pass = pass2.generatePassword();
     System.out.println(pass);

   }
}
```

2. ISBN

General Problem: Write an `ISBN` class.
Refined Problem: Define necessary instance variables, a constructor with one integer parameter, and two methods.

Algorithm:

- Declare a private instance variable to hold the 9-digit ISBN number.
- Define a constructor with one integer parameter and assign the value of the parameter to the instance variable.
- Write a `calculateCheckDigit` method that returns a string containing the check digit.
  - For each of the 9 digits in the ISBN number, find the product of digit and the weighted value.
  - Maintain a sum of those products.
  - Determine the check digit by using the formula given.
- Write a `generateNumber` method that concatenates the instance variable with the result of the call to the `calculateCheckDigit` method.

Java Code:

```
public class ISBN
{
   private int isbnNumber;

   public ISBN(int number)
   {
      isbnNumber = number;
   }

   public String calculateCheckDigit()
   {
          int temp = isbnNumber;
          int digit, product;
          int multiple = 2;
          int sum = 0;
          for (int i = 1; i <= 9; i++)
          {
              digit = temp % 10;
              temp /= 10;
              product = digit * multiple;
              sum += product;
              multiple++;
          }
          int check = 11 - sum % 11;
          if (check == 10)
              return "X";
          return "" + check;
   }

   public String generateNumber()
   {
          return isbnNumber + "-" + calculateCheckDigit();
   }
}
```

Scoring Guidelines: ISBN

ISBN class **1 point**

+1 Complete, correct header for ISBN

state maintenance **1 point**

+1 Declares a private instance variable capable of maintaining the 9-digit ISBN number

ISBN Constructor **2 points**

+1 Correctly formed header
+1 Sets appropriate state variables based on parameter

calculateCheckDigit **3 points**

+1 Correctly formed header
+1 Finds the sum of the products of digits and weighted values
+1 Returns correct check digit

generateNumber **2 points**

+1 Correctly formed header
+1 Returns 10-digit ISBN number

Sample Driver:

There are many ways to write these methods. Maybe yours is a bit different from our sample solutions and you are not sure if it works. Here is a sample driver program. Running it will let you see if your code works, and will help you debug it if it does not.

Copy ISBNDriver into your IDE along with the complete `ISBN` class (including your solutions).

```
public class ISBN Driver
{
   public static void main(String [] args)
   {
        ISBN book1 = new ISBN(126045491);
        String check1 = book1.calculateCheckDigit();     //  "6" will be returned
        String isbnNumber1 = book1.generateNumber();     // "126045491-6" will be
                                                         // returned

        ISBN book2 = new ISBN(030640611);
        String check2 = book2.calculateCheckDigit();     //  "X" will be returned
        String isbnNumber2 = book2.generateNumber();     // "030640611-X" will be
                                                         //  returned
   }
}
```

3. Train

(a) **General Problem:** Write the `getTotalWeight` method to find the total weight of the engine and train.

Refined Problem: The method should loop through each element of the `ArrayList` and add it to a sum variable.

Algorithm:

- Initialize a sum variable to 0.
- Loop through each element of the `ArrayList`.
  - Add the value of the current element to the accumulated sum.
- Return the accumulated sum.

Java Code:

```
public double getTotalWeight()
{
    double weight = 0;
    for (Double car : trainCars)
       weight += car;
    return weight;
}
```

Common Errors:

- Not initializing the sum to 0.
- Not accessing each element of the `ArrayList`.
- Not returning the sum.

Java Code Alternate Solution:

```
public double getTotalWeight()
{
  double weight = 0;
  for (int i = 0; i < trainCars.size(); i++)
     weight += trainCars.get(i);
  return weight;
}
```

(b) **General Problem:** Write the `removeExcessTrainCars` method of the `Train` class that removes cars from the train until the train is light enough to be pulled by the engine.

Refined Problem: Calculate the initial weight of the train. If the weight exceeds the weight limit, TrainCars are removed from the end of the train until the weight is in the acceptable range. All train cars that are removed are returned in an `ArrayList` in the order they are removed from the train.

Algorithm:

- Create an `ArrayList` of Double to hold the removed cars.
- While the total weight is greater than the Engine's maximum weight,
  - Remove a train car from the end of the train
  - Add it to the `ArrayList` to be returned
- Return the `ArrayList` of removed cars (which may be empty).

Java Code:

```
public ArrayList<Double> removeExcessTrainCars()
{
   if (getTotalWeight() < getMaximumWeight())
     return null;
  ArrayList<Double> removed = new ArrayList<Double>();

   while (getTotalWeight() > getMaximumWeight())
   {
      Double removedCar = trainCars.remove(trainCars.size() - 1);
      removed.add(removedCar);
   }
   return removed;
}
```

Common Errors:

- Not creating a temporary `ArrayList` when needed.
- Remember that remove also returns the removed element. You do not need a `get` followed by a `remove`.
- Not returning the temporary `ArrayList`.

Scoring Guidelines: Train

Part (a) getTotalWeight 4 points

+1 Declare and initialize a weight variable
+1 Accesses the weight of every element of the `ArrayList`; no bounds errors, no missed elements
+1 Calculates the total weight of the train
+1 Returns the accumulated weight

Part (b) removeExcessTrainCars 5 points

+1 Declares and instantiates an `ArrayList` for the removed values
+3 Removes necessary cars from the train
  +1 Writes a loop with an appropriate terminating condition
  +1 Removes a car from the end of the train
  +1 Adds the car to the end of the declared `ArrayList`
+1 Returns the correctly generated `ArrayList`

Sample Driver:

There are many ways to write these methods. Maybe yours is a bit different from our sample solutions and you are not sure if it works. Here is a sample driver program. Running it will let you see if your code works, and will help you debug it if it does not.

Copy TrainDriver and the complete `Train` class into your IDE. Add your `getTotalWeight` and `removeExcessTrainCars` methods to the bottom of the `TrainDriver` class.

```
import java.util.ArrayList;

public class TrainDriver
{
     public static void main(String[] args)
     {
          ArrayList <Double> trainCars = new ArrayList <Double>();
          trainCars.add(200000.0);
          trainCars.add(100000.0);
          trainCars.add(140000.0);
          trainCars.add(50000.0);
```

```
            trainCars.add(100000.0);
            trainCars.add(60000.0);

            Train t = new Train(475000, trainCars);
            double weight = t.getTotalWeight();
            System.out.println("The total weight is " + weight);
            ArrayList <Double> removed = t.removeExcessTrainCars();
            System.out.println("The removed train cars are " + removed);
        }
}

import java.util.ArrayList;

public class Train
{
    private double maxWeight;
    private ArrayList <Double> trainCars;

    public Train (Double max, ArrayList<Double> tc)
    {
     maxWeight = max;
     trainCars = tc;
    }

    public double getMaximumWeight()
    {
     return maxWeight;
    }


/* insert your code here */

}
```

4. Pixels

(a) **General Problem:** Write the `generatePixelArray` method to convert three 2-D int arrays into a 2-D array of `Pixel` objects.

Refined Problem: Use a nested `for` loop to instantiate a `Pixel` object for each element of the `Pixel` object array. Use the corresponding values in the red, green, and blue arrays as parameters to the Pixel constructor. When the loops are complete, return the completed array.

Algorithm:

- Create a 2-D array of type Pixel to hold the new `Pixel` objects. The dimensions of this array are the same as any of the color arrays because there is a precondition saying all the arrays must be the same size.
- Write a `for` loop that goes through all the rows of the new array.
  - Nested in that `for` loop, write a `for` loop that goes through all the columns of the new array. (These can be switched. Row-major order is more common, but column-major order will also work here.)
    - Instantiate a new `Pixel` object, passing the values in the red, green, and blue arrays at the position given by the two loop counters, and assign it to the element of the Pixel array given by the two loop counters.
- When the loops are complete and every element has been processed, return the completed Pixel array.

Java Code:

```
public static Pixel[][] generatePixelArray(int[][] reds, int[][] greens, int[][] blues)
{
   Pixel[][] pix = new Pixel[reds.length][reds[0].length];
   for (int r = 0; r < pix.length; r++)
      for (int c = 0; c < pix[r].length; c++)
         pix[r][c] = new Pixel(reds[r][c], greens[r][c], blues[r][c]);
   return pix;
}
```

(b) **General Problem:** Write a `flipImage` method that takes a 2-D array of `Pixel` objects and flips it into a mirror image, either vertically or horizontally.

Refined Problem: Create a new array to hold the altered image. Determine whether to flip the image horizontally or vertically. Write a nested `for` loop to move all the Pixels to their mirror-image location in the new array, either horizontally or vertically. When the loops are complete, return the new array.

Algorithm:

- Instantiate a 2-D Pixel array that has the same dimensions as the array passed as a parameter. This array will hold the altered image.
- Create an `if-else` statement with one clause for a horizontal flip and one for a vertical flip.
- If the flip is horizontal, write a nested `for` loop that goes through the array in row-major order.
  - For each iteration of the loop, an entire row is moved into its new "flipped" place in the altered array.
- Otherwise, the flip is vertical. Write a nested `for` loop that goes through the array in column-major order.
  - For each iteration of the loop, an entire column is moved into its new "flipped" place in the altered array.
- Return the altered array.

Java Code:

```
public static Pixel[][] flipImage(Pixel[][] image, boolean horiz)
{
   Pixel[][] flipped = new Pixel[image.length][image[0].length];
   if (horiz)
      for (int r = 0; r < image.length; r++)
         for (int c = 0; c < image[0].length; c++)
            flipped[r][c] = image[image.length - 1 - r][c];
   else
       for (int c = 0; c < image[0].length; c++)
          for (int r = 0; r <image.length; r++)
             flipped[r][c] = image[r][image[0].length - 1 - c];
   return flipped;
}
```

Common Errors:

- Watch off-by-one errors. Always think: do I want length or length − 1? Using variables, like in the alternate solution, can help you be consistent.
- Be sure you understand the difference between row-major and column-major order.
- Check your answer with both even and odd numbers of rows and columns.
- Create a new array to hold the "flipped" version. Do not overwrite the array that is passed in. This is called *destruction of persistent data* and incurs a penalty.

Java Code Alternate Solution:

- This solution only loops halfway through the array. It flips a pair of lines on each iteration.

This solution also uses a few extra variables to keep things easier to read. This is not necessary but it reduces the amount of typing and the chance of an off-by-one error with length, as opposed to length − 1.

```
public static Pixel[][] flipImage(Pixel[][] image, boolean horiz)
{
   Pixel[][] flipped = new Pixel[image.length][image[0].length];
   int maxR = image.length - 1;
   int maxC = image[0].length - 1;
   if (horiz)
   {
      for (int r = 0; r <= maxR / 2; r++)
      {
         for (int c = 0; c <= maxC; c++)
         {
            flipped[r][c] = image[maxR - r][c];
            flipped[maxR - r][c] = image[r][c];
         }
      }
   }
   else
   {
      for (int c = 0; c <= maxC / 2; c++)
      {
         for (int r = 0; r <= maxR; r++)
         {
            flipped[r][c] = image[r][maxC - c];
            flipped[r][maxC - c] = image[r][c];
         }
      }
   }
   return flipped;
}
```

Scoring Guidelines: Pixels

Part (a) **generatePixelArray** **3 points**

+1 Instantiates a 2-D array of `Pixel` objects with the correct dimensions

+1 Instantiates a new `Pixel` object with the corresponding elements of the red, green, and blue arrays for every element of the array; no bounds errors, no missed elements

+1 Returns the properly constructed and filled array

Part (b) **flipImage** **6 points**

+1 Correctly determines whether the image should be flipped vertically or horizontally

+2 Flips the array horizontally

- **+1** Correctly repositions at least one element
- **+1** Correctly repositions all elements

+2 Flips the array vertically

- **+1** Correctly repositions at least one element
- **+1** Correctly repositions all elements

+1 Returns the correctly generated flipped array

Sample Driver:

There are many ways to write these methods. Maybe yours is a bit different from our sample solutions and you are not sure if it works. Here is a sample driver program. Running it will let you see if your code works, and will help you debug it if it does not.

Copy PixelDriver and the complete `Pixel` class into your IDE. Add your `generatePixelArray` and `flipImage` methods to the bottom of the `PixelDriver` class.

```
public class PixelDriver {

   public static void main(String[] args) {
      int[][] red = { { 0, 0, 0, 0 }, { 1, 1, 1, 1 }, { 2, 2, 2, 2 },
                      { 3, 3, 3, 3 } };
      int[][] green = { { 0, 1, 2, 3 }, { 0, 1, 2, 3 }, { 0, 1, 2, 3 },
                        { 0, 1, 2, 3 } };
      int[][] blue = { { 100, 100, 100, 100 }, { 100, 100, 100, 100 },
                       { 100, 100, 100, 100 }, { 100, 100, 100, 100 } };
      System.out.println("generatePixelArray test");
      System.out.println("Expecting:");
      for (int r = 0; r < red.length; r++) {
         for (int c = 0; c < red[r].length; c++)
            System.out.print("(" + red[r][c] + ", " + green[r][c] +
                  ", " + blue[r][c] + ") ");
         System.out.println();
      }
      Pixel[][] pix = generatePixelArray(red, green, blue);
      System.out.println("\nYour Answer:");
      printIt(pix);

      final boolean HORIZ = true;
      final boolean VERT = !HORIZ;
      System.out.println("\nflipImage - Horizontal test");
      System.out.println("Expecting:");
      for (int r = red.length-1; r >=0; r--) {
         for (int c = 0; c < red[r].length; c++)
            System.out.print("(" + red[r][c] + ", " + green[r][c] +
                  ", " + blue[r][c] + ") ");
         System.out.println();
      }
      System.out.println("\nYour Answer:");
      Pixel[][] pix2 = flipImage(pix, HORIZ);
      printIt(pix2);

      System.out.println("\nflipImage - Vertical test");
      System.out.println("Expecting:");
      for (int r = 0; r < red.length; r++) {
         for (int c = red[r].length - 1; c >= 0; c--)
            System.out.print("(" + red[r][c] + ", " + green[r][c] +
                  ", " + blue[r][c] + ") ");
         System.out.println();
      }
      System.out.println("\nYour Answer:");
      Pixel[][] pix3 = flipImage(pix, VERT);
      printIt(pix3);
   }

   public static void printIt(Pixel[][] pix) {
      for (int r = 0; r < pix.length; r++) {
         for (int c = 0; c < pix[r].length; c++) {
            System.out.print(pix[r][c] + " ");
         }
         System.out.println();
      }
   }
   // Enter your generatePixelArray and flipImage methods here
}
```