**AP Computer Science - Practice Test # 1 Answers and Explanations**

|  |  |  |  |
| --- | --- | --- | --- |
| **Question** | **Answer** | **Correct** | **Explanation** |
| 1 | D | 6/17 | Constructors are not inherited. If a subclass has no constructor, the default constructor for the superclass is generated. if the superclass does not have a default constructor, a compile-time error will occur. |
| 2 | B | 12/17 | The programmer is using an object-oriented approach to writing the program and plans to test the simplest classes first. This is bottom-up development |
| 3 | A | 15/17 | In the constructor, the private instance variables mySuit and myValue must be initialized to the appropriate parameter values. Only A does this. |
| 4 | B | 6/17 | II works because it invokes the toString method of the Card class. Implementation I fails because there is not default toString method for arrays. Implementation III will cause a ClassCastException since you cannot cast a Card to a String. |
| 5 | E | 4/17 | Since values in the array cannot be greater than the constant Integer.MAX\_VALUE, the test in the while loop will be true at least once and will lead to the smallest element being stored in min. Similarly, if we assign min the value of the first element in the array (min = arr[0]; ) we will have a legitimate value to begin our search for the min. |
| 6 | A | 5/17 | The array will not be changed by the increment method. Nothing is returned and a single integer value sent to a void method will not be changed outside the method. If you send the entire array to a method, its contents can be altered, except when using a for..each loop. |
| 7 | A | 5/17 | Same as 6. We are sending integer values into the method, not the array itself, therefore any changes made inside will not affect the array outside the method. |
| 8 | B | 6/17 | The maximum number will be achieved if /\* test \*/ is true in each pass through the loop. So the question boils down to: How many times is the loop executed? Try one odd and one even value of n. if n=7, I =0, 2, 4, 6 (answer is 4) if n=8, I = 0, 2, 4, 6 (Answer =4 also) B is the only answer that works for both n=7 and n=8. |
| 9 | C | 4/17 | Golden rule of programming. Don't start programming until every aspect of the specification is crystal clear. A programmer should never make unilateral decisions about ambiguities in a specification. |
| 10 | B | 7/17 | When x <=y a recursive call is made to whatIsIt(x-1, y). If x decreases at every recursive call, there is no way to reach a successful "exit" from the recursive call. Thus the method never terminates and eventually exhausts all available memory. |
| 11 | A | 17/17 | The expression !(max != a[i]) is equivalent to max==a[i], so the given expression is equivalent to a[i]=max || max == a[i], which is equivalent to a[i]==max |
| 12 | C | 6/17 | A base-b number can be represented with b characters. Thus base-2 (binary) uses 0,1. Base 10 uses 0-9. Hex (base 16) uses 0-9, A-F. The largest 2 digit hex number is FF, which converted to decimal is 255. (15 x 161 + 15 x 160) |
| 13 | B | 9/17 | The index range for ArrayList is 0<=index<= size()-1. Thus for methods get, remove and set, the last in-bounds index is size()-1. the one exception is the add method-to add an element to the end of the list takes an index parameter list.size() |
| 14 | E | 4/17 | Subclasses of Quadrilateral may also be abstract, in which case they will inherit perimeter and/or area as abstract methods. |
| 15 | C | 12/17 | Segment I starts correctly but fails to initialize the additional private variables of the Rectangle class. Segment II is wrong because by using super with topLeft and botRight, it implies that theses values are used in the Quadrilateral superclass. This is false - there isn't even a constructor with three arguments in the superclass. |
| 16 | A | 10/17 | During execution the appropriate area method for each quad in quadList will be determined (polymorphism) |
| 17 | E | 13/17 | Classic Bubblesort. Store first item or value in temp. Set 2nd item = to 1st item. Set 2nd item = to temp. Switches these two items. |
| 18 | A | 6/17 | Subtracting .5 from a negative number and then truncating it (by casting as int) produces the same result as rounding. |
| 19 | E | 5/17 | The method call whatsIt(347) calls whatsIt(34), which calls whatsIt(3), which satisfies the break condition of the while loop (while n >10). At this point we work out way back out, outputting n%10. 3%10 = 3, 34 %10 = 4, 347%10 = 7, so 347 is outputted. |
| 20 | B | 3/17 | Recall that insertion sort takes each element in turn and (a) finds its insertion point and (b) moves elements to insert that element at its correct spot. Thus if the array is in reverse sorted order, the insertion point will always be at the front of the array, leading to the maximum number of comparisons and data moves- very inefficient. Therefore choices A, C and E are false. Selection sort finds the smallest element in the array and swaps it with a[0] and then finds the next smallest element in the rest of the array and swaps it with a[1], and so on. Thus the same number of comparisons and moves will occur, irrespective of the original arrangement of elements in the array. so choice B is true and choice D is false. |
| 21 | E | 0/17 | Method call I fails because ClassOne does not have access to the methods of its subclass. Method call II fails because c2 needs to be cast to ClassTwo to be able to access methodTwo. Thus the following would be okay: ((ClassTwo) c2).methodTwo(); Method call III works because ClassTwo inherits methodOne from its superclass, ClassOne. |
| 22 | D | 11/17 | Notice that in the original code, if n=1, k is incremented by 1 and if n is 4, k is incremented by 4. This is equivalent to saying"if nis 1 or 4, k is incremented by n". |
| 23 | E | 3/17 | Segment I will throw a NullPointerException when s.equals… is invoked because s is a null reference. Segment III looks suspect, but when the start index of a substring method equals s.length(), the value return is the empry string. If, however, startindex > s.length(), a StringIndecOutOfBoundsException is thrown. |
| 24 | A | 12/17 | Since results of calculations with floating-point numbers are not always represented exactly (round-off error), direct tests for equality are not reliable. |
| 25 | C | 4/17 | If arr has elements 2, 3, 5, the values of value are 2, 2\*10+3 (23), 23\*10+5=235 |
| 26 | D | 8/17 | The point of the binary search algorithm is that the interval containing key is repeatedly narrowed down by splitting it in half. For each iteration of the while loop, if key is in the list, arr[first] <= key <= arr[last]. Note that (i) the endpoints of the interval must be included, and (ii) key is not necessarily in the list. |
| 27 | B | 3/17 | 1st iteration - first=0, last=13, mid = 6, a[mid]=50  2nd iteration - first = 7, last=13, mid = 10, a[mid]=220  3rd iteration - first = 7 last = 9, mid = 8, a[mid]=101  4th iteration - first = 9, last=9, mid = 9, a[mid] = 205 |
| 28 | A | 5/17 | The data structure is an array, not an ArrayList, so you cannot use the add method for inserting elements into the list. This eliminates choices B and D. The expression to return a random integer from 0 to k-1 inclusive is: (int) Math.random()\*k)  Thus, to get integers from 0 to 100 requires k to be 101, which eliminates choice C. Choice E fails because it gets integers from 1 to 100. |
| 29 | A | 6/17 | Recall that s.substring(x, y) will return a substring of s, starting at position x and ending at y-1. s.substring(x) will return a substring starting at position x to the end of the string. Thus if s="koolaid", s.substring(2, 5) will return "ola". s.substring(3) will return "laid". |
| 30 | A | 12/17 | changeMatrix examines each element and changes it to its absolute value when its row number matches its column number. Given the matrix in the problem, mat[0][0] and mat[1][1] will be changed to its absolute value. The rest is left unchanged. |
| 31 | C | 4/17 | Composition is the "has-a" relationship. A PlayerGroup has-a Players (several of them, in fact). Inheritance is the is-a relationship, which doesn’t apply here. Neither A, B or E applies to this situation. An interface is a single class composed of only abstract methods. Encapsulation is the bundling together of data fields and operations into a single unit (a class). |
| 32 | E | 9/17 | All of these are reasonable. They all represent 20 bingo numbers and a convenient way to cross them out. |
| 33 | D | 5/17 | A NullPointerException is thrown whenever an attempt is made to invoke a method with an object that hasn't been created with new. Choice A doesn't make sense: To test the Caller constructor require a statement of the form: Caller c = new Caller();  Choice B is wrong: A missing return statement would trigger a compilation error; C doesn't make sense and choice E is just bizarre. |
| 34 | D | 6/17 | Location.RIGHT is 90. A bug's default behavior only turns 45 degrees. |
| 35 | E | 10/17 | Location.NORTHWEST=315 and Location.HALF\_LEFT = -45, thus 315 + -45 = 270 |
| 36 | B | 7/17 | The canMove method in the Bug class is the only ocde that needs to be changed. here is the change:  return (neighbor==null) || (neighbor instanceofFlower) || (neighbor instanceof Rock);  Note that the BoxBug class does not need to be changed. in the act method, the inherited canMove method will now return true if the location in front of the BoxBug is empty or contains flower or rock. The Actor class also does not need to be changed. The moveTo method specifies that if there is another Actor at the new location, that actor will be removed. |
| 37 | D | 1/17 | The ChameleonCritter randomly picks either the rock of the bug, and changes its color to match that of the selected actor. Elimnate Choice A- the ChameleonCrittter cannot remain blue. next, the ChameleonCritter moves to an empty neighboring location, changes its direction to match the direction in which it moved. Eliminate choice B, since (1,1) is not an empty neighboring location. Choices C and E both change color correctly and move to a valid location. The ChameleonCritter, however, ends up facing the wrong direction. The direction from (1,1) to (1,2) is east, not northeast. The direction from (1,1) to (2,0) is southwest, not south. |
| 38 | E | 8/17 | The specification for moveTo states that :  \*The actor will be moved to a new location  \*if there's already an actor in this location, it will be removed.  The key here is that two actors may not occupy the same location simultaneously. Therefore, you cannot use moveTo to place two Monkey objects in the same location at a zoo. |
| 39 | A | 7/17 | The critter will eat the bugs in (0,0) and (1,2) and the flower in (2,0, leaving those locations empty and available. Also available are the other empty locations: (2,1), (2,2) and (0,2). The ChameleonCritter doesn't eat its neighbors, so the only available locations are the ones that were empty to begin with: (2,1), (2,2) and (0,2)q |
| 40 | B | 11/17 | The act method of the Critter class must be changed so that the age variable is incrased by 1 each time act is called. Statement I is wrong because not all actors age. Statement III is wrong because a ChameleonCritter inherits the Critter's act method. |