



Student Performance Q&A:

2006 AP[®] Computer Science AB Free-Response Questions

The following comments on the 2006 free-response questions for AP[®] Computer Science AB were written by the Chief Reader, David Reed of Creighton University in Omaha, Nebraska. They give an overview of each free-response question and of how students performed on the question, including typical student errors. General comments regarding the skills and content that students frequently have the most problems with are included. Some suggestions for improving student performance in these areas are also provided. Teachers are encouraged to attend a College Board workshop to learn strategies for improving student performance in specific areas.

Question 1

What was the intent of this question?

This question focused on manipulating `Sets` and `Maps` in order to provide the functionality of a thesaurus. The skeleton of a `Thesaurus` class was provided, with a private data field of type `Map` (mapping words to `Sets` of synonyms). In part (a) students were required to complete the `addSynonym` method, which took a word/synonym pair and added a corresponding entry into the `Map`. This involved checking whether an entry for that word already existed, initializing a new entry if necessary, accessing the current `Set` of synonyms associated with the word, and adding the new synonym to that `Set`. In part (b) students were required to complete the `removeSynonym` method, which took a synonym and removed every entry containing that synonym from the `Map`. This involved iterating through all words in the `Map`, accessing each corresponding synonym `Set`, testing whether that `Set` contained the desired synonym, and removing each matching synonym. In solving this problem, students demonstrated a solid understanding of the `Map` and `Set` collections and effectively utilized these collections in organizing and accessing data.

How well did students perform on this question?

This question was comparable in difficulty to similar collections-oriented questions on previous exams. Student performance was excellent, with the second highest mean score on the exam: 6.04 out of 9 possible points. The distribution of scores was skewed to the high end, with many scores in the 7–9 range. Scores between 0 and 6 were fairly evenly distributed, and there were few blanks on this question. This distribution suggests that stronger students had little difficulty with the question, and even weaker students were prepared and perhaps even expecting a question of this form.

What were common errors or omissions?

This question had a great number of students with near canonical solutions. The most common errors were due to confusion over the types of stored values. Many students would attempt to store a single word as a value in the `Map`, instead of a singleton `Set` containing that word. Likewise, students would get a value from the `Map` and treat it as if it were a single word. In part (b) many student errors involved incorrect attempts to iterate through all of the synonym sets. Some students failed to get the initial `keyset` to iterate through or even attempted indexing (using `[]`) to iterate through the `Map` and/or `Set`. When constructing the `Set` of affected words in part (b), many students tried to instantiate a new `Set` instead of specifying a `HashSet` or `TreeSet`.

Based on your experience of student responses at the AP Reading, what message would you like to send to teachers that might help them to improve the performance of their students on the exam?

Continue to emphasize the Java Collection classes. Students need to know how to construct collections and use methods to access and update the collections. The Quick Reference Sheet, provided on the exam and available at AP Central, is a valuable reference that students should be encouraged to use (both on the exam and in class). While this question did not involve any performance restrictions, previous questions of this type have required students to know the Big-Oh performance of basic operations and to select from alternative collections based on performance. Finally, the difference between interfaces (e.g., `Set` and `List`) and classes (e.g., `HashSet` and `LinkedList`) needs to be better understood by students.

Question 2

What was the intent of this question?

This question focused on students' ability to design a hierarchy of classes using inheritance. A `Product` interface was provided, along with an `Item` class that implemented the interface. The `Item` class contained private data fields for storing a product description and a unit price, a constructor for initializing the fields, accessor methods for retrieving the description and unit price, and a mutator method for changing the price. In part (a) students were required to design and implement the `Pack` class, which also implemented the `Product` interface. A `Pack` could store an arbitrary quantity of a `Product` and so needed private fields for the `Product` and its quantity. The class also required a constructor (to initialize the fields) and the `getPrice` method (to calculate the price of the `Pack` by multiplying the `Product` price by its quantity). In part (b) students were required to design and implement a `Bundle` class, which also implemented the `Item` interface. A `Bundle` could store an arbitrary collection of `Products` and so needed a private field for the collection. The class also required a constructor (to initialize the field), the `getPrice` method (to iterate through the collection and calculate the total cost of all `Products`), and an `add` method (to add a new `Product` to the collection). In addition to testing whether the students could design and implement classes from scratch, this question measured their understanding of polymorphism. Each class implemented the `Product` interface and contained fields of type `Product`, which allowed for arbitrary structures such as `Bundles of Bundles` containing `Packs of Bundles`.

How well did students perform on this question?

This question was easier than similar design questions on previous exams. While the polymorphism aspect added an interesting wrinkle, the methods that the students were required to write were algorithmically simple. For the `Pack` class, the `getPrice` calculation merely involved multiplying

the `Product` price with its quantity. For the `Bundle` class, the `getPrice` calculation did require traversing the `Product` collection, but the summing up of prices was a fairly standard code sequence for AP students. As a result, student performance was very high on this question, with the highest mean score on the exam: 7.21 out of 9 possible points. There was an extremely high number of scores in the 7–9 range, suggesting that strong students had no trouble with the design aspect of this problem. Scores between 1 and 8 were fairly evenly distributed, although still skewed to the higher end of that range. There were very few zeros and blanks, suggesting that even weaker students were able to complete some of the design and receive partial credit.

What were common errors or omissions?

The most common error in both parts (a) and (b) was in failing to correctly implement the desired polymorphism. The constructor in the `Pack` class needed to take a `Product` as parameter and store that `Product` in a private field. Many students specified an `Item` instead, and some even used a `String`. Likewise, in part (b) many students specified an `Item`, `Pack`, or `String` as parameter to the `add` method and stored entries of that type. These incorrect definitions destroy the ability for `Packs` and `Bundles` to store any type of `Products`, including `Packs` and `Bundles`. Most students used an `ArrayList` (or just `List`) as the type of the underlying collection in `Bundles`, although some did use `Sets` and even `Maps` with varying degrees of success. A significant number of students failed to declare the collection field as `private`, which caused a half-point deduction. A more general design error that occurred in some solutions was failing to store the `Products` in the `Bundle` and instead accumulating their prices to a sum as they were added. While this approach would work in some cases, it would produce wrong price totals if the price of any item were changed after having been added to a `Bundle`. Since the problem statement explicitly referred to this possibility and required `getPrice` to return the correct total, this design approach was penalized.

Based on your experience of student responses at the AP Reading, what message would you like to send to teachers that might help them to improve the performance of their students on the exam?

Design questions such as this one, which require the student to make choices in the creation of classes or data structures, are likely to appear on future exams. While this question focused almost exclusively on design and polymorphism, students should be aware that design may be combined with more complicated algorithms and performance analysis on future exams. The role of polymorphism in design, particularly in creating general-purpose collections from a class hierarchy, should be understood and appreciated by students.

Question 3

What was the intent of this question?

This question focused on linked-list manipulation and abstraction. A `WaitingList` class that had a linked-list of `ListNodes` and a node count as private fields was provided. In part (a) students were required to complete the `private findKth` method, which returned the `ListNode` at a specified index of the list. This involved traversing the links in the list, keeping track of a counter, and returning the desired `ListNode` when reached. In part (b) students were required to complete the `transferNodesFromEnd` method, which transferred a specified number of `ListNodes` from the end of a different list onto the current one. This involved locating the desired `ListNode` in the other list (using the `findKth` method from part (a)), linking that node to the end of the current list, resetting

the end of the other list, and updating the node counts for both lists. While the amount of code required to complete these tasks was modest, doing so required a solid understanding of linked structures and their manipulation.

How well did students perform on this question?

This question was comparable in difficulty to previous linked-structures questions. While it did not require recursion, it did involve a significant amount of detail in correctly disconnecting/connecting the lists and updating node counts. Student performance was very good, with a mean score of 5.25 out of 9 possible points. There were relatively few 9s, suggesting that even strong students had a difficult time getting all of the details correct. However, a majority of the scores fell in the 5–8 range, suggesting that most students were able to produce partial solutions. For many of these students, part (a) proved to be more straightforward and a relatively easy source of 3 points, while part (b) earned only partial credit. This question did have the largest number of zeros and blanks on the exam, suggesting that weaker students may have been scared off by the question.

What were common errors or omissions?

Errors in part (a) tended to be minor. They included using the front reference to traverse the list (and thus losing the original front), attempting to use an iterator on the linked structure, and off-by-one errors when finding and returning the `ListNode`. The most common error, by far, came in part (b) and centered on misunderstanding the meaning of private access. In order to update the list and node count of the other `WaitingList`, private fields of that list needed to be accessed and the private method `getKth` needed to be called. Many students wrongly believed that private fields and methods are completely inaccessible, even to instances of the same class. As such, they either commented that a solution was impossible or else added accessor and mutator methods that drastically changed the problem. Other common errors included failing to set the front of the `other` list to `null` if all nodes were removed, and failing to update the node counts for both lists.

Based on your experience of student responses at the AP Reading, what message would you like to send to teachers that might help them to improve the performance of their students on the exam?

In addition to using high-level Java Collection classes, students are still expected to be able to manipulate low-level linked structures, using `ListNode`, `TreeNode`, or similar node structures. From this particular question, it is clear that many students did not fully understand how private access works. They need to realize that instances of a class can access private fields and methods of other instances of the class. This is not a bug or a unique feature of Java—many methods (e.g., `compareTo`, `equals`) would be impossible to write without this feature. This also raises a general point of advice. If students are unsure about how to do a particular part of a question, they should make an attempt and go on to complete the rest of their solution. Some students simply did not answer part (b) because they thought the private access made a solution impossible. If they had instead written solutions to the parts they understood, they could have obtained substantial partial credit.

Question 4

What was the intent of this question?

This question was based on the Marine Biology Simulation (MBS) case study and focused on abstraction, recursion, and algorithm implementation. Students needed to show their understanding of the case study and its interacting classes in order to recursively search for a path in an `Environment`. A skeleton of the `PathFinder` class was provided, which contained an `Environment` as a private field. A recursive algorithm for finding an NE-path between two `Locations` in the `Environment` was described, and students were required to implement that search algorithm. In part (a) students were required to complete the `private possibleEnds` method, which determined whether two `Locations` could be endpoints of a NE-path. This involved checking to see that both `Locations` were valid and that the ending `Location` was north and/or east of the starting `Location`. In part (b) students were required to complete the `findNEPath` method, which performed the recursive search between the two `Locations` and returned a path if one existed. This involved checking base cases (the two `Locations` were the same, or one was nonempty), performing recursive searches from neighboring locations, and collecting the path in a `List of Locations` when a path was found. While the algorithm for conducting the recursive search was specified, there was considerable freedom in the implementation. Determining whether two `Locations` were possible endpoints in part (a) was most commonly accomplished by comparing row and column numbers but also could be accomplished using the `getDirection` method from `Environment`. Likewise, the recursion in part (b) could be expanded forwards from the starting `Location` (using a north or east neighbor) or backwards from the ending `Location` (using a south or west neighbor).

How well did students perform on this question?

This was a difficult question, combining recursion with a complex algorithm that had to be understood and implemented. As with most recursion questions, the amount of code in the solution was modest, but a correct solution required a deep understanding of recursion and attention to detail in handling all cases. While the mean score was the lowest on the exam, 4.68 out of 9 possible points, this still represents a strong showing by most students. The distribution of scores was fairly even, skewed toward the middle range. There were few 1s and zeros, suggesting that even weaker students were able to produce partial solutions that earned some credit. A large number of blanks, however, suggests that many students were either intimidated by the complex algorithm description or ran out of time before reaching this question.

What were common errors or omissions?

Most responses in part (a) featured the structure of the canonical solution. The most common errors involved malformed method calls (e.g., leaving out the object, `theEnv`), failing to check whether the end `Locations` were empty (e.g., calling `isValid` instead of `isEmpty`), and forgetting to check for equality when comparing rows or columns. Some students interpreted the `possibleEnds` method as requiring there to be an actual path between the two `Locations` and thus performed the recursive search as in part (b). Correct solutions of this type were not penalized. In part (b) a variety of errors occurred with the recursion. Many students made only one recursive call, or placed their recursive calls at the wrong places in the method (e.g., at the beginning of the method, before the base cases). There were many syntax errors on recursive calls, as students failed to provide the correct number of parameters or specified parameters of the wrong types. Finally, many students declared the path `List` but either failed to initialize it or failed to add `Locations` to it.

Based on your experience of student responses at the AP Reading, what message would you like to send to teachers that might help them to improve the performance of their students on the exam?

Recursion continues to be a difficult topic for many students, especially when used in a nonstandard way as in this question. Recursion needs to be seen as more than just a way of traversing a tree or implementing quick sort. Questions such as this one show the power of recursion as a general problem-solving tool and should continue to be emphasized in classes. In general, students should be prepared for exam questions like this one, where a complex algorithm is described and they are asked to implement it. Also, this question demonstrates that case study questions need not utilize the same context but instead can use familiar classes from the case study in other application areas.