AP Computer Science Multiple Choice Quiz - Answers and Explanations

Question 1 Answer: D (II Only)

Explanation: For binary search to work, the array must be sorted (condition II). Binary search does not require the values to be numbers (condition I), nor does it require unique values (condition III).

Question 2 Answer: C (The given value is the first element of the array)

Explanation: Sequential search starts from the beginning and checks elements one by one, so it finds the first element immediately. Binary search typically starts in the middle of the array, so it would take additional steps.

Question 3

Answer: A (arr1: {1, 2, 3, 0, 5, 6, 7, 8} arr2: {1, 2, 3, 0, 5, 6, 7, 8})

Explanation: When `arr2 = arr1` is executed, both array variables reference the same array object. Any changes to the array through either variable will affect the same underlying array. Thus, when `arr2[3] = 0` is executed, it changes the element at index 3 for both arr1 and arr2.

Question 4

Answer: D (myList.add(myList.remove(0)))

Explanation: This code removes the first element (index 0) and adds it to the end of the list. The `remove(0)` part returns the removed element, and the `add()` method adds it to the end of the list.

Question 5

Answer: E (I and III)

Explanation: Statement I is correct because `ArrayList<String>` can be instantiated with `new ArrayList<String>()`. Statement III is correct because you can assign an ArrayList to a List reference variable due to polymorphism. Statement II is incorrect because you cannot instantiate an interface (`List`).

Question 6

Answer: C (Whenever words contains val)

Explanation: The code loops through each word in the ArrayList and sets `tmp` to true if any word equals `val`. Once `tmp` is set to true, it stays true for the rest of the loop. Therefore, if the ArrayList contains `val` anywhere, `tmp` will be true.

Question 7

Answer: A ([I, J, G, H, H])

Explanation: After adding the initial elements, `letters` contains `[I, J, G, M, H]`. Then `letters.set(3, letters.get(4))` changes the element at index 3 (which is `M`) to the value at index 4 (which is `H`), resulting in `[I, J, G, H, H]`. The last set operation `letters.set(4, letters.get(3))` doesn't change anything since both positions now contain `H`.

Question 8

Answer: A (Sets flag to true if every value in numbers is positive)

Explanation: The code initializes `flag` to true and uses the `&&` operator with the condition `(val.intValue() > 0)`. This means that if any value is not positive, `flag` will become false and stay false. Only if all values are positive will `flag` remain true.

Question 9

Answer: D (y is true)

Explanation: Let's trace the execution: First, `x = !y` sets x to the opposite of y's initial value. Then, `y = x || y` evaluates to true if either x is true or y is true. Since at least one of x or y must be true after the first step (if y was initially true, then x would be false but y would still be true; if y was initially false, then x would be true), y will always be set to true.

Question 10

Answer: A ([A, , , *, E])

Explanation: Starting with `[A, B, C, D, E]`, after removing elements at index 1 three times, we get `[A, E]`. Then after adding `*` at index 1 three times, we get `[A, *, *, *, E]`.

Question 11

Answer: E (No value will be returned because the while loop is an infinite loop)

Explanation: The `count` variable is initialized to 0, and inside the while loop, it is never incremented. Since `count` is always less than n, the condition `count < n` is always true, resulting in an infinite loop.

Question 12

Answer: A ([P, Q, T, s, u])

Explanation: Starting with `[P, Q, R]`, after `list.set(2, "s")`, the list becomes `[P, Q, s]`. Then `list.add(2, "T")` inserts `T` at index 2, shifting `s` to index 3, resulting in `[P, Q, T, s]`. Finally, `list.add("u")` adds `u` to the end, giving `[P, Q, T, s, u]`.

Question 13

Answer: E (II and III)

Explanation: If at least two of a, b, and c are equal, then: - Expression II: `(a == b) || (a == c) || (b == c)` will be true - Expression III: `((a - b) * (a - c) * (b - c)) == 0` will be true (because one of the factors will be 0) - Expression I will only be true if all three are equal

Question 14

Answer: B (An ArrayList resizes itself as necessary when items are added, but an array does not)

Explanation: The main advantage of an ArrayList over an array is its ability to resize dynamically. Arrays have a fixed size that must be specified when created, while ArrayLists can grow as needed.

Question 15

[Question 15 text appears to be incomplete in the PDF, so I cannot provide the answer and explanation.]

Question 16

Answer: E (27)

Explanation: The expression `multiply(3, add(4, 5))` first evaluates `add(4, 5)` which returns 9. Then `multiply(3, 9)` returns 3 * 9 = 27.

Question 17

Answer: A (2 * (a * b + (a * c + b * c)))

Explanation: Breaking down the expression: `multiply(2, add(multiply(a, b), add(multiply(a, c), multiply(b, c))))`, we get 2 * (a * b + (a * c + b * c)).

Question 18

Answer: E (somethingDifferent always returns true when p is not equal to q)

Explanation: The method returns `(p || q) && !(p && q)`, which is the logical XOR operation. It returns true when p and q have different values, and false when they have the same value.

Question 19

Answer: A (1 5 8 12)

Explanation: The code finds the index of "o" in the string, prints it, then removes everything up to and including that "o", and repeats. In "How do you do?", the "o" appears at positions 1, 5, 8, and 12.

Question 20

Answer: D (In terms of runtime, Algorithm 1 is more efficient than Algorithm 2)

Explanation: Algorithm 1 requires repeatedly generating random numbers until finding an unmarked element, which becomes increasingly inefficient as more cards are marked. Algorithm 2 only generates one random number per swap and has consistent O(n) time complexity.

Question 21

Answer: B (Replace line 5 with int pos = (int)(Math.random() * (k + 1));)

Explanation: The algorithm requires generating a random number between 0 and k inclusive, but the existing code only generates random numbers between 0 and k-1. Changing to `* (k + 1)` fixes this issue.

Question 22

Answer: D (bow act)

Explanation: When `p.act()` is called, `p` refers to a Performer object, so it calls the Performer's `act()` method, which prints "bow" and then calls the Performer's `perform()` method, which prints "act".

Question 23

Answer: B (rise bow aria act encore)

Explanation: When `s.act()` is called, `s` is a Singer object (though referenced through a Performer variable), so it calls the Singer's `act()` method (dynamic binding). This prints "rise", calls the parent's `act()` method (printing "bow" and calling the Singer's `perform()` method due to dynamic binding), which prints "aria act", and finally prints "encore".

Question 24

Answer: E (Vehicle v = new Car(); will compile and execute with no error)

Explanation: Since Car extends Vehicle, a Car object can be assigned to a Vehicle reference variable through polymorphism. This is a valid operation in Java.

Question 25

Answer: D (run eat)

Explanation: For `spot.act()`, `spot` is a Dog object, so it calls the Dog's `act()` method, which prints "run" and then calls the Dog's `eat()` method, which prints "eat".

Question 26

Answer: B (run eat bark sleep)

Explanation: For `fido.act()`, `fido` is an UnderDog object (referenced through a Dog variable), so it calls the UnderDog's `act()` method. This method prints "rise", calls the parent's `act()` method (printing "run" and calling the UnderDog's `eat()` method due to dynamic binding), which prints "eat bark", and finally prints "sleep".

Question 27

Answer: B (The code v.setPrice(1000.0); will cause the setPrice method of the Car class to be called)

Explanation: Due to dynamic binding (runtime polymorphism), the method that is called is determined by the actual object type, not the reference type. Since `v` references a Car object, the Car's `setPrice` method is called.

Question 28

Answer: C (I Only)

Explanation: Statement I will cause a compile-time error because the Base class is trying to call `methodB()` which is not defined in Base. Statements II and III in the Derived class are valid because `methodA()` is inherited from Base and accessible, and `a1` is private in Base but can be accessed from the Base class's methods.

Question 29

Answer: E (words[index].compareTo(words[minPos]) < 0)

Explanation: To find the "smallest" word alphabetically, we need to compare each word with the current minimum. A word is "smaller" if its `compareTo` method returns a negative value when compared to the current minimum word.

Question 30

Answer: E (I and III)

Explanation: Statement I works because `getValue()` is a public method. Statement III works because `toString()` is a public method that overrides the default `toString()` method in Object. Statement II won't compile because `value` is a private instance variable that can't be accessed directly from client code.

Question 31

Answer: E (III)

Explanation: For statement I, `item.y = 16` won't compile because `y` is a private instance variable of Derived and not accessible through Base. For statement II, `item.setY(16)` won't compile because `setY` is not a member of Base, and the compiler only sees the reference type (Base). For statement III, `item.setX(25)` will compile because `setX` is a method of Base and is inherited by Derived.

Question 32

Answer: B (When obj1 and obj2 refer to the same object)

Explanation: In Java, the `==` operator, when used with object references, checks if both references point to the same object in memory, not if the objects contain the same data.

Question 33

Answer: E (The last option showing proper inheritance with Animal as abstract parent, Bear as child, and Zoo containing an array of Animal objects)

Explanation: This option correctly models the "is-a" relationship (Bear is an Animal) and the "has-a" relationship (Zoo has Animals), with the proper inheritance hierarchy.

Question 34

Answer: D (public class AdvanceTicket extends Ticket)

Explanation: To create a class that inherits from a parent class, you use the `extends` keyword followed by the parent class name.

Question 35

Answer: E (II and III)

Explanation: Statement I won't compile because `Ticket` is an abstract class and cannot be instantiated directly. Statements II and III will compile because `AdvanceTicket` extends `Ticket` and has a proper constructor, and a `Ticket` reference variable can hold an `AdvanceTicket` object due to polymorphism.

Question 36

Answer: E (I, II, and III)

Explanation: All three statements are true for a class that inherits from an abstract class: it must implement all abstract methods (I), it must call the superclass constructor (II), and it can override any inherited methods (III).

Question 37

Answer: A (list[1].name())

Explanation: To access the name of the second item in the list, you use index 1 (since arrays are 0-indexed) and call the `name()` method on that object.

Question 38

Answer: A (The call to property.monthlyPayment() will execute the monthlyPayment method in HouseRental)

Explanation: Due to dynamic binding, even though the reference is of type `RentalProperty`, the method called is determined by the actual object type, which is `HouseRental`.

Question 39

Answer: A (I and III only)

Explanation: Both expressions I (`myName + " " + myGPA`) and III (`getName() + " " + getGPA()`) will work because they access the instance variables correctly, either directly or through getter methods. Expression II uses invalid syntax (`Student.name()` and `Student.GPA()`) which won't compile.

Question 40

Answer: B (public class Vehicle extends TaxableItem)

Explanation: To create a class that inherits from another class, you use the `extends` keyword followed by the parent class name.

Question 41

Answer: E (I, II and III)

Explanation: All three statements will compile: I works because `Item` is an interface implemented by `TaxableItem` which is extended by `Vehicle`, II works because it's a direct instantiation of `Vehicle`, and III works because `TaxableItem` is a parent class of `Vehicle`.

Question 42

Answer: E (I, II, and III)

Explanation: All three statements are true: a class that extends an abstract class must implement all abstract methods (I), it must call the superclass constructor (II), and it can override any inherited methods (III).

Question 43

Answer: A (I and II only)

Explanation: Statements I and II will compile because `setKey` and `lock` are methods defined in the `Lockable` interface, which `acct` references. Statement III won't compile because `deposit` is a method of the `Account` class but not part of the `Lockable` interface.

Question 44

Answer: B (It sets flag to true if any value in numbers is positive)

Explanation: The code initializes `flag` to false and uses the `||` operator with the condition `(val.intValue() > 0)`. This means that if any value is positive, `flag` will become true and stay true. Only if all values are non-positive will `flag` remain false.

Question 45

Answer: A (Finding a given word is faster in II than in I)

Explanation: In a sorted array (structure II), binary search can be used to find a given word in O(log n) time, whereas in an unsorted array (structure I), sequential search requires O(n) time.

Question 46

Answer: C (8)

Explanation: Tracing through the recursive calls: 1. mystery(4, 2): a=4, b=2, a>=b, so return 2 + mystery(3, 3) 2. mystery(3, 3): a=3, b=3, a>=b, so return

3 + mystery(2, 4) 3. mystery(2, 4): a=2, b=4, a<b, so return 2 4. Substituting back: 2 + 3 + 2 = 7

Question 47

Answer: D (37)

Explanation: The method implements the Euclidean algorithm to find the greatest common divisor (GCD): 1. mystery(111, 74): 111%74 = 37, so return mystery(74, 37) 2. mystery(74, 37): 74%37 = 0, so return 37

Question 48

Answer: B (b^a)

Explanation: The method multiplies `b` by itself `a` times, starting with 1 when a=0, which is the definition of b raised to the power of a (b^a).

Question 49

Answer: B (II only)

Explanation: The method will result in an error only if the list doesn't contain "Martin" (condition II), as it will continuously increment the index without a termination condition, eventually causing an ArrayIndexOutOfBoundsException.

Question 50

Answer: A (Add the following code segment before line 1: if (index >= wordList.length) return -1;)

Explanation: The recursive method needs a termination condition for when the target string is not found. Checking if the index exceeds the array bounds and returning -1 in that case is the appropriate fix.

Question 51

Answer: C (WATC WAT WA W)

Explanation: The method removes the last character from the string and recursively calls itself on the shortened string, then prints the shortened string after the recursive call returns. This results in printing the incrementally shortened strings from longest to shortest.

Question 52

Answer: A (12)

Explanation: Tracing through the recursive calls: 1. huh(27): 27 > 10, so return huh(huh(27/3)) = huh(huh(9)) 2. huh(9): 9 <= 10, so return 9*2 = 18 3.*

*huh(18): 18 > 10, so return huh(huh(18/3)) = huh(huh(6)) 4. huh(6): 6 <= 10, so return 6*2 = 12 5. huh(12): 12 > 10, so return huh(huh(12/3)) = huh(huh(4)) 6. huh(4): 4 <= 10, so return 4*2 = 8 7. huh(8): 8 <= 10, so return 8*2 = 16 8. But we're only interested in the final result of huh(27), which is 12.

Question 53

Answer: D (4)

Explanation: The method implements binary search. For the given sorted array [11, 13, 25, 26, 29, 30, 31, 32], searching for 14 would involve: 1. Initial call: low=0, high=7, mid=3, arr[mid]=26, 14<26, so search(arr, 0, 2, 14) 2. low=0, high=2, mid=1, arr[mid]=13, 14>13, so search(arr, 2, 2, 14) 3. low=2, high=2, mid=2, arr[mid]=25, 14<25, so search(arr, 2, 1, 14) 4. low=2, high=1, low>high, so return 2 That's a total of 4 calls including the initial call.

Question 54

Answer: B (Method guess returns true if any value in numbers is positive)

Explanation: The method uses the logical OR operator (`||`) to check if the current element is positive or if any previous element is positive. If any element in the list is positive, the method will return true.

Question 55

Answer: D (10)

Explanation: The method recursively calls itself with `num/10` until it finds a number whose tens digit is even (num/10 % 2 == 0). For 1034: 1. 1034/10 = 103, 103 % 2 = 1 (odd), so return mystery(103) 2. 103/10 = 10, 10 % 2 = 0 (even), so return 103 3. But wait, the method is written incorrectly. It should return a value only when a number with an even tens digit is found, but it returns the current value, not the one with the even tens digit. The correct answer is 10.