Practice Test 4: Answers and Explanations

PRACTICE TEST 4 ANSWER KEY

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

ANSWER EXPLANATIONS

Section I: Multiple-Choice Questions

1. **A** You start with the division in the second pair of parenthesis to get a result of 2.5. This is not the final answer, however, so (B) is not correct. When this number is cast to an integer by the `(int)`, it removes the decimal to result in a final value of 2. Casting to an integer never rounds so (C) is incorrect.

2. **E** The += operator is short a shortcut to adding to a numeric variable. Answer (A) is incorrect because it changes the value of num to 1 instead of adding 1. Answer (B) successfully adds 1 to the value of num, but it does not store the new result in num so the variable remains unchanged. (C) would be correct without the 1 at the end, but as written it would result in a syntax error. (D) is incorrect because in an assignment statement the variable being assigned a value need to appear on the left.

3. **C** The `Math.random( )` method returns a number between 0 and 1 (exclusive). Since 0 is the smallest number the method can return, we need to add 5 to reach the minimum number we want. That eliminates options (A), (B), and (D). Since 1 is not included in the `Math.random( )` method we need to multiply the result of `Math.random( )` times the maximum number—the minimum number plus 1. That would be 25 – 5 + 1, which equals 21. This eliminates option (E) since it multiples `Math.random( )` by 25 instead of 21.

4. **C** The `compareTo` method returns 0 if two strings are equivalent. Since "Chicago" and "London" are not equivalent answer (A) is incorrect. The `compareTo` method returns the alphabetical difference between the Strings, starting with the first letter. Since "C" is 9 letters before "L" in the alphabet, the compareTo method will return –9. This rules out answers (B) and (D).

5. **D** For Strings, the `compareTo` method will return a number less than 0 if the first String comes before the second String alphabetically. That eliminates answers (B) and (E). While we don't know how the input from the user is implemented, if it is possible one of the cities remains `null`, there could be a `NullPointerException`.

6. **A** While you could use a truth table to solve this problem, it will go much quicker if you use DeMorgan's Laws to simplify the expression. The expression !(x < 5 && y > 10) expands to (x >= 5 || y <= 10) and !(x < 5 || y > 10). You can then simplify that expression to x >= 5 && y <= 10.

7. **B** Since result is a boolean variable it can only stored true or false. That rules out answers (A) and (D). Answer (E) might seem correct because the variable num is equal to 0 and you divide by num at the end of the expression, but there is no error because of short circuiting. Since the first part of the expression (`num == 0`) is true and the expression is combined using an or statement (`||`), the compiler does not need to check the second half of the expression since it has already determined that the expression is true.

8. **B** When you have an if/else if/else series only one of the statements in the series will run. That eliminates options (D) and (E). When two objects have the same value, the == operator only returns true if they are stored in the same location in memory. Since city and president are have the same value, but were never set equal to each other they are not stored in the same memory location. This eliminates option (A). Using the equals method on objects checks if they have the same value so the second statement will run.

9. **B** When you have an if/else if/else series only one of the statements in the series will run. That means num will either go up by 1 (the if and else if statements) or be set equal to 10. This eliminates answers (A), (C), and (D). The if statement condition (num < 10) is true since num is equal to 0 at the start. That means that num will be increased by 1.

10. **C** This question is almost equivalent to question 8. The only difference is the else if from question 8 is an if statement in question 9. We already know that the first if statement runs so num will not remain 0, eliminating answer (A). Since num is 1 when the second if statement (`num == 1`) is evaluated, num is again increased by 1 to reach 2. The else statement does not run because exactly one statement runs in an if/else pair.

11. **E** We know that the first if statement runs since it is equivalent to the previous two questions, eliminating answer (A). The else if statement does not run because exactly one statement in an if/else if pair will run. Since num is equal to 1 when we get to the second if statement (`num == 2`), it will run the else statement resulting in num equaling 10 when the code is complete.

12. **D** The condition of the while loop is `num < 10`. That means the loop will run as long as num is less than 10. Since num starts at 0, the loop will run at least once. This eliminates answer (A). The body of the loop increases num by 1 (`num++`) and since the loop runs 10 times (0…9) num will equal 10 when the loop ends.

13. **C** The condition of the while loop is `num < 10`. That means the loop will run as long as num is less than 10. Since there is an if/else pair in the loop and num increases in the if and the else, num will eventually become 10 or greater. This eliminates answer (E). The modulus operator (%) returns the remainder of division. The `num % 2 == 0` condition is checking to see if num is divisible by 2. The variable num is divisible by 2 when it is 0 so num goes up to 1. When num is 1 it is not divisible by 2, so it is instead multiplied by 2 to go up to 2. 2 is divisible by 2 so num becomes 3. 3 is not divisible by 2 so num becomes 6. 6 is divisible by 2 so num becomes 7. 7 is not divisible by 2 so num becomes 14. Since 14 is not less than 10, the loop ends.

14. **B** The condition must be a condition that will be true when num is 1, otherwise the loop won't run. This eliminates answers (A), (D), and (E) since 1 is not equal to 4, greater than 4, or greater than or equal to 4. If the condition is true when num is equal to 4, that will cause it to run one extra time and num will end up equal to 8 so this eliminates answer (C).

15. **D** Since `i` and `num` are both equal to 0 when the for loop starts, answer (E) would cause the loop to not run at all. Since we want the loop to stop based on the value of num we should be using num in the condition, which eliminates answers (A) and (B). If the loop continues to run when num equals 10, it will go beyond 10 so we don't want the condition that checks if num is <= 10.

16. **C** The loop variable i increases by 2 each time the loop runs (`i += 2`) so i will equal 0, 2, 4, 6, 8, and 10 while the loop is running. Once i becomes 12, the loop ends. That means the loop body ran 6 times.

17. **C** The loop will start at 1000 (`int i = 1000`) and count down by 1 (`i--`) to 0, but not include the 0. This means it will run 1000 times.

18. **E** The outer for loop will run 5 times (i = 0…i = 4). The inner for loop will run 2 times (k = 0…k = 1). This will cause the inner for loop body to run a total of 10 times (5 x 2), causing num to go up to a value of 10.

19. **C** The loop will add a single character from `firstString` to `secondString` each time the loop runs. Using the `substring` method with i and i+1 as the arguments is what causes it to add a character at a time. Since the loop runs 5 times (0…4), it will add the first 5 characters from `firstString` "missi" to `secondString`.

20. **C** This question uses an enhanced for loop, also known as a for-each loop. An enhanced for loop looks at every element in a collection and stores it in a temporary variable. The temporary variable in this case is called `city`. Since an enhanced for loop looks at every element and there are no errors in the code, we can eliminate answers (A) and (E). Since a `println` statement is used the loop will print each element on a separate line. This eliminates answer (B). Finally, an enhanced for loop prints the elements from the collection in the same order so Chicago will be printed first and Manila will be printed last.

21. **B** When you are trying to set the value of a variable that already exists, you shouldn't precede the variable name with the variable type. This eliminates answers (C), (D), and (E). The variable you are trying to change should come before the = so `firstName`, `lastName`, and `number` should be to the left of the =.

22. **D** You should never return something from a void method so that eliminates answers (A) and (B). As stated in the explanation for question 19, you shouldn't precede a variable name with the variable type if you are trying to change the value of an already-existing variable. That eliminates answer (E). Since `position` is the variable we are trying to change, it should come to the left of the =.

23. **D** You should not change the names of any variables you are given. Doing so could cause problems elsewhere in the program. This eliminates answers (A) and (B). The `this` keyword is used to access class-level variables if you have local variables with the same name.

24. **E** When you initialize a new array you need to initialize it with the size and variable type. That eliminates answers (C) and (D). You also need to put [ ] after the variable type when you declare the array, which eliminates answers (A) and (B).

25. **B** The `compareTo` method returns a number less than 0 if the first String comes before the second String. That rules out answers (A), (C), and (E). The loop continues to run over the entire length of the array, so it will eventually find the String that comes first alphabetically.

26. **D** The variable `x` is only going up by 1 (`x++`) each time the if statement is true, so the code can't be getting a total of the numbers. This eliminates answers (B) and (C). The if statement uses modulus (%) to determine if the current number in the array is even since and even number has no remainder when it is divided by 2.

27. **A** Since array indices start at 0, the 10th element of an array would be index 9. That eliminates answers (B), (D), and (E). Answer (C) is not correct either since it is attempting to set the array equal to 9 instead of accessing index 9 using square brackets [ ].

28. **C** You can use binary search on any data structure, so that rules out answers (A), (B), and (E). Binary search can also be used on any primitive type (such as integers, doubles, and booleans) or any Comparable type (such as Strings), which rules out answer (D).

29. **E** The `substring` method will look at the first letter of each element of `myList` (which are stored in the `temp` variable) and see if it is equal to "s". The variable `x` will increment by 1 (`x++`) for each element that starts with "s."

30. **B** Insertion sort works by finding the minimum element and swapping it with the first element. It then moves to the second element and find the second-smallest element. Bubble sort and quick sort are not tested on the AP Computer Science A exam.

31. **D** When initializing a 2D array, you declare the number of rows in the first pair of square brackets and the number of columns in the second pair. That eliminates answers (A), (C), and (E) since they declare 10 rows instead of 5 rows. Answer (B) is incorrect because it does not have two pairs of square brackets after the variable type to the left of the equals sign.

32. **D** The nested for-each loops will iterate through the entire 2D array. Since the code is adding `temp` to the variable `x` (`x += temp`), that will result in a sum of all of the values in the 2D array being stored in `x`.

33. **C** Getting the length of a 2D array gives you the number of rows. This eliminates answers (A) and (B). You need to access an element in a row of the 2D array and get its length to determine the number of columns. You can do this by putting `myArray[0].length` or `myArray[row].length`.

34. **B** When you are creating a class that is a subclass of another class you need to use the `extends` keyword. This eliminates answers (A), (C), and (E). You always put the name of the current class directly after the keyword `class`, which eliminates answer (D).

35. **A** When declaring a constructor, there is no return type like a normal method. Removing the void keyword would fix the error. Line 2 correctly sends the name to the superclass constructor and line 3 correctly initializes the class-level variable salary.

36. **E** Abstraction is hiding the details of how something works and just focusing on what it does. For example, you may use the `substring` method quite a bit, but you don't need to know how it actually calculates the substring you ask for in order to use it. Inheritance allows code reusability between classes. Encapsulation hides unnecessary details through the use of private variables and methods.

37. **D** The `Object` class is the superclass for all other classes in Java.

38. **C** The base case is the value that will be returned that stops making recursive calls. You can see that line 6 makes another recursive call so that rules out answer (D). The actual base case is when the value of the base case is set, which is why answers (B) and (E) are not correct.

39. **E** Here is what the recursive calls look like:

```
sum(5) = 5 + sum(4)

sum(4) = 4 + sum(3)

sum(3) = 3 + sum(2)

sum(2) = 2 + sum(1)

sum(1) = 1
```

Going back through the stack of recursive calls:

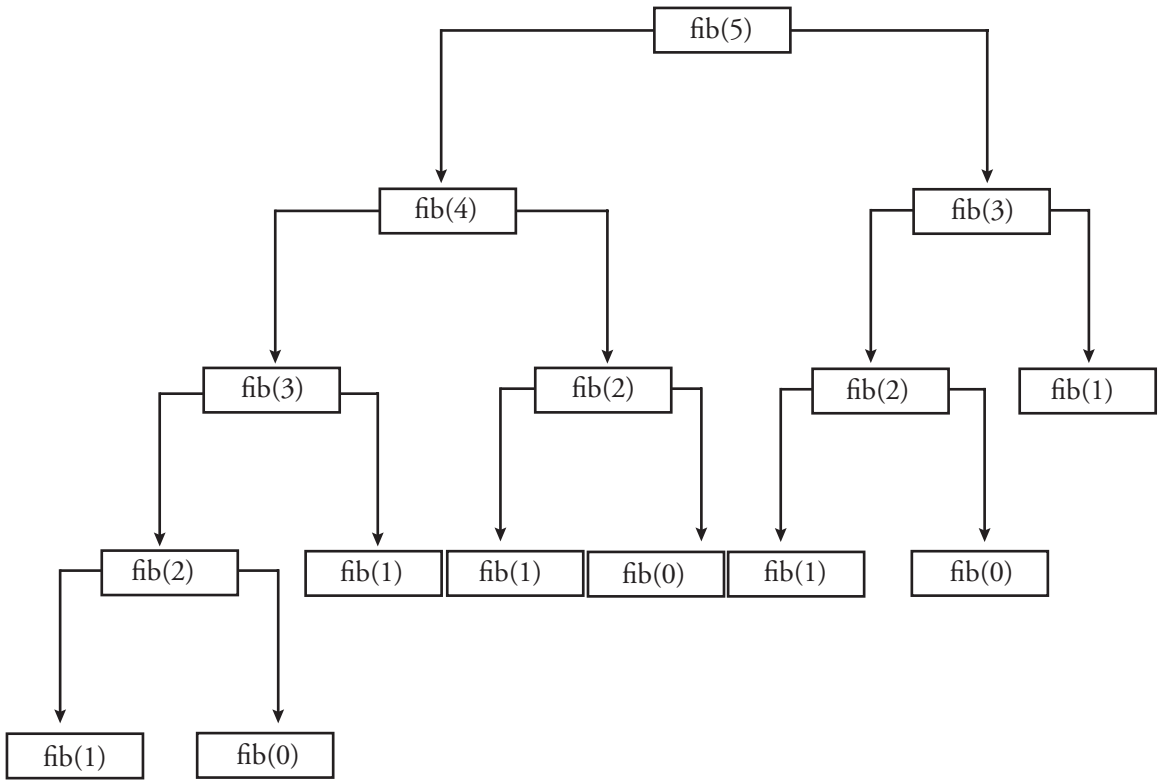
```
sum(2) = 2 + 1 = 3

sum(3) = 3 + 3 = 6

sum(4) = 4 + 6 = 10

sum(5) = 5 + 10 = 15
```

40. **E** This problem can be solved by creating a tree of the recursive calls. Based on the tree below, you can see there are 15 recursive calls.



Section II: Free Response Questions

1.

```
public class DayCare
{
  private ArrayList<Employee> employees;
  private ArrayList<Child> children;
  private int maxRatio;

  public DayCare(int maxRatio)
  {
     employees = new ArrayList<Employee>();
     children = new ArrayList<Child>();
     this.maxRatio = maxRatio;
  }
```

(a)

```
  public boolean findEmployeeForChild(Child c)
  {
  for (Employee e: employees)
  {
     if (e.childrenAssigned() < maxRatio && e.canTeach(c.
getAge()))
     {
         e.assignChild(c);
         return true;
     }
  }

  return false;
}
```

This can also been done with a typical for loop or a while loop and the .get method. A loop is necessary to look at each `Employee` in the ArrayList. You must then make sure the chosen `Employee` doesn't already have the maximum number of children allowed. You must also send the age of the *Child* using the `getAge` accessor to the `canTeach` method to see if the chosen `Employee` is eligible to teach the given `Child`.

If an `Employee` is found for the given `Child` you need to assign the `Child` to the `Employee` using the `assignChild` method and return true.

You should not return false inside the loop since it is possible a different `Employee` is eligible to teach the given `Child`.

Rubric:

(+1) For loop correctly iterates through all elements of the employees list.
(+1) If statement correctly determines if children assigned to the current employee are less than the maximum allowed.
(+1) If statement correctly determines if current employee can teach the given child based on their age.
(+1) The child is assigned to the employee if they meet both conditions.
(+1) True is returned if the child is assigned to an employee, false is returned otherwise.

(b)

```
public boolean runDayCare()
{
  for (Child c: children)
  {
     if (findEmployeeForChild(c) == false)
     return false;
  }

  return true;
}
```

This can also been done with a typical for loop or a while loop and the .get method. A loop is needed to look at each `Child` in the ArrayList. You must then call the `findEmployeeForChild` method from Part A to see if there is an `Employee` eligible to teach the current `Child`. If not, you need to return false. You shouldn't return true inside the loop since it is possible there is a later `Child` who can't be taught by any of the `Employees` in the ArrayList.

Rubric:

(+1) For loop correctly iterates through all elements of the children list.
(+1) The findEmployeeForChild method is called correctly.

(c)

```
  public boolean addChild(Child c)
  {
     if (findEmployeeForChild(c) == true)
     {
         children.add(c);
         return true;
     }

     return false;
  }
}
```

You must then call the `findEmployeeForChild` method from Part A to see if there is an `Employee` eligible to teach the given `Child`. If there is, you add the `Child` to the ArrayList using the `add` method and return true. If there isn't, you return false.

Rubric:

(+1) The findEmployeeForChild method is called correctly.
(+1) Children are added to the children list correctly if an employee is found.

2.

```
public class Player extends Person
{
  private String position;

  public Player(String name, int age, String pos)
  {
     super(name, age);
     position = pos;
  }

  public void changePosition(String p)
  {
     position = p;
  }
}
```

The class header must look exactly the same as the header above. The public class Player part would be necessary for any class you are writing called `Player`. The extends Person part is necessary because a `Player` is a `Person`.

The position variable **must** be declared as private and it must be a String.

The constructor (public Player), must take three parameters in the order shown above since the example shows the name, age, and position in that order. They can be called whatever you want, however. The name and age variables must be sent to the `Person` class using the super( ) call and they must be in the given order. The position variable should be set after the super( ) call.

The `changePosition` method should be void and should take a String parameter. The only thing it needs to do is set the class-level position variable.

Rubric:

(+1) `public class Player`
(+1) `extends Person`
(+1) A String variable is declared as private.
(+1) The constructor header is correct (`public Player`)
(+1) The constructor takes a String parameter, an integer parameter, and a String parameter in that order.
(+1) The constructor calls uses super to initialize the name and age.
(+1) The constructor initializes the class-level String variable.
(+1) The header for `changePosition` is correct.
(+1) The `changePosition` method correctly modifies the class-level String variable.

3.

(a)

```
public static String getFirstName(String name) {
  int space = name.indexOf(" ");
  String first = name.substring(0, space);
  return first;
}
```

You need to use the `indexOf` method to find the location of the space. Once you know where the space is located, you can use the `substring` method to extract from the beginning of the name (index 0) up to the space. Since the second parameter of the `substring` method is excluded, the space will not be included when the first name is returned.

Rubric

(+1) The `indexOf` method is used correctly to find the first space.
(+1) The `substring` method is used correctly to extract the first name.
(+1) The first name is returned correctly.

(b)

```
public static String getLastName(String name) {
  int space = name.indexOf(" ");
  String last = name.substring(space + 1);
  return last;
}
```

The solution to part b is similar to part a. You still need to find the location of the space, but the substring starts at the location of the space plus 1, which will be the first letter of the last name. You could add a second parameter of `name.length()`, but it isn't required.

Rubric

(+1) The `indexOf` method is used correctly to find the first space.
(+1) The `substring` method is used correctly to extract the last name.
(+1) The last name is returned correctly.

(c)

```
public static int countVowels(String name) {
  int count = 0;
  for (int i = 0; i < name.length(); i++)
  {
      String letter = name.substring(i, i+1);
      if (letter.equals("a") || letter.equals("i") || letter.equals("e") ||
letter.equals("o") || letter.equals("u"))
          count++;
  }
  return count;
}
```

You need to create a loop to go through the entire name. You could use a for-each loop to extract characters, but characters are not part of the AP subset. If you use them, make sure you use them correctly. With a traditional for loop, you need to extract the letter using the `substring` method and see if it equals one of the vowels. A count variable is increased by 1 each time a vowel is found.

Rubric

(+1) A loop is used to look at each letter in the name.
(+1) An if statement is used to determine if the current letter is a vowel.
(+1) The correct vowel count is returned.

4.

(a)

```
public int openSpaces()
  {
     int taken = 0;
     for (int r = 0; r < lot.length; r++)
     {
        for (int c = 0; c < lot[r].length; c++)
        {
          if (lot[r][c] != null)
               taken++;
        }
     }
     return (lot.length * lot[0].length) - taken;
  }
```

You need to use nested for loops to iterate through the lot 2D array. You can also use `lot[0].length` in the second for loop that iterates through the columns instead of `lot[r].length`. If you find a spot that is not equal to `null` (meaning there is already a car parked there), then you should increase a counter variable by 1. That variable should be subtracted from the size of the 2D array to get the final result.

Rubric

(+1) A variable is declared to keep track of the taken parking spaces.
(+1) Nested for loops are used correctly to iterate through the lot 2D array.
(+1) An if statement checks if the current spot in the array is not `null`.
(+1) The correct number of open spaces is returned.

(b)

```
public boolean parkCar(Car newCar)
{
  if (openSpaces() > 0)
  {
      for (int r = 0; r < lot.length; r++)
      {
          for (int c = 0; c < lot[r].length; c++)
          {
              if (lot[r][c] == null)
              {
                   lot[r][c] = newCar;
                   return true;
              }
          }
      }
  }

  return false;
}
```

You need to use nested for loops to iterate through the lot 2D array. You can also use `lot[0].length` in the second for loop that iterates through the columns instead of `lot[r].length`. If you find a spot that is `null`, you need to set that spot to `newCar` and return `true`, in that order. There should be a `return false` at the end of the method, or in an else statement. On the real AP exam, if a question tells use to use a method (such as `openSpaces`) and you don't use it you can lose points.

Rubric

(+1) The `openSpaces` method is called correctly to determine if there are spaces available.
(+1) Nested for loops are used correctly to iterate through the lot 2D array.
(+1) An if statement checks if the current spot in the array is `null`.
(+1) The `newCar` is assigned correctly to a `null` element in the 2D array.
(+1) The method returns `true` if a spot was found, and `false`