CollegeBoard
Advanced Placement Program

Student Performance Q&A:

2004 AP® Computer Science AB Free-Response Questions

The following comments on the 2004 free-response questions for AP® Computer Science AB were written by the Chief Reader, Chris Nevison of Colgate University. 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.

General Comments

The new exam introduced several new ideas. With Java as the programming language, an object-oriented approach to programming was expected. Most questions involved writing code within the context of class definitions for interacting objects. Students had to understand how to call methods for interacting objects from different classes. The use of methods is fundamental, so students were expected to call the appropriate method and not rewrite equivalent code.

A new type of question involving the design of classes was introduced on this exam (question 1). Students needed to understand inheritance and polymorphism to answer these questions correctly. For full credit students had to create a good design that followed the specification given in the problem, including choosing data structures that met time-complexity (big-O) requirements.

This exam also required familiarity with several data structures implemented by classes in the java.util library and defined by interfaces for the AP Computer Science program. Question 2 involved `Sets` and `Maps` from the java.util library, and question 4 involved an implementation of the `PriorityQueue` interface.

Although new types of questions were added, the exam did not have a question that required students to work with a linked list created from the `ListNode` class; it did have a question working with a binary tree using the `TreeNode` class. It is likely that future exams will have one or the other of these dynamic structures, but not necessarily both.

Some general scoring principles are described on the "General Usage" sheet. Students were writing a draft solution under time constraints, so we did not penalize minor errors that did not reflect on the students' understanding. For example, confusion about the use of `length`, `length()`, and `size()`

for accessing the length of a `String`, array, `List`, `Set`, or `Map` were not penalized. Other errors on the usage sheet indicate penalties taken if the error was not covered specifically in the grading rubric. Some minor errors were not penalized, because a newer version of the language (Java 1.5) that allows different syntax is now available. These non-penalized errors include failure to downcast when removing objects from a `List`, `Set`, or `Map` and failure to correctly convert between primitive types and their wrapper classes (e.g., `int` and `Integer`).

Question 1

What was the intent of this question?

This was a design question. Part (a) required students to define an interface, `LibraryItem`, given a description of its specification. Part (b) required students to define a class, `LibraryBook`, which extends the given class `Book` and implements the `LibraryItem` interface. This required students to recognize those methods that needed to be defined to implement the interface and those that did not need to be defined because they were inherited. Part (c) required students to select a data structure for storing `LibraryItem` objects so that three methods could be implemented efficiently. Students had to understand the time-complexity for adding items to, or accessing items in, different structures, so that an appropriate choice (a `Map`) could be made. Students were also required to state the time-complexity for the methods in big-O terms.

How well did students perform on this question?

Very few students gave no response to this question, and the scores were spread evenly over the nine-point range, with a bias toward the top end. Students seemed to have a good understanding of what was needed. In a design question of this nature, there are no algorithmic complexities where students lose points in the details of the implementation, so students who understood how to define the interface and class scored well. The mean score was 5.2 out of 9.

What were common student errors or omissions?

In Part (a) students often erred by putting instance variables, constructors, or implementations of methods into the interface. Sometimes they included a method that could change the ID, thus violating the specification; they lost points for putting things into their code that did not belong. Students also sometimes failed to include one or more of the methods needed to satisfy the specification, particularly the accessor methods.

In Part (b) students would often fail to call `super` in the constructor to set the `Book` instance variables. In fact, students would sometimes declare their own local variables for the book author and title, as well as those needed to implement the `LibraryItem` interface. They would then override the methods `getAuthor` and `getTitle` inherited from the `Book` class. Although this could work in terms of returning the right values, it creates an inheritance hierarchy that may fail if changes are made in the base class; therefore, it's not a good design. Sometimes students would lose points for declaring instance variables as public rather than private or for not providing the needed instance variables (one for ID, one for holder) and the methods to implement their `LibraryItem` interface.

In Part (c) many students recognized that a map, with ID as the key, was the correct choice (either `TreeMap` or `HashMap` were acceptable). However, many students were not familiar with the appropriate application of a map and chose another data structure. Students who chose a map usually also got the correct complexities (all O(1) for a `HashMap` and all O(log n) for a `TreeMap`). Students who chose another data structure had mixed results on the complexity part of the question, missing those points more often than not.

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?

Teach students the difference between an interface, an abstract class, and a concrete (non-abstract) class and what can and *cannot* be included in each. In particular, an interface can only specify methods and cannot have a constructor nor any instance variables defined. The methods are automatically abstract and cannot be implemented.

Emphasize how inheritance works and explain that a subclass should not override (redefine) a method that is inherited unless it is intended to work differently. Teach how a constructor in a subclass should call `super` appropriately in order to set instance variables defined in the `super` class.

Demonstrate where and when the various data structures should be used, both those from the java.util library (Lists, Sets, Maps) and the standard stacks, queues, and priority queues. The time-complexity of different operations for these structures is an important topic.

Question 2

What was the intent of this question?

This question asked the student to write a constructor and methods for working with a data structure that is a `Map`, with `String` objects as keys (candidate names), and `Integers` as values. Students had to put the data into the `Map` in the constructor by iterating over a List of Sets and then iterating over each of those Sets. Then they had to iterate over the key set for the map to find those candidates that tied for the maximum number of votes, placing them into a `Set` that was returned. Thus, students had to be comfortable with Maps and Sets and using Iterators. Finally, students were asked to state the time-complexity for their implementation of the `candidatesWithMost` method.

How well did students perform on this question?

This question was relatively difficult, as there was considerable detail for students to manage. About 1,000 students (out of 5,800) had no-response or zero scores. Still, many students who had clearly worked with sets and maps scored well, with 30% of students scoring 7, 8, or 9. The mean score was 4.1 out of 9.

What were common student errors or omissions?

Many students left out the needed downcasts when accessing elements in a `Set` or `Map` and did not correctly convert between `int` and `Integer` and vice-versa. However, these errors were not penalized, so students were really tested on their understanding of the data structures and the algorithms, not on the syntactic details.

Students would make errors on iterating through a `List` or `Set`. Some students clearly had not worked very much with Iterators. In Part (a) they would sometimes fail to recognize the nested structure, iterating only through the `List` and not the `Set`. Students also missed the logic of checking whether a candidate had already been added to the `Map` and adding a new entry, or updating the existing entry, accordingly.

In Part (b) some students would re-implement the code to find the maximum number of votes, the functionality of the `maxVotes` method. This can work, but it is poor programming practice and did not receive full credit. Some students would incorrectly use `itr.next()` within the loop, as if it returned the same element on two different calls. Students would sometimes err by calling an iterator method for a `Map`. In Part (a) students would often try to instantiate `Map` and in Part (b), `Set`, rather than one of

the implementing classes for each of these interfaces.

In Part (c) students who did not specify the type of either the `Map` (`TreeMap` or `HashMap`) or the `Set` returned in Part (b) did not receive the point for this part, since the correct answer depended on that choice. Sometimes students would miss this point if they selected either `TreeMap` or `TreeSet` and incorrectly indicated a complexity of O(C). Students also missed this point if they repeatedly called `maxVotes` within the loop in Part (c), making the complexity quadratic.

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?

Be sure that students work with the `Set` and `Map` data structures in various ways and that they learn how to use Iterators to traverse these structures. Students should understand the time-complexity of the different operations on these structures and how these complexities combine in an algorithm that loops over one of these structures.

Question 3

What was the intent of this question?

This question tested the students' knowledge of the Marine Biology Simulation case study. It also tested understanding of inheritance, as the students were required to define a new subclass of the `Fish` class. The `PredatorFish` class needed a new instance variable to keep track of the number of steps since it last ate. This required students to understand how to correctly define a constructor for a subclass (something that was modeled very well for them in the case study itself). Part (b) asked students to define a new method for the `PredatorFish,` which required understanding the relationships among the existing classes of the case study and their methods. Part (c) required the student to override the `act` method and demonstrate understanding of the logic of the new `act` and how to access the same method in the superclass.

How well did students perform on this question?

In the past, a significant number of students did not attempt the case study question, indicating that they hadn't studied it adequately in their school course. This year there were fewer no-response and zero scores on this question than on either question 2 or 4, indicating that only those students who were poorly prepared for the exam did not attempt the question. It seems that teachers are teaching the case study much better than they did in the past.

Students did very well on this question, showing a good understanding of the case study and how to extend the Fish class. It was the easiest question on the exam and could have appeared on the Computer Science A exam. The mean score was 5.6 out of 9.

What were common student errors or omissions?

In Part (a) students would sometimes fail to call `super` correctly in the constructor.

In Part (b) students would use the `Environment isEmpty` method to check whether food was in the location ahead; that was incorrect, since `isEmpty` returns `false` for a location outside the environment, as well as for a location where there is a fish. Sometimes students would fail to return the correct Boolean in all cases.

Some students updated the variable checking steps since it last ate within the `eat` method rather than in the `act` method, but either way earned credit if the update was correct. Students often updated one case

and not the other.

In Part (c), many students failed to include the check `"if( isInEnv()) return;"` which was necessary to prevent a `PredatorFish` that was eaten by another from being resurrected. Students also made errors in the logic of the conditionals for determining when `super.act()` should be called. Finally, some students re-implemented the code for `super.act()` rather than making the method call. Although this could work, it would violate the intended inheritance relationship.

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?

Teach the MBS case study thoroughly (most teachers have been doing this). The case study is a good opportunity to demonstrate the interactions of objects of different classes in a large program. It is also a good opportunity to teach inheritance using the `Fish` class as an example. The relationship of the subclass to the superclass (in particular, the use of `super`) are important elements of this. Students must be taught not to reimplement code when a method is already provided with the same functionality.

Question 4

What was the intent of this question?

This question asked the student to implement a priority queue with data objects stored multiple times by building a binary search tree containing `Item` objects containing the data objects and number of repetitions. The binary search tree was built using `TreeNode` objects. Students were required to write the `peekMin` method that must traverse to the leftmost node in the tree and return the data stored there. They were also required to write the method `addHelper` that added an object to the priority queue; `addHelper` was set up so that it could be done recursively, but it also could be done iteratively. In either case, it required a standard search of a binary tree, then attaching a new node or updating a node that already contained the object. The binary tree involved two layers of abstraction for storing the data; the `TreeNode` objects contained `Item` objects that contained the data object and a count.

How well did students perform on this question?

This was the hardest question on this exam: 1,200 students (out of 5,800) had no-response or zero scores. Among other students the scores were spread quite evenly across the nine-point range, with somewhat fewer at the top end. The mean score was 3.7 out of 9.

What were common student errors or omissions?

In Part (a) students simply needed to traverse to the leftmost node. However, many students tried to do some other traversal of the tree structure. Both in Part (a) and Part (b), students sometimes did not understand the two-level access (`node.getValue().getData()`) needed to access an object in the priority queue.

In Part (b) students would often forget the base case or empty tree case in the recursion. The general case included three cases: data equal required an increment of the count, and data greater or less required recursion or traversal to the right or left. In either case, the recursive calls were often made but not accompanied by the needed `setLeft` or `setRight` to attach the returned node back into the tree. (Note: A simple recursive call could work if there were a “look-ahead” where the new node was attached using `setLeft` or `setRight` only when `getLeft` or `getRight` returned `null`; this approach usually failed to handle the empty-tree case.)

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?

Show students how to build dynamic structures such as binary trees (using `TreeNode`) or linked lists (using `ListNode`). The fact that all parameters are passed by value means that the logic for recursively adding to a dynamic structure is different from languages such as C++ and Pascal that have reference parameters; explain this logic to students. In the future, some of the manipulation of binary trees is likely to be handled with polymorphism, and teachers should be aware of this new approach with the object-oriented paradigm. Traversals of, insertions into, and deletions from these structures are important topics in the AB curriculum.