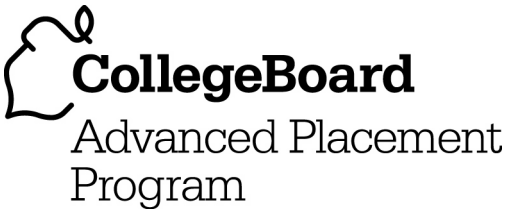

AP® Computer Science AB
2004 Scoring Commentary

AP® COMPUTER SCIENCE AB
2004 SCORING COMMENTARY

Question 1

Sample: A
Score: 9

Parts (a) and (b) are correct. A `HashMap` as the choice for the data representation with id as the key for the `HashMap` was a correct answer for part (c). The expected Big Oh efficiencies given by the student are consistent with the `HashMap` choice.

Sample: B
Score: 5

Part (a) is correct. In part (b), the student loses the correctness point for the constructor header, and since the instance variables are declared `public`, the student loses the instance variable declaration. Credit is given for an attempt on the constructor because `myID` is correctly set but loses this correctness point. In part (c) the choice of `HashSet` is not correct so no points are earned for data representation and organization. Although the student correctly lists the Big Oh efficiency for `add`, points are lost for `checkout` and `getHolder` efficiencies.

Sample: C
Score: 2.5 becomes 3

In part (a) the student earned points for correctly including interface header and headers for required methods. However, the student incorrectly included data fields and a constructor header. In part (b), the class header is correct and the student earned the attempt point for the constructor. There were point deductions for omitting the instance variable declarations and the method implementations. The student also lost ½ point for failure to implement the methods that were included in the part (a) interface. The choice of a `HashSet` for the data representation was incorrect for part (c). The expected Big Oh efficiency listed in part (c) for `add` was correct for the `HashSet` choice, but those listed for `checkout` and `getHolder` were not correct.

.

Question 2

Sample: A
Score: 9

Parts (a) and (b) are correct. The missing cast of `voteCount.keySet()` to a `Set` in part (b) was ignored. The student has used a `HashMap` for `voteCounts` in part (a) and a `HashSet` for the candidate set in part (b), and the complexity given in part (c) correctly reflects these choices.

Sample: B
Score: 7

In part (a), the student fails to initialize `voteCount`. The `ballotList` and ballot iterations are correct, as is the check for the candidate's name. The student loses the increment point for attempting to use the post-increment operator on a method return value, and the create point for putting a 0 rather than a 1 into the `Map`.

Part (b) is correct. The student iterates over the Map's `keySet`, using the iterator `remove` method to remove non-winning candidates. This has the side effect of removing the corresponding key-value pairs from the `Map`. Since the problem did not explicitly state that the `Map` should not be altered, this approach was considered acceptable. Finally, the student retrieves the altered `keySet` from the `Map` and returns it.

Since the types of the `Map` in part (a) and the `Set` in part (b) are unknown, the student is ineligible for the complexity point in part (c).

Sample: C
Score: 3

In part (a), the student fails to initialize `voteCount`. The student earns attempt but not correct for the `ballotList` loop, since he or she refers to `list` rather than `ballotList`. The student earns attempt but not correct for ballot iteration, since the iterator is not initialized properly (an instance of an interface cannot be created using `new`). The student earns the check, increment, and create points; automatic conversions between `ints` and `Integers` were assumed, since this will become unnecessary with the introduction of Java 1.5.

No points are earned on part (b). Since the types of the `Map` in part (a) and the `Set` in part (b) are unknown, the student is ineligible for the complexity point in part (c).

Question 3

Sample: A Score: 9

In part (a), the solution uses two `int` instance variables to count the `turnsOfFasting` and `fastingLimit`, which are both initialized in the body of the constructor, after `super(env,loc)`. (The use of two instance variables was not the norm.)

For part (b), the student declares and initializes an `Environment` object `env`, to save writing when accessing the `Environment` class functions: a `Location oneInFront` and a `Locatable prey`. If `prey` is `null`, then the `boolean` value `false` is returned; otherwise `prey` is cast as a `Fish` so that the `die` method from the `Fish` class can be used and the value `true` is returned.

In part (c), the student first checks to see whether this predator fish is in bounds—if not, return. The `turnsOfFasting` is either reset to zero or incremented depending upon the Boolean value returned from the `eat` method. If `turnsOfFasting` equals or exceeds `fastingLimit`, the `die` method is called, then return. If this predator fish is still alive, `super.act` is called.

Sample: B Score: 6

For part (a), the student correctly writes the class header and declares the private instance variable, but loses both half points since the constructor contains no body.

In part (b), the student attempts the most common method for determining whether there is a fish to eat by using the `isEmpty` method from the `Environment` class, correctly returning `false` when `isEmpty` returns `true` (the predator did not eat). Unfortunately the student does not understand that when `isEmpty` returns `false`, the location either contains an object or is invalid so both must be checked. The student attempts to cast a location as a `Locatable` and remove it from the environment. Student does receive credit for returning the correct `boolean` in this case, even though the testing and removing is incorrect.

For part (c), the student checks to see whether this predator fish is in bounds, assigns the value returned by the `eat` method to a Boolean variable, correctly resets or increments the instance variable, and tests for starvation. The only deductions on this part were for reimplementing instead of calling `super.act()`.

Sample: C Score: 4

Part (a) received no points since the class header is missing `extends`, the instance variable is missing the modifier `private`, and the constructor has no body.

In part (b), the student attempts to use the `isEmpty` method without also using `isValid`; this student uses `env` everywhere `environment()` should be used, without declaring and initializing it so the "missing declaration & initialization" usage point is deducted. The student correctly initializes a `boolean` variable to `false`, then assigns `true` to it in the context of eating and returns the Boolean variable.

For part (c) the student fails to set the instance variable back to zero when the call to eat returns true, and the student reimplements the code for `super.act()` rather than calling it.

Question 4

Sample: A
Score: 8.5

Part (a) is correct. The test of `if (root == null)` is not necessary and will not execute as per the precondition. No points are deducted for innocuous code.

In part (b), the student's base case is checked with `if (t == null)`, but the student incorrectly returns `t` without making a `new TreeNode`. The student can earn credit for an attempt to create the new `TreeNode` in the "looking ahead" case further in the code. The lines following the tests for `if (t.getRight() == null)` and `if (t.getLeft() == null)` each correctly set the new nodes. The failure to return from the base case is the student's only error. The use of the `.equals( )` method with `Comparable` data is allowed.

Sample: B
Score: 5

In part (a), the student correctly starts at the root of the tree but does not traverse in any direction. Since credit for returning data is awarded only if there is an attempt to traverse to the leftmost node, no credit is awarded for the return or the traversing.

In part (b), the student checks for the base case correctly and attempts to make a new `TreeNode`. The return is not correct because the student fails to make a `new Item(obj)` in the parameter list. The student accesses the data correctly with calls to `getValue()` and `getData()`. The test for equality and call to `incrementCount()` is also correct. The calls to `addHelper(obj, t.getLeft())` and `addHelper(obj, t.getRight())` correctly traverse down the tree and since they are called without a return, the linking of the tree is preserved. The student fails to set the new `TreeNode` with `t.setLeft(…)` and `t.setRight(…)` statements, so will not earn the 1 point for the correct set.

Sample: C
Score: 3

In Part (a), the student shows an attempt to start at the root. Since `isEmpty` is used as the control structure and root is never updated, the loop does not iterate or terminate. This results in no attempt point for the traverse, which then results in no return points for this part.

In Part (b), the student correctly checks for `t == null` but does not return a new `TreeNode` with an `Item` object. Since the student does not use a look ahead process, no credit is earned for the base case. The student uses the `getData()` and `getValue()` methods to attempt to access the data; however since they are in the wrong order, only ½ point is earned. The test of equality is correct, but the `incrementCount` method needs to be invoked by an `Item` instead of a `TreeNode`. Thus the student only earns ½ point in that section. The test for traversal direction is correct, and although the recursive calls are made both left and right, the student does not correctly set the new `TreeNode` with `t.setLeft()` and `t.setRight()` calls, so no point is earned for set. Since `t` was modified by the recursive calls, the return point is not earned.